instead be made as additional or updated software routines in the mobile equipment which is adapted to function as a service platform. If the service logic is saved in the mobile equipment, extra memory is also needed.

In the subscriber identity module USIM, the extra memory possibly needed for implementing the invention is limited to a small amount sufficient for saving the extra authentication data, the possible verification data and the decryption algorithms possibly needed.

It will be understood that the above description and the figures related to it are only presented for the purpose of illustrating the present invention. The various modifications and variations of the invention will be obvious to those skilled in the art without departing from the scope or spirit of the invention disclosed in the attached claims.

CLAIMS

1. A method for preventing unauthorized use of a service in a mobile communication system, in which method

a service request is received from a user of the service, and

the service is generated by means of a service logic,

characterized in that in the method

authentication data is appended to the service logic (3-6),

the user requesting the service is authenticated by means of the authentication data (3-9), and

the service logic is executed only if the authentication succeeds (3-14).

2. A method as claimed in claim 1, **characterized** in that

verification data of the service provider is also appended to the service logic,

the service provider is verified in connection with user authentication (3-13), and

the service logic is executed only if the verification also succeeds.

3. A method as claimed in claim 2, **characterized** in that

a first hash calculated from the service logic is used as verification data of the service logic,

the service logic is loaded onto a service platform where it is executed to generate the service,

the service provider is verified on the service platform by calculating a second hash from the service logic, and

if the first and the second hash are the same, the verification succeeds,

if the first and the second hash differ, the verification fails.

4. A method as claimed in claim 2, **characterized** in that

the signature of the service provider is used as the verification data,

the service logic is signed by encrypting it with the secret key of the service provider, and

the service provider is verified by decrypting the encryption of the service logic with the public key of the service provider.

5. A method as claimed in any one of the above claims, **characterized** in that

the secret key of the subscriber is saved in the subscriber identity module (USIM) of the user of the service,

the public key of the subscriber is used as the authentication data,

a challenge encrypted with the public key of the subscriber is sent to the subscriber identity module located in the mobile equipment of the user requesting the service (207),

the challenge is decrypted into plain text with the secret key of the subscriber in the identity module,

the plain text is received from the identity module (208),

a check is made to see if the unencrypted challenge and the plain text correspond to each other (209), and

if they correspond, the authentication succeeds, and

if they do not correspond, the authentication fails.

6. A method as claimed in claims 1, 2, 3 or 4, **characterized** in that

individual identity of the subscriber is used as the authentication data,

a service request is received from the user (3-1),

the individual subscriber identity related to the user is requested (3-7),

the requested identity is received (3-8),

a check is made to see if the authentication data and the requested identity correspond to each other (3-9), and

if they correspond, the authentication succeeds, and

if they do not correspond, the authentication fails.

7. A method as claimed in any one of the above claims, **characterized** in that

the service logic is loaded onto the service platform where it is executed to generate the service, and

the authentication data is appended to the service logic in connection with the loading.

8. A method as claimed in claim 7, **characterized** in that

the service logic, the authentication data appended to it, and the data indicating the user are saved on the service platform in connection with the loading (204),

a service request is received from the user,

a check is made to see if the service logic related to the requested service is saved on the service platform (201), and

if not, the service logic is loaded (203),

if yes,

- a check is made to see if authentication data has been saved for the user requesting the service (217), and

- if yes, the user is authenticated,

- if not,

-- authentication data is requested for the user (218),

-- the authentication data and the data indicating the user are saved in the service logic (220), and

-- the user is authenticated.

9. A telecommunication system comprising

a first part (SP) to produce the service for the user by means of a service logic, and

a second part to provide the service (SN, MT) to the user of the service,

in which system the first part (SP) is adapted to identify the user requesting the service and to check, if the service is subscribed to the user, and if the service is subscribed to the user, to generate the service by loading the service logic into the second part (SN, MT) which is adapted to provide the service by executing the loaded service logic,

characterized in that

the first part (SP) is adapted to append authentication data into the service logic being loaded for user authentication, and

the second part (SN, MT) is adapted to authenticate the user and to execute the service logic only in response to a successful authentication.

10. A system as claimed in claim 9, **characterized** in that

the first part (SP) is adapted to sign the service logic by encrypting it with an encryption key agreed with the second part, and

the second part (SN, MT) is adapted to verify the first part by decrypting the encryption of the service logic with a key corresponding to the agreed key and to execute the service logic only if the verification also succeeds.

11. A system as claimed in claim 9 or 10, **characterized** in that

the telecommunication system is a mobile communication system (IMT-2000) comprising at least one service provider and serving network,

the first part is the service provider (SP), and

the second part is the serving network (SN) comprising at least one network element (SCN).

12. A system as claimed in claim 9 or 10, **characterized** in that

the telecommunication system is a mobile communication system (IMT-2000) comprising at least one service provider (SP) and mobile terminal (MT) which is connected to the service provider through a serving network (SN) and which mobile terminal (MT) comprises in addition to actual mobile equipment (ME) a subscriber identity module (USIM) which is detachably connected to the mobile equipment,

the first part is the service provider (SP), and

the second part is the actual mobile equipment (ME).

13. A network element (SP) generating a telecommunication system service for a user, which produces the service by means of a service logic and which comprises means for identifying the user requesting the service and for checking if the service is subscribed to the user and for loading the service logic into the telecommunication system if the service is subscribed to the user,

characterized in that the network element (SP) comprises means for appending the authentication data to the service logic being loaded so that the user of the service is authenticated before the service logic is executed.

14. A network element (SP) as claimed in claim 13, **characterized** in that it comprises means for signing the service logic before it is loaded into the network.

15. A network element (SP) as claimed in claim 13 or 14, **characterized** in that it comprises a processor arranged to execute software routines, and said means have been implemented as software routines.

16. An apparatus of a telecommunication system, which apparatus comprises service logic executing means for providing a service from a service provider of a telecommunication system to a user of the service,

characterized in that the apparatus (SCN, ME) comprises

separation means for separating the authentication data of a user from a loaded service logic,

authentication means responsive to the separation means for user authentication, and

service logic execution means are adapted to be responsive to the authentication means.

17. An apparatus (SCN, ME) as claimed in claim 16, **characterized** in that

it comprises verification means for service provider verification by means of verification data in the loaded service logic, and

the service logic verification means are adapted to be responsive to the authentication means.

18. An apparatus (SCN, ME) as claimed in claim 16 or 17, **characterized** in that it comprises a processor arranged to execute software routines, and said means are implemented as software routines.

19. An apparatus as claimed in claim 16, 17 or 18, **characterized** in that it is a network element (SCN) of a mobile communication system, which is adapted to function as a service platform.

20. An apparatus as claimed in claim 16, 17 or 18, **characterized** in that it is the mobile equipment (ME) in a mobile communication system.

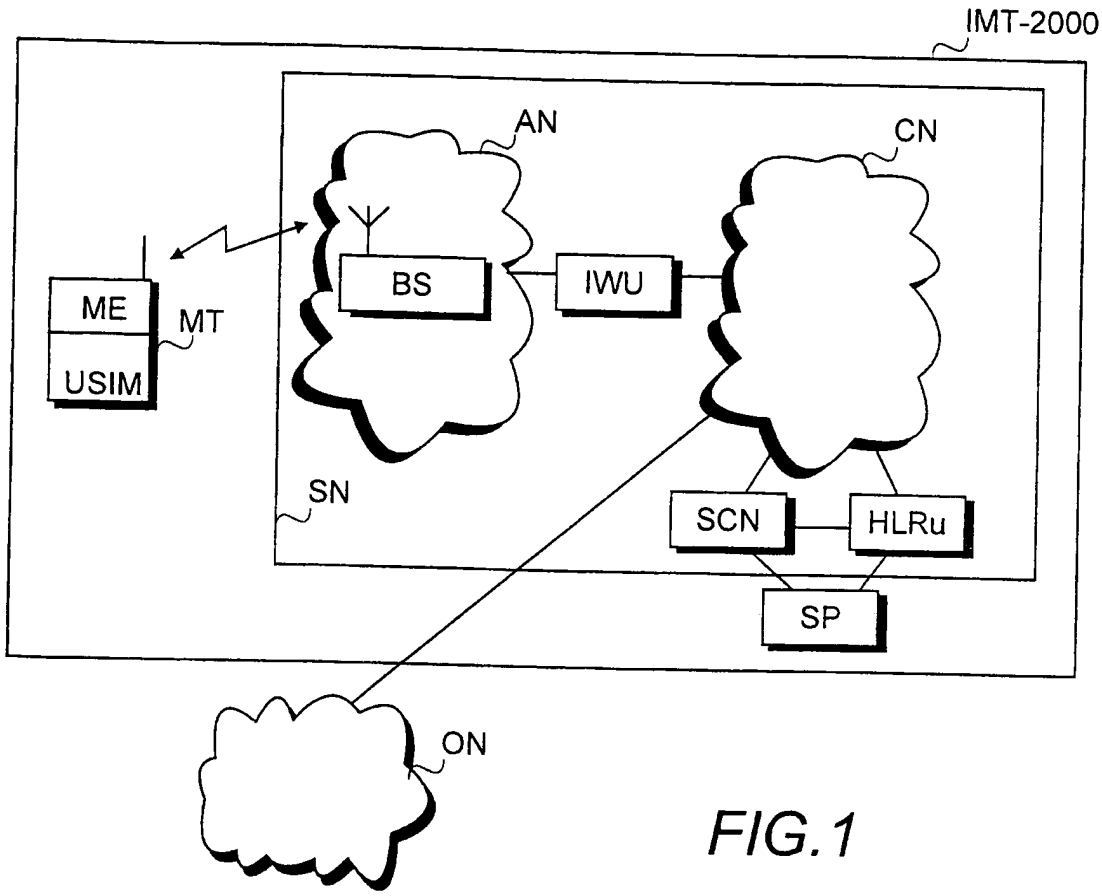


FIG.1

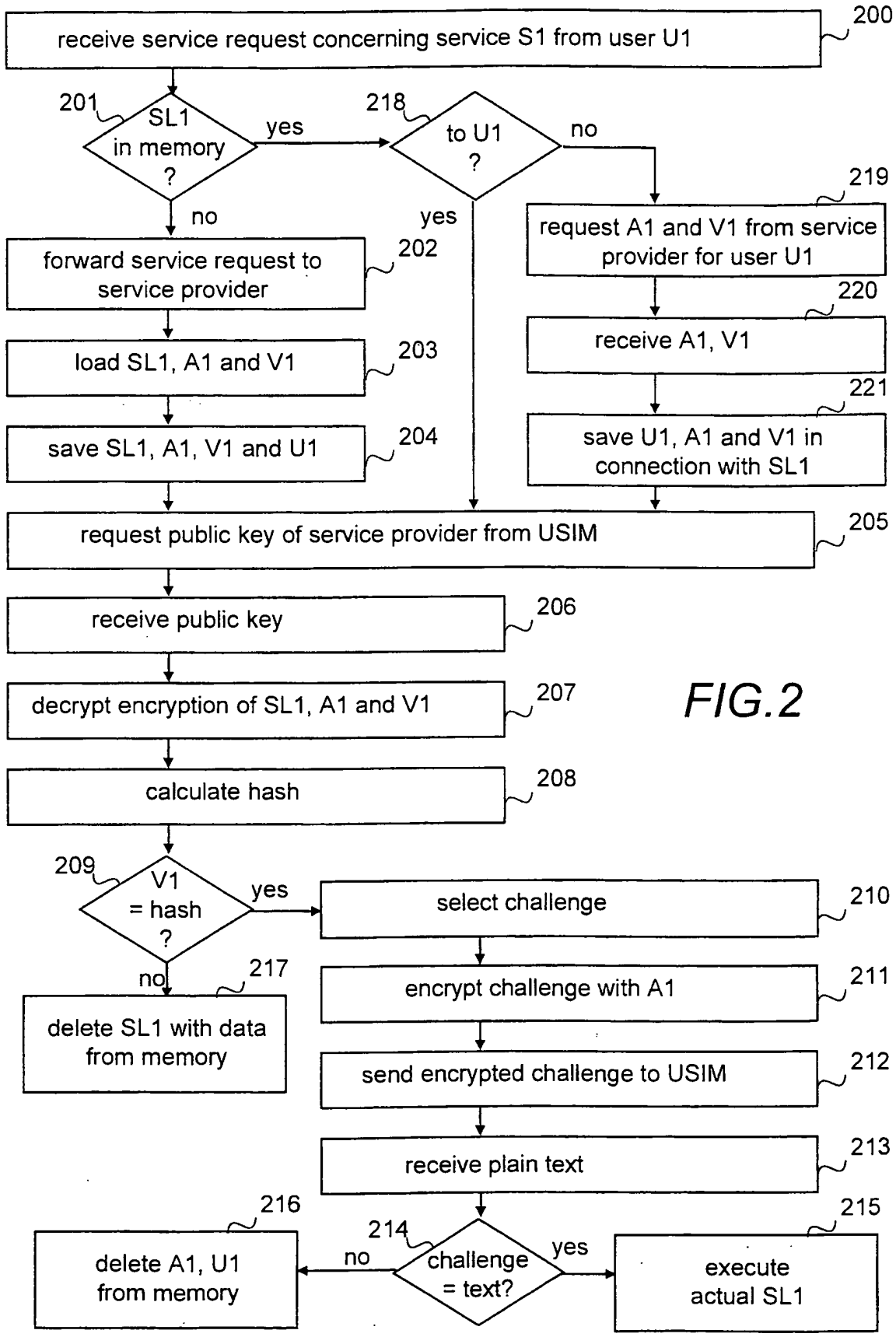
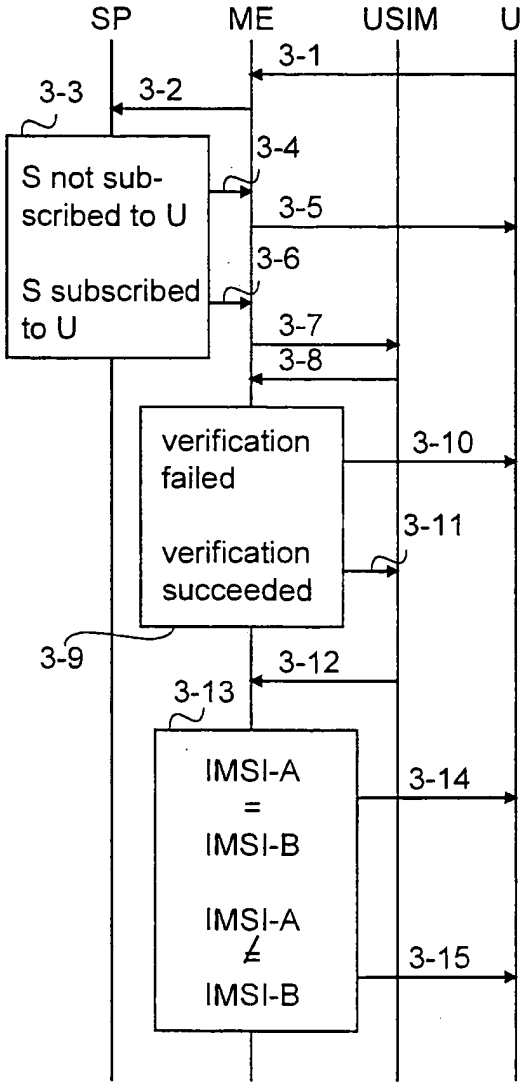

receive service request concerning service S1 from user U1
200
201
SL1 in memory ?
yes
218
to U1 ?
no
219
request A1 and V1 from service provider for user U1
no
yes
forward service request to service provider
202
220
receive A1, V1
load SL1, A1 and V1
203
221
save U1, A1 and V1 in connection with SL1
save SL1, A1, V1 and U1
204
request public key of service provider from USIM
205
receive public key
206
decrypt encryption of SL1, A1 and V1
207
calculate hash
208
209
V1 = hash ?
yes
select challenge
210
no
217
encrypt challenge with A1
211
delete SL1 with data from memory
send encrypted challenge to USIM
212
receive plain text
213
216
214
challenge = text?
no
yes
215
delete A1, U1 from memory
execute actual SL1

FIG.2



3-1. Service S
3-2. Service S to U

3-4. Negative acknowledgement
3-5. No service

3-6. SL, IMSI-A and hash
3-7. Give SP's public key
3-8. SP's public key

3-10. No service

3-11. Give IMSI

3-12. IMSI-B

3-14. Service

3-15. No service

FIG.3

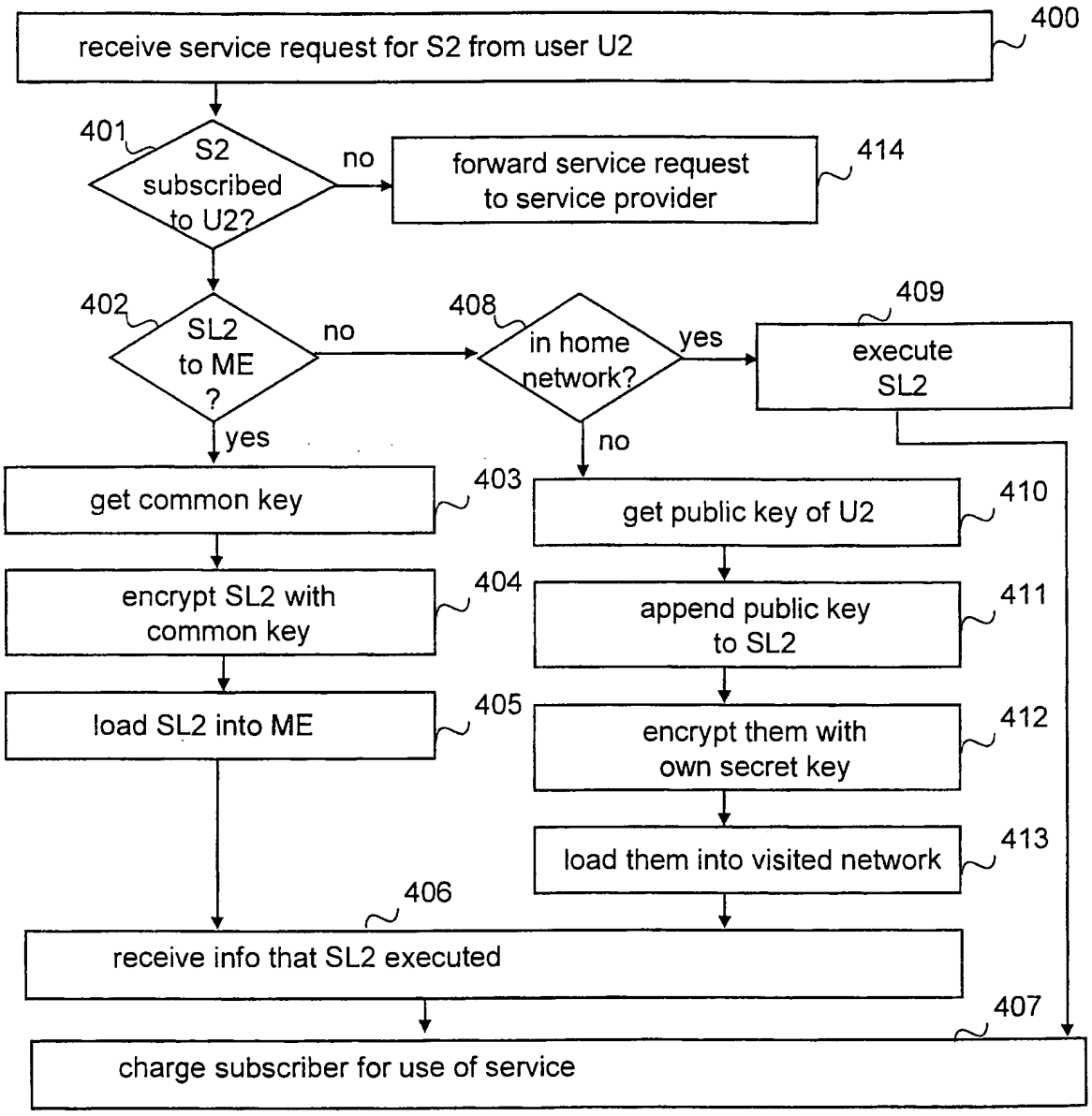

receive service request for S2 from user U2
400
401
S2 subscribed to U2?
no
forward service request to service provider
414
402
SL2 to ME ?
no
408
in home network?
yes
409
execute SL2
yes
no
get common key
403
get public key of U2
410
encrypt SL2 with common key
404
append public key to SL2
411
load SL2 into ME
405
encrypt them with own secret key
412
load them into visited network
413
406
receive info that SL2 executed
407
charge subscriber for use of service

FIG.4

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/30 — A1

(11) International Publication Number: WO 99/35782

(43) International Publication Date: 15 July 1999 (15.07.99)

(21) International Application Number: PCT/US98/27896

(22) International Filing Date: 31 December 1998 (31.12.98)

(30) Priority Data:

60/070,344	2 January 1998 (02.01.98)	US
60/089,529	15 June 1998 (15.06.98)	US

(71) Applicant: CRYPTOGRAPHY RESEARCH, INC. [US/US]; Suite 1088, 870 Market Street, San Francisco, CA 94102 (US).

(72) Inventors: KOCHER, Paul, C.; 143 Fillmore Street, San Francisco, CA 94117 (US). JAFFE, Joshua, M.; 21B Bird Street, San Francisco, CA 94110 (US).

(74) Agents: LAURIE, Ronald, S. et al.; Skadden, Arps, Slate, Meagher & Flom LLP, 525 University Avenue, Palo Alto, CA 94301 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

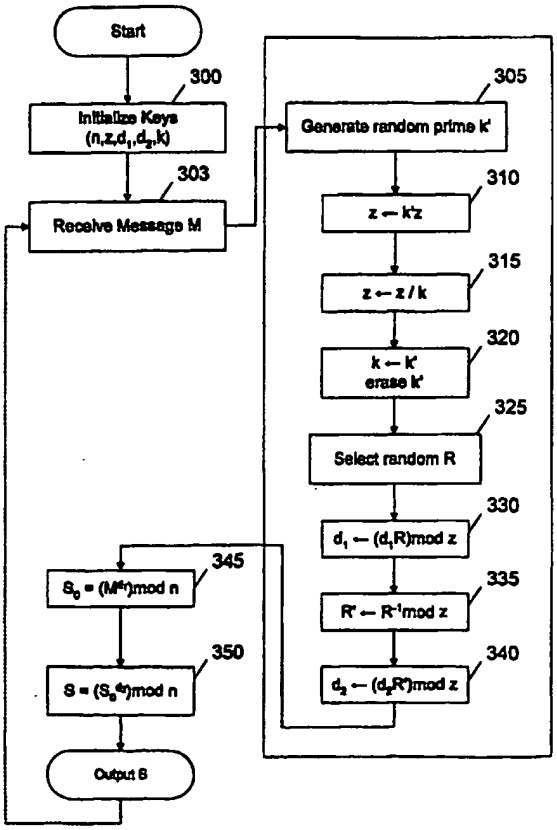
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: LEAK-RESISTANT CRYPTOGRAPHIC METHOD AND APPARATUS

(57) Abstract

The present invention provides a method and apparatus for securing cryptographic devices against attacks involving external monitoring and analysis. A "self-healing" property is introduced, enabling security to be continually re-established following partial compromises. In addition to producing useful cryptographic results, a typical leak-resistant cryptographic operation modifies or updates (330) secret key material in a manner designed to render useless any information about the secrets that may have previously leaked from the system. Exemplary leak-proof and leak-resistant implementations of the invention are shown for symmetric authentication (350), certified Diffie-Hellman (when either one or both users have certificates), RSA, ElGamal public key decryption (303).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

LEAK-RESISTANT CRYPTOGRAPHIC METHOD AND APPARATUS

This application claims the benefit of US Provisional Application No. 60/070,344 filed January 2, 1998, and US Provisional Application No. 60/089,529 filed June 15, 1998.

TECHNICAL FIELD

The method and apparatus of the present invention relate generally to cryptographic systems and, more specifically, to securing cryptographic tokens that must maintain the security of secret information in hostile environments.

BACKGROUND OF THE INVENTION

Most cryptosystems require secure key management. In public-key based security systems, private keys must be protected so that attackers cannot use the keys to forge digital signatures, modify data, or decrypt sensitive information. Systems employing symmetric cryptography similarly require that keys be kept secret. Well-designed cryptographic algorithms and protocols should prevent attackers who eavesdrop on communications from breaking systems. However, cryptographic algorithms and protocols traditionally require that tamper-resistant hardware or other implementation-specific measures prevent attackers from accessing or finding the keys.

If the cryptosystem designer can safely assume that the key management system is completely tamper-proof and will not reveal any information relating to the keys except via the messages and operations defined in the protocol, then previously known cryptographic techniques are often sufficient for good security. It is currently extremely difficult, however, to make hardware key management systems that provide good security, particularly in low-cost unshielded cryptographic devices for use in applications where attackers will have physical control over the device. For example, cryptographic tokens (such as smartcards used in electronic cash and copy protection schemes) must protect their keys even in potentially hostile environments. (A token is a device that contains or manipulates cryptographic keys that need to be protected from attackers. Forms in which tokens may be manufactured include, without limitation, smartcards, specialized encryption and key management devices, secure telephones, secure picture phones, secure web servers, consumer electronics devices using cryptography, secure microprocessors, and other tamper-resistant cryptographic systems.)

A variety of physical techniques for protecting cryptographic devices are known, including enclosing key management systems in physically durable enclosures, coating integrated circuits with special coatings that destroy the chip when removed, and wrapping devices with fine wires that detect tampering. However, these approaches are expensive, difficult to use in single-chip solutions (such as smartcards), and difficult to evaluate since there is no mathematical basis for their security. Physical tamper resistance techniques are also ineffective against some attacks. For example, recent work by Cryptography Research has shown that attackers can non-invasively extract secret keys using careful measurement and analysis of many devices' power consumption. Analysis of timing measurements or electromagnetic radiation can also be used to find secret keys.

Some techniques for hindering external monitoring of cryptographic secrets are known, such as using power supplies with large capacitors to mask fluctuations in power consumption, enclosing devices in well-shielded cases to prevent electromagnetic radiation, message blinding to prevent timing attacks, and buffering of inputs/outputs to prevent signals from leaking out on I/O lines. Shielding, introduction of noise, and other such countermeasures are often, however, of limited value, since skilled attackers can still find keys by amplifying signals and filtering out noise by averaging data collected from many operations. Further, in smartcards and other tamper-resistant chips, these countermeasures are often inapplicable or insufficient due to reliance on external power sources, impracticality of shielding, and other physical constraints. The use of blinding and constant-time mathematical algorithms to prevent timing attacks is also known, but does not prevent more complex attacks such as power consumption analysis (particularly if the system designer cannot perfectly predict what information will be available to an attacker, as is often the case before a device has been physically manufactured and characterized).

The present invention makes use of previously-known cryptographic primitives and operations. For example: U.S. patent 5,136,646 to Haber et al. and the pseudorandom number generator used in the RSAREF cryptographic library use repeated application of hash functions; anonymous digital cash schemes use blinding techniques; zero knowledge protocols use hash functions to mask information; and key splitting and threshold schemes store secrets in multiple parts.

SUMMARY OF THE INVENTION

The present invention introduces leak-proof and leak-resistant cryptography, mathematical approaches to tamper resistance that support many existing cryptographic primitives, are inexpensive, can be implemented on existing hardware (whether by itself or via software capable of running on such hardware), and can solve problems involving secrets leaking out of cryptographic devices. Rather than assuming that physical devices will provide perfect security, leak-proof and leak-resistant cryptographic systems may be designed to remain secure even if attackers are able to gather some information about the system and its secrets. This invention describes leak-proof and leak-resistant systems that implement symmetric authentication, Diffie-Hellman exponential key agreement, ElGamal public key encryption, ElGamal signatures, the Digital Signature Standard, RSA, and other algorithms.

One of the characteristic attributes of a typical leak-proof or leak-resistant cryptosystem is that it is "self-healing" such that the value of information leaked to an attacker decreases or vanishes with time. Leak-proof cryptosystems are able to withstand leaks of up to L_{MAX} bits of information per transaction, where L_{MAX} is a security factor chosen by the system designer to exceed to the maximum anticipated leak rate. The more general class of leak-resistant cryptosystems includes leak-proof cryptosystems, and others that can withstand leaks but are not necessarily defined to withstand any defined maximum information leakage rate. Therefore, any leak-proof system shall also be understood to be leak-resistant. The leak-resistant systems of the present invention can survive a variety of monitoring and eavesdropping attacks that would break traditional (non-leak-resistant) cryptosystems.

A typical leak-resistant cryptosystem of the present invention consists of three general parts. The initialization or key generation step produces secure keying material appropriate for the scheme. The update process cryptographically modifies the secret key material in a manner designed to render useless any information about the secrets that may have previously leaked from the system, thus providing security advantages over systems of the background art. The final process performs cryptographic operations, such as producing digital signatures or decrypting messages.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows an exemplary leak-resistant symmetric authentication method.

Figure 2 shows an exemplary leak-resistant Diffie-Hellman exponential key exchange operation.

Figure 3 shows an exemplary leak-resistant RSA private key operation.

Figure 4 shows an exemplary leak-resistant ElGamal signing operation.

DETAILED DESCRIPTION OF THE INVENTION

The sections following will describe an introduction to leak-proof/leak-resistant cryptography, followed by various embodiments of the general techniques of the invention as applied to improve the security of common cryptographic protocols.

I. Introduction and Terminology

The leakage rate L is defined as the number of bits of useful information about a cryptosystem's secrets that are revealed per operation, where an operation is a cryptographic transaction. Although an attacker may be able to collect more than L bits worth of measurement data, by definition this data yields no more than L bits of useful information about the system's secrets.

The implementer of a leak-proof system chooses a design parameter L_{MAX}, the maximum amount of leakage per operation the system may allow if it is to remain uncompromised. L_{MAX} should be chosen conservatively, and normally should significantly exceed the amount of useful information known to be leaked to attackers about the system's secrets during each transaction. Designers do not necessarily need to know accurately or completely the quantity and type of information that may leak from their systems; the choice of L_{MAX} may be made using estimates and models for the system's behavior. General factors affecting the choice of L_{MAX} include the types of monitoring potentially available to attackers, the amount of error in attackers' measurements, and engineering constraints that limit L_{MAX}. (Larger values of L_{MAX} increase memory and performance requirements of the device, and in some cases may increase L.) To estimate the amount of useful information an attacker could collect by monitoring a device's power consumption, for example, a designer might consider the amount of noise in the device's power usage, the power line capacitance, the useful time resolution for power consumption measurements, as well as the strength of the signals being monitored. Similarly, the designer knows that timing measurements can rarely yield more than a few bits of information per operation, since timing information is normally quantized to an integral number of clock cycles. In choosing L_{MAX}, the designer should assume that

attackers will be able to combine information gleaned from multiple types of attacks. If the leakage rate is too large (as in the extreme case where L equals the key size because the entire key can be extracted during a single transaction), additional design features should be added to reduce L and reduce the value needed for L_{MAX}. Such additional measures can include known methods, such as filtering the device's power inputs, adding shielding, introducing noise into the timing or power consumption, implementing constant-time and constant execution path algorithms, and changing the device layout. Again, note that the designer of a leak-resistant system does not actually need to know what information is being revealed or how it is leaked; all he or she need do is choose an upper bound for the rate at which attackers might learn information about the keys. In contrast, the designer of a traditional system faces the much harder task of ensuring that no information about the secrets will leak out.

There are many ways information about secrets can leak from cryptosystems. For example, an attacker can use a high-speed analog-to-digital converter to record a smartcard's power consumption during a cryptographic operation. The amount of useful information that can be gained from such a measurement varies, but it would be fairly typical to gain enough information to guess each of 128 key bits correctly with a probability of 0.7. This information can reduce the amount of effort required for a brute force attack. For example, a brute force attack with one message against a key containing k bits where each bit's value is known with probability p can be completed in

$$E(k,p)=\sum_{i=0}^{k}\left[\binom{k}{i}(1-p)^{i}p^{k-i}\left[\left(\sum_{j=0}^{i}\binom{k}{j}\right)-\frac{1}{2}\binom{k}{i}\right]+\frac{1}{2}\right]$$

operations. The reduction in the effort for a brute force attack is equivalent to shortening the key by $L = \log_2(E(k,½) / E(k,p)) = \log_2(k - E(k,p) - 1)$ bits. (For example, in the case of $k =$ 128 and $p = 0.7$, L is estimated to be about 11 bits for the first measurement. With a multiple message attack, the attacker's effort can fall to as low as $E(k,p)=\frac{1}{p^{k}}$.) Attackers can gain additional information about the keys by measuring additional operations; unless leak-resistance is used, finding the key becomes easy after just a few dozen operations.

When choosing L_{MAX}, a system designer should consider the signal-to-noise ratio of an attacker's measurements. For example, if the signal and noise are of roughly equivalent magnitude, the designer knows that an attacker's measurements should be incorrect about 25 percent of the time (e.g., $p = 0.75$ if only one observation per key bit is possible). Many

measurement techniques, such as those involving timing, may have signal-to-noise ratios of 1:100 or worse. With such systems, L is generally quite small, but attackers who can make a large number of measurements can use averaging or other statistical techniques to recover the entire key. In extreme cases, attackers may be able to obtain all key bits with virtually perfect accuracy from a single transaction (i.e., $L = k$), necessitating the addition of shielding, noise in the power consumption (or elsewhere), and other measures to reduce p and L. Of course, L_{MAX} should be chosen conservatively; in the example above where less than 4 useful bits are obtained per operation for the given attack, the designer might select $L_{MAX} = 64$ for a leak-proof design.

Leak-proof (and, more generally, leak-resistant) cryptosystems provide system designers with important advantages. When designing a traditional (i.e., non-leak-resistant and non-leak-proof) cryptosystem, a careful cryptosystem designer should study all possible information available to attackers if he or she is to ensure that no analytical techniques could be used to compromise the keys. In practice, many insecure systems are developed and deployed because such analysis is incomplete, too difficult even to attempt, or because the cryptographers working on the system do not understand or cannot completely control the physical characteristics of the device they are designing. Unexpected manufacturing defects or process changes, alterations made to the product by attackers, or modifications made to the product in the field can also introduce problems. Even a system designed and analyzed with great care can be broken if new or improved data collection and analysis techniques are found later. In contrast, with leak-proof cryptography, the system designer only needs to define an upper bound on the maximum rate at which attackers can extract information about the keys. A detailed understanding of the information available to attackers is not required, since leak-proof (and leak-resistant) cryptosystem designs allow for secret information in the device to leak out in (virtually) any way, yet remain secure despite this because leaked information is only of momentary value.

In a typical leak-proof design, with each new cryptographic operation i, the attacker is assumed to be able to choose any function F_i and determine the L_{MAX}-bit result of computing F_i on the device's secrets, inputs, intermediates, and outputs over the course of the operation. The attacker is even allowed to choose a new function F_i with each new operation. The system may be considered leak-proof with a security factor n and leak rate L_{MAX} if, after observing a large number of operations, an attacker cannot forge signatures, decrypt data, or

perform other sensitive operations without performing an exhaustive search to find an n-bit key or performing a comparable $O(2^n)$ operation. In addition to choosing L_{MAX}, designers also choose n, and should select a value large enough to make exhaustive search infeasible. In the sections that follow, various embodiments of the invention, as applied to improve the security of common cryptographic operations and protocols, will be described in more detail.

II. Symmetric Cryptographic Protocols

A. Symmetric Authentication

An exemplary cryptographic protocol that can be secured using the techniques of the present invention is symmetric authentication.

1. Conventional Symmetric Authentication

Assume a user wishes to authenticate herself to a server using an n-bit secret key, K, known to both the server and the user's cryptographic token, but not known to attackers. The cryptographic token should be able to resist tampering to prevent, for example, attackers from being able to extract secrets from a stolen token. If the user's token has perfect tamper resistance (i.e., L=0), authentication protocols of the background art can be used. Typically the server sends a unique, unpredictable challenge value R to the user's token, which computes the value $A = H(R \| K)$, where "||" denotes concatenation and H is a one-way cryptographic hash function such as SHA. The user sends A to the server, which independently computes A (using its copy of K) and compares its result with the received value. The user authentication succeeds only if the comparison operation indicates a match.

If the function H is secure and if K is sufficiently large to prevent brute force attacks, attackers should not be able to obtain any useful information from the (R, A) values of old authentication sessions. To ensure that attackers cannot impersonate users by replaying old values of A, the server generates values of R that are effectively (with sufficiently high probability) unique. In most cases, the server should also make R unpredictable to ensure that an attacker with temporary possession of a token cannot compute future values of A. For example, R might be a 128-bit number produced using a secure random number generator (or pseudorandom number generator) in the server. The properties of cryptographic hash functions such as H have been the subject of considerable discussion in the literature, and need not be described in detail here. Hash functions typically provide functionality modeled after a random oracle, deterministically producing a particular output from any input. Ideally, such functions should be collision-resistant, non-invertable, should not leak partial

information about the input from the output, and should not leak information about the output unless the entire input is known. Hash functions can have any output size. For example, MD5 produces 128-bit outputs and SHA produces 160-bit outputs. Hash functions may be constructed from other cryptographic primitives or other hash functions.

While the cryptographic security of the protocol using technology of the background art may be good, it is not leak-proof; even a one-bit leak function (with L=1) can reveal the key. For example, if the leak function F equals bit (R mod n) of K, an attacker can break the system quickly since a new key bit is revealed with every transaction where (R mod n) has a new value. Therefore, there is a need for a leak-proof/leak-resistant symmetric authentication protocol.

2. Leak-Resistant Symmetric Authentication

The following is one embodiment of a leak-resistant (and, in fact, also leak-proof) symmetric authentication protocol, described in the context of a maximum leakage rate of L_{MAX} bits per transaction from the token and a security factor n, meaning that attacks of complexity $O(2^n)$, such as brute-force attacks against an n-bit key, are acceptable, but there should not be significantly easier attacks. The user's token maintains a counter t, which is initialized to zero, and an $(n+2L_{MAX})$-bit shared secret K_t, which is initialized with a secret K_0. Note that against adversaries performing precomputation attacks based on Hellman's time/memory trade-off, larger values of n may be in order. Note also that some useful protocol security features, such as user and/or server identifiers in the hash operation inputs, have been omitted for simplicity in the protocol description. It is also assumed that no leaking will occur from the server. For simplicity in the protocol description, some possible security features (such as user and/or server identifiers in the hash operation inputs) have been omitted, and it is assumed that the server is in a physically secure environment. However, those skilled in the art will appreciate that the invention is not limited to such assumptions, which have been made as a matter of convenience rather than necessity.

As in the traditional protocol, the server begins the authentication process by generating a unique and unpredictable value R at step 105. For example, R might be a 128-bit output from a secure random number generator. At step 110, the server sends R to the user's token. At step 112, the token receives R. At step 115, the token increments its counter t by computing $t \leftarrow t + 1$. At step 120, the token updates K_t by computing $K_t \leftarrow H_K(t \| K_t)$, where H_K is a cryptographic hash function that produces an $(n+2L_{MAX})$ bit output from the old value

of K_t and the (newly incremented) value of t. Note that in the replacement operations (denoted "←"), the token deletes the old values of t and K_t, replacing them with the new values. By deleting the old K_t, the token ensures that future leak functions cannot reveal information about the old (deleted) value. At step 122, the token uses the new values of t and K_t to compute an authenticator $A = H_A(K_t \| t \| R)$. At step 125, the token sends both t and the authenticator A to the server, which receives them at step 130. At step 135, the server verifies that t is acceptable (e.g., not too large but larger than the value received in the last successful authentication). If t is invalid, the server proceeds to step 175. Otherwise, at step 140, the server initializes its loop counter i to zero and its key register K_t' to K_0. At step 145, the server compares i with the received value of t, proceeding to step 160 if they are equal. Otherwise, at step 150, the server increments i by computing $i \leftarrow i + 1$. At step 155, the server computes $K_t' \leftarrow H_K(i \| K_t')$, then proceeds back to step 145. At step 160, the server computes $A' = H_A(K_t' \| t \| R)$. Finally, at step 165, the server compares A and A', where the authentication succeeds at step 170 if they match, or fails at 175 if they do not match.

This design assumes that at the beginning of any transaction the attacker may have L_{MAX} bits of useful information about the state of the token (e.g., K_t) that were obtained using the leak function F in a previous operation. During the transaction, the attacker can gain an additional L_{MAX} bits of useful information from the token. If, at any time, any $2L_{MAX}$ (or fewer) bits of useful information about the secret are known to the attacker, there are still $(n+2L_{MAX})-2L_{MAX} = n$ or more unknown bits. These n bits of unknown information ensure that attacks will require $O(2^n)$ effort, corresponding to the desired security factor. However, the attacker should have no more than L_{MAX} bits of useful information about K_t at the end of the transaction. The property that attackers lose useful information during normal operation of the system is a characteristic of the leak-proof or leak-resistant cryptosystem. In general, this information loss is achieved when the cryptosystem performs operations that convert attackers' useful partial information about the secret into useless information. (Information is considered useless if it gives an attacker nothing better than the ability to test candidate values in an $O(2^n)$ exhaustive search or other "hard" operation. For example, if exhaustive search of X is hard and H is a good hash function, $H(X)$ is useless information to an attacker trying to find X.)

Thus, the attacker is assumed to begin with L_{MAX} bits of useful information about K_t before the token's $K_t \leftarrow H_K(t \| K_t)$ computation. (Initial information about anything other

than K_t is of no value to an attacker because K_t is the only secret value in the token. The function H_K and the value of t are not assumed to be secret.) The attacker's information can be any function of K_t produced from the previous operation's leaks.

3. Security Characteristics of Leak-Proof Systems

The following section provides a technical discussion of the security characteristics of the exemplary leak-proof system described above. The following analysis is provided as an example of how the design can be analyzed, and how a system may be designed using general assumptions about attackers' capabilities. The discussion and assumptions do not necessarily apply to other embodiments of the invention and should not be construed as limiting the scope or applicability of the invention in any way.

During the course of a transaction, the leak function F might reveal up to L_{MAX} information about the system and its secrets. The design assumes that any information contained in the system may be leaked by F, provided that F does not reveal useful new information about values of K_t that were deleted before the operation started, and F does not reveal useful information about values of K_t that will be computed in future operations. These constraints are completely reasonable, since real-world leaks would not reveal information about deleted or not-yet-existent data. (The only way information about future K_t values could be leaked would be the bizarre case where the leak function itself included, or was somehow derived from, the function H_K.) In practice, these constraints on F are academic and of little concern, but they are relevant when constructing proofs to demonstrate the security of a leak-proof system.

If the leak occurs at the beginning of the H_K computation, it could give the attacker up to $2L_{MAX}$ bits of useful information about the input value of K_t. Because K_t contains $(2L_{MAX}+n)$ bits of secret information and the attacker may have up to $2L_{MAX}$ bits of useful information about the initial value of K_t, there remain at least $(2L_{MAX}+n)-2L_{MAX} = n$ bits of information in K_t that are secret. The hash function H_K effectively mixes up these n bits to produce a secure new K_t during each transaction such that the attacker's information about the old K_t is no longer useful.

If the leak occurs at the end of the H_K computation, it could give an attacker up to L_{MAX} bits of information about the final value of H_K, yielding L_{MAX} bits of information about

the input to the subsequent transaction. This is not a problem, since the design assumes that attackers have up to L_{MAX} bits of information about K_t at the beginning of each transaction.

A third possibility is that the attacker's L_{MAX} bits of information might describe intermediates computed during the operation H_K. However, even if the attacker could obtain L_{MAX} new bits of information about the input to H_K and also L_{MAX} bits of information about the output from H_K, the system would be secure, since the attacker would never have more than $2L_{MAX}$ bits of information about the input K_t or more than L_{MAX} bits of information about the output K_t. Provided that L_{MAX} bits of information from within H_K cannot reveal more than L_{MAX} bits of information about the input, or more than L_{MAX} bits of information about the output, the system will be secure. This will be true unless H_K somehow compresses the input to form a short intermediate which is expanded to form the output. While hash functions whose internal states are smaller than their outputs should not be used, most cryptographic hash functions are fine.

A fourth possibility is that part or all of the leak could occur during the $A = H_A(K_t \| t \| R)$ calculation. The attacker's total "budget" for observations is L_{MAX} bits. If L_1 bits of leak occur during the H_K computation, an additional L_2 bits of information can leak during the $A = H_A(K_t \| t \| R)$ operation, where $L_2 \leq L_{MAX} - L_1$. If the second leak provides information about K_t, this is no different from leaking information about the result of the H_K computation; the attacker will still conclude the transaction with no more than L_{MAX} bits of information about K_t because $L_1 + L_2 \leq L_{MAX}$. However, the second leak could reveal information about A. To keep A secure against leaks (to prevent, for example, an attacker from using a leak to capture A and using A before the legitimate user can), the size of A should include an extra L_{MAX} bits (to provide security even if $L_2 = L_{MAX}$). Like H_K, H_A should not leak information about deleted or future values of K_t that are not used in or produced by the given operation. As with the similar assumptions on leaks from H_K, this limitation is primarily academic and of little practical concern, since real-world leak functions do not reveal information about deleted or not-yet-computed data. However, designers might be cautious when using unusual designs for H_A that are based on or derived from H_K, particularly if the operation $H_A(K_t \| t \| R)$ could reveal useful information about the result of computing $H_K(t \| K_t)$.

B. Other Leak-Resistant Symmetric Schemes

The same basic technique of updating a key (K) with each transaction, such that leakage about a key during one transaction does not reveal useful information about a key in a

subsequent (or past) transaction, can be easily extended to other applications besides authentication.

1. Symmetric Data Verification

For example and without limitation, leak-resistant symmetric data verification is often useful where a device needs to support symmetrically-signed code, data, content, or parameter updates (all of which will, as a matter of convenience, be denoted as "data" herein). In existing systems, a hash or MAC of the data is typically computed using a secret key and the data is rejected if computed hash or MAC does not match a value received with the data. For example, a MAC may be computed as HMAC(K, data), where HMAC is defined in "RFC 2104, HMAC: Keyed-Hashing for Message Authentication" by H. Krawczyk, M. Bellare, and R. Canetti, 1997. Traditional (non-leak-resistant) designs are often vulnerable to attacks including power consumption analysis of MAC functions and timing analysis of comparison operations.

In an exemplary leak-resistant verification protocol, a verifying device (the "verifier") maintains a counter t and a key K_t, which are initialized (for example at the factory) with $t \leftarrow 0$ and $K_t \leftarrow K_0$. Before the transaction, the verifier provides t to the device providing the signed data (the "signer"), which also knows K_0. The signer uses t to compute K_{t+1}' (the prime indicating a quantity derived by the signer, rather than at the verifier) from K_0 (or K_t' or any other available value of K_i'). using the relation $K_i' = H_K(i \| K_{i-1}')$, computes signature $S' = \text{HMAC}(K_{t+1}', \text{data})$, and sends S' plus any other needed information (such as data or t) to the verifier. The verifier confirms that the received value of t (if any) matches its value of t, and rejects the signature if it does not. If t matches, the verifier increments t and updates K_t in its nonvolatile memory by computing $t \leftarrow t + 1$ and $K_t \leftarrow H_K(t \| K_t)$. In an alternative embodiment, if the received value of t is larger than the internal value but the difference is not unreasonably large, it may be more appropriate to accept the signature and perform multiple updates to K_t (to catch up with the signer) instead of rejecting the signature outright. Finally, the verifier computes $S = \text{HMAC}(K_t, \text{data})$ and verifies that $S = S'$, rejecting the signature if S does not equal the value of S' received with the data.

2. Symmetric Encryption

Besides authentication and verification, leak-resistant symmetric cryptography can also be tailored to a wide variety of applications and environments. For example, if data

encryption is desired instead of authentication, the same techniques as were disclosed above may be used to generate a key K_t used for encryption rather than verification.

3. Variations in Computational Implementation

In the foregoing, various applications were disclosed for the basic technique of updating a key K_t in accordance with a counter and deleting old key values to ensure that future leakage cannot reveal information about the now-deleted key. Those skilled in the art will realize, however, that the exemplary techniques described above may be modified in various ways without departing from the spirit and scope of the invention. For example, if communications between the device and the server are unreliable (for example if the server uses voice recognition or manual input to receive t and A), then small errors in the signature may be ignored. (One skilled in the art will appreciate that many functions may be used to determine whether a signature corresponds – sufficiently closely -- to its expected value.) In another variation of the basic technique, the order of operations and of data values may be adjusted, or additional steps and parameters may be added, without significantly changing the invention. In another variation, to save on communication bandwidth or memory, the high order bits or digits of t may not need to be communicated or remembered. In another variation, as a performance optimization, devices need not recompute K_t from K_0 with each new transaction. For example, when a transaction succeeds, the server can discard K_0 and maintain the validated version of K_t. In another variation, if bi-directional authentication is required, the protocol can include a step whereby the server can authenticates itself to the user (or user's token) after the user's authentication is complete. In another variation, if the server needs to be secured against leaks as well (as in the case where the role of "server" is played by an ordinary user), it can maintain its own counter t. In each transaction, the parties agree to use the larger of their two t values, where the device with the smaller t value performs extra updates to K_t to synchronize t. In an alternate embodiment for devices that contain a clock and a reliable power source (e.g., battery), the update operation may be performed periodically, for example by computing $K_t \leftarrow H_K(t \parallel K_t)$ once per second. The token uses the current K_t to compute $A = H_A(K_t \parallel t \parallel R)$ or, if the token does not have any means for receiving R, it can output $A = H_A(K_t)$. The server can use its clock and local copy of the secret to maintain its own version of K_t, which it can use to determine whether received values of A are recent and correct. All of the foregoing show that the method and apparatus of the present invention can be implemented using numerous variations and modifications to

the exemplary embodiments described herein, as would be understood by one skilled in the art.

III. Asymmetric Cryptographic Protocols

The foregoing illustrates various embodiments of the invention that may be used with symmetric cryptographic protocols. As will be seen below, still other techniques of the present invention may be used in connection with asymmetric cryptographic operations and protocols. While symmetric cryptosystems are sufficient for some applications, asymmetric cryptography is required for many applications. There are several ways leak resistance can be incorporated into public key cryptosystems, but it is often preferable to have as little impact as possible on the overall system architecture. Most of the exemplary designs have thus been chosen to incorporate leak resistance into widely used cryptosystems in a way that only alters the key management device, and does not affect the certification process, certificate format, public key format, or processes for using the public key.

A. Certified Diffie-Hellman

Diffie-Hellman exponential key exchange is a widely used asymmetric protocol whereby two parties who do not share a secret key can negotiate a shared secret key. Implementations of Diffie-Hellman can leak information about the secret exponents, enabling attackers to determine the secret keys produced by those implementations. Consequently, a leak-resistant implementation of Diffie-Hellman would be useful. To understand such a leak-resistant implementation, it will be useful to first review a conventional Diffie-Hellman implementation.

1. Conventional Certified Diffie-Hellman

Typical protocols in the background art for performing certified Diffie-Hellman exponential key agreement involve two communicating users (or devices) and a certifying authority (CA). The CA uses an asymmetric signature algorithm (such as DSA) to sign certificates that specify a user's public Diffie-Hellman parameters (the prime p and generator g), public key (p^x mod g, where x is the user's secret exponent), and auxiliary information (such as the user's identity, a description of privileges granted to the certificate holder, a serial number, expiration date, etc.). Certificates may be verified by anyone with the CA's public signature verification key. To obtain a certificate, user U typically generates a secret exponent (x_U), computes his or her own public key $y_u = g^{x_u} \bmod p$, presents y_U along with any required auxiliary identifying or authenticating information (e.g., a passport) to the CA,

who issues the user a certificate C_u. Depending on the system, p and g may be unique for each user, or they may be system-wide constants (as will be assumed in the following description of Diffie-Hellman using the background art).

Using techniques of the background art, Alice and Bob can use their certificates to establish a secure communication channel. They first exchange certificates (C_{Alice} and C_{Bob}). Each verifies that the other's certificate is acceptable (e.g., properly formatted, properly signed by a trusted CA, not expired, not revoked, etc.). Because this protocol will assume that p and g are constants, they also check that the certificate's p and g match the expected values. Alice extracts Bob's public key (y_{Bob}) from C_{Bob} and uses her secret exponent (x_{Alice}) to compute $z_{\text{Alice}} = (y_{\text{Bob}})^{x_{\text{Alice}}} \bmod p$. Bob uses his secret exponent and Alice's public key to compute $z_{\text{Bob}} = (y_{\text{Alice}})^{x_{\text{Bob}}} \bmod p$. If everything works correctly, $z_{\text{Alice}} = z_{\text{Bob}}$, since:

$$\begin{aligned} z_{\text{Alice}} &= (y_{\text{Bob}})^{x_{\text{Alice}}} \bmod p \\ &= \left(g^{x_{\text{Bob}}}\right)^{x_{\text{Alice}}} \bmod p \\ &= \left(g^{x_{\text{Alice}}}\right)^{x_{\text{Bob}}} \bmod p \\ &= (y_{\text{Alice}})^{x_{\text{Bob}}} \bmod p \\ &= z_{\text{Bob}}. \end{aligned}$$

Thus, Alice and Bob have a shared key $z = z_{\text{Alice}} = z_{\text{Bob}}$. An attacker who pretends to be Alice but does not know her secret exponent (x_{Alice}) will not be able to compute $z_{\text{Alice}} = (y_{\text{Bob}})^{x_{\text{Alice}}} \bmod p$ correctly. Alice and Bob can positively identify themselves by showing that they correctly found z. For example, each can compute and send the other the hash of z concatenated with their own certificate. Once Alice and Bob have verified each other, they can use a symmetric key derived from z to secure their communications. (For an example of a protocol in the background art that uses authenticated Diffie-Hellman, see "The SSL Protocol Version 3.0" by A. Freier, P. Karlton, and P. Kocher, March 1996.)

2. Leak-Resistant Certified Diffie-Hellman

A satisfactory leak-resistant public key cryptographic scheme should overcome the problem that, while certification requires the public key be constant, information about the corresponding private key should not leak out of the token that contains it. In the symmetric protocol described above, the design assumes that the leak function reveals no useful information about old deleted values of K_t or about future values of K_t that have not yet been

computed. Existing public key schemes, however, require that implementations repeatedly perform a consistent, usually deterministic, operation using the private key. For example, in the case of Diffie-Hellman, a leak-resistant token that is compatible with existing protocols and implementations should be able to perform the secret key operation $y^x \bmod p$, while ensuring that the exponent x remains secret. The radical reshuffling of the secret provided by the hash function H_K in the symmetric approach cannot be used because the device should be able to perform the same operation consistently.

The operations used by the token to perform the private key operation are modified to add leak resistance using the following variables:

Register	Comment
x_1	First part of the secret key (in nonvolatile updateable memory)
x_2	Second part of the secret key (in nonvolatile updateable memory)
g	The generator (not secret).
p	The public prime, preferably a strong prime (not secret).

The prime p and generator g may be global parameters, or may be specific to individual users or groups of users (or tokens). In either case, the certificate recipient should be able to obtain p and g securely, usually as built-in constants or by extracting them from the certificate.

To generate a new secret key, the key generation device (often but not always the cryptographic token that will contain the key) first obtains or generates p and g, where p is the prime and g is a generator mod p. If p and g are not system-wide parameters, algorithms known in the background art for selecting large prime numbers and generators may be used. It is recommended that p be chosen with $\frac{p-1}{2}$ also prime, or at least that $\phi(p)$ not be smooth. (When $\frac{p-1}{2}$ is not prime, information about x_1 and x_2 modulo small factors of $\phi(p)$ may be leaked, which is why it is preferable that $\phi(p)$ not be smooth. Note that ϕ denotes Euler's totient function.) Once p and g have been chosen, the device generates two random exponents x_1 and x_2. The lowest-order bit of x_1 and of x_2 is not considered secret, and may be set to 1. Using p, g, x_1, and x_2, the device can then compute its public key as $g^{x_1 x_2} \bmod p$ and submit it, along with any required identifying information or parameters needed (e.g., p and g), to the CA for certification.

Figure 2 illustrates the process followed by the token to perform private key operations. At step 205, the token obtains the input message y, its own (non-secret) prime p, and its own secret key halves (x_1 and x_2). If x_1, x_2, and p are stored in encrypted and/or

authenticated form, they would be decrypted or verified at this point. At this step, the token should verify that $1 < y < p-1$. At step 210, the token uses a random number generator (or pseudorandom number generator) to select a random integer b_0, where $0 < b_0 < p$. At step 215, the token computes $b_1 = b_0^{-1} \bmod p$. The inverse computation mod p may be performed using the extended Euclidean algorithm or the formula $b_1 = b_0^{\phi(p)-1} \bmod p$. At step 220, the token computes $b_2 = b_1^{x_1} \bmod p$. At this point, b_1 is no longer needed; its storage space may be used to store b_2. Efficient algorithms for computing modular exponentiation, widely known in the art, may be used to complete step 220. Alternatively, when a fast modular exponentiator is available, the computation b_2 may be performed using the relationship $b_2 = b_0^{\phi(p)-x_1} \bmod p$. At step 225, the token computes $b_3 = b_2^{x_2} \bmod p$. At this point, b_2 is no longer needed; its storage space may be used to store b_3. At step 230, the token computes $z_0 = b_0 y \bmod p$. At this point, y and b_0 are no longer needed; their space may be used to store r_1 (computed at step 235) and z_0. At step 235, the token uses a random number generator to select a random integer r_1, where $0 < r_1 < \phi(p)$ and $\gcd(r_1, \phi(p)) = 1$. (If $\frac{p-1}{2}$ is known to be prime, it is sufficient to verify that r_1 is odd.) At step 240, the token updates x_1 by computing $x_1 \leftarrow x_1 r_1 \bmod \phi(p)$. The old value of x_1 is deleted and replaced with the updated value. At step 245, the token computes $r_2 = (r_1^{-1}) \bmod \phi(p)$. If $\frac{p-1}{2}$ is prime, then r_2 can be found using a modular exponentiator and the Chinese Remainder Theorem. Note that r_1 is not needed after this step, so its space may be used to store r_2. At step 250, the token updates x_2 by computing $x_2 \leftarrow x_2 r_2 \bmod \phi(p)$. The old value of x_2 should be deleted and replaced with the updated value. At step 255, the token computes $z_1 = (z_0)^{x_1} \bmod p$. Note that z_0 is not needed after this step, so its space may be used to store z_1. At step 260, the token computes $z_2 = (z_1)^{x_2} \bmod p$. Note that z_1 is not needed after this step, so its space may be used to store z_2. At step 265, the token finds the exponential key exchange result by computing $z = z_2 b_3 \bmod p$. Finally, at step 270, the token erases and frees any remaining temporary variables.

The process shown in Figure 2 correctly computes $z = y^x \bmod p$, where $x = x_1 x_2 \bmod \phi(p)$, since:

$$
\begin{aligned}
z &= z_2 b_3 \bmod p \\
&= \left(z_1^{x_2} \bmod p\right)\left(b_2^{x_2} \bmod p\right) \bmod p \\
&= \left(\left(z_0^{x_1} \bmod p\right)^{x_2}\right)\left(\left(b_1^{x_1} \bmod p\right)^{x_2}\right) \bmod p \\
&= \left(b_0 y \bmod p\right)^{x_1 x_2}\left(b_0^{-1} \bmod p\right)^{x_1 x_2} \bmod p \\
&= y^{x_1 x_2} \bmod p \\
&= y^{x} \bmod p.
\end{aligned}
$$

The invention is useful for private key owners communicating with other users (or devices) who have certificates, and also when communicating with users who do not.

If Alice has a certificate and wishes to communicate with Bob who does not have a certificate, the protocol proceeds as follows. Alice sends her certificate (C_{Alice}) to Bob, who receives it and verifies that it is acceptable. Bob extracts y_{Alice} (along with p_{Alice} and g_{Alice}, unless they are system-wide parameters) from C_{Alice}. Next, Bob generates a random exponent x_{BA}, where $0 < x_{\text{AB}} < \phi(p_{\text{Alice}})$. Bob then uses his exponent x_{AB} and Alice's parameters to calculate $y_{\text{BA}} = \left(g_{\text{Alice}}^{x_{\text{BA}}}\right) \bmod p_{\text{Alice}}$ and the session key $z = \left(y_{\text{Alice}}^{x_{\text{BA}}}\right) \bmod p_{\text{Alice}}$. Bob sends y_{BA} to Alice, who performs the operation illustrated in Figure 2 to update her internal parameters and derive z from y_{BA}. Alice then proves that she computed z correctly, for example by sending Bob $H(z \parallel C_{\text{Alice}})$. (Alice cannot authenticate Bob because he does not have a certificate. Consequently, she does not necessarily need to verify that he computed z successfully.) Finally, Alice and Bob can use z (or, more commonly, a key derived from z) to secure their communications.

If both Alice and Bob have certificates, the protocol works as follows. First, Alice and Bob exchange certificates (C_{Alice} and C_{Bob}), and each verifies that other's certificate is valid. Alice then extracts the parameters p_{Bob}, g_{Bob}, and y_{Bob} from C_{Bob}, and Bob extracts p_{Alice}, g_{Alice}, and y_{Alice} from C_{Alice}. Alice then generates a random exponent x_{AB} where $0 < x_{\text{AB}} < \phi(p_{\text{Bob}})$, computes $y_{\text{AB}} = \left(g_{\text{Bob}}\right)^{x_{\text{AB}}} \bmod p_{\text{Bob}}$, and computes $z_{\text{AB}} - \left(y_{\text{Bob}}\right)^{x_{\text{AB}}} \bmod p_{\text{Bob}}$. Bob generates a random x_{BA} where $0 < x_{\text{BA}} < \phi(p_{\text{Alice}})$, computes $y_{\text{BA}} = \left(g_{\text{Alice}}\right)^{x_{\text{BA}}} \bmod p_{\text{Alice}}$, and computes $z_{\text{BA}} - \left(y_{\text{Alice}}\right)^{x_{\text{BA}}} \bmod p_{\text{Alice}}$. Bob sends y_{BA} to Alice, and Alice sends y_{AB} to Bob. Alice and Bob each perform the operation shown in Figure 2, where each uses the prime p from their own certificate and their own secret exponent halves (x_1 and x_2). For the message y in Figure 2, Alice uses y_{BA} (received from Bob), and Bob uses y_{AB} (received from Alice). Using the process shown in Figure 2, Alice computes z. Using z and z_{AB} (computed

previously), she can find a session key K. This may be done, for example, by using a hash function H to compute $K = H(z \| z_{AB})$. The value of z Bob obtains using the process shown in Figure 2 should equal Alice's z_{AB}, and Bob's z_{BA} (computed previously) should equal Alice's z. If there were no errors or attacks, Bob should thus be able to find K, e.g., by computing $K = H(z_{BA} \| z)$. Alice and Bob now share K. Alice can prove her identity by showing that she computed K correctly, for example by sending Bob $H(K \| C_{Alice})$. Bob can prove his identity by sending Alice $H(K \| C_{Bob})$. Alice and Bob can then secure their communications by encrypting and authenticating using K or a key derived from K.

Note that this protocol, like the others, is provided as an example only; many variations and enhancements of the present invention are possible and will be evident to one skilled in the art. For example, certificates may come from a directory, more than two parties can participate in the key agreement, key escrow functionality may be added, the prime modulus p may be replaced with a composite number, etc. Note also that Alice and Bob as they are called in the protocol are not necessarily people; they would normally be computers, cryptographic devices, etc.

For leak resistance to be effective, attackers should not be able to gain new useful information about the secret variables with each additional operation unless a comparable amount of old useful information is made useless. While the symmetric design is based on the assumption that leaked information will not survive the hash operation H_K, this design uses multiplication operations mod $\phi(p)$ to update x_1 and x_2. The most common variety of leaked information, statistical information about exponent bits, is not of use to attackers in this design, as the exponent update process ($x_1 \leftarrow x_1 r_1 \bmod \phi(p)$ and $x_2 \leftarrow x_2 r_2 \bmod \phi(p)$) destroys the utility of this information. The only relevant characteristic that survives the update process is that $x_1 x_2 \bmod \phi(p)$ remains constant, so the system designer should be careful to ensure that the leak function does not reveal information allowing the attacker to find new useful information about $x_1 x_2 \bmod \phi(p)$.

There is a modest performance penalty, approximately a factor of four, for the leak-resistant design as described. One way to improve performance is to remove the blinding and unblinding operations, which are often unnecessary. (The blinding operations prevent attackers from correlating input values of y with the numbers processed by the modular exponentiation operation.) Alternatively or additionally, it is possible to update and reuse

values of b_0, b_3, r_1, and r_2 by computing $b_0 \leftarrow (b_0)^v \bmod p$, $b_3 \leftarrow (b_3)^v \bmod p$, $r_1 \leftarrow (r_1)^w \bmod \phi(p)$, and $r_2 \leftarrow (r_2)^w \bmod \phi(p)$, where v and w are fairly short random exponents. Note that the relationship $b_3 \leftarrow b_0^{-x_1 x_2} \bmod p$ remains true when b_0 and b_3 are both raised to the power v (mod p). The relationship $r_2 = (r_1^{-1}) \bmod \phi(p)$ also remains true when r_1 and r_2 are exponentiated (mod $\phi(p)$). Other parameter update operations may also be used, such as exponentiation with fixed exponents (e.g., $v = w = 3$), or multiplication with random values and their inverses, mod p and $\phi(p)$. The time per transaction with this update process is about half that of the unoptimized leak-resistant implementation, but additional storage is required and care should be taken to ensure that b_0, b_3, r_1, and r_2 will not be leaked or otherwise compromised.

It should also be noted that with this particular type of certified Diffie-Hellman, the negotiated key is the same every time any given pair of users communicate. Consequently, though the blinding operation performed using b_0 and b_3 does serve to protect the exponents, the result K can be leaked in the final step or by the system after the process is complete. If storage is available, parties could keep track of the values of y they have received (or their hashes) and reject duplicates. Alternatively, to ensure that a different result is obtained from each negotiation, Alice and Bob can generate and exchange additional exponents, w_{Alice} and w_{Bob}, for example with $0 < w < 2^{128}$ (where $2^{128} << p$). Alice sets $y = (y_{\text{BA}})^{w_{\text{Alice}} w_{\text{Bob}}} \bmod p$ instead of just $y = y_{\text{BA}}$, and Bob sets $y = (y_{\text{AB}})^{w_{\text{Bob}} w_{\text{Alice}}} \bmod p$ instead of $y = y_{\text{AB}}$ before performing the operation shown in Figure 2.

B. Leak-Resistant RSA

Another asymmetric cryptographic protocol is RSA, which is widely used for digital signatures and public key encryption. RSA private key operations rely on secret exponents. If information about these secret exponents leaks from an implementation, its security can be compromised. Consequently, a leak-resistant implementation of RSA would be useful.

To give RSA private key operations resistance to leaks, it is possible to divide the secret exponent into two halves such that information about either half is destroyed with each operation. These are two kinds of RSA private key operations. The first, private key signing, involves signing a message with one's own private key to produce a digital signature verifiable by anyone with one's corresponding public key. RSA signing operations involve computing $S = M^d \bmod n$, where M is the message, S is the signature (verifiable using $M = S^e$

mod n), d is the secret exponent and equals e^{-1} mod $\phi(n)$, and n is the modulus and equals pq, where n and e are public and p and q are secret primes, and ϕ is Euler's phi function. An RSA public key consists of e and n, while an RSA private key consists of d and n (or other representations of them). For RSA to be secure, d, $\phi(n)$, p, and q should all be secret.

The other RSA operation is decryption, which is used to recover messages encrypted using one's public key. RSA decryption is virtually identical to signing, since the decrypted message M is recovered from the ciphertext C by computing $M = C^d$ mod n, where the ciphertext C was produced by computing $C=M^e$ mod n. Although the following discussion uses variable names from the RSA signing operation, the same techniques may be applied similarly to decryption.

An exemplary leak-resistant scheme for RSA implementations may be constructed as illustrated in Figure 3. At step 300, prior to the commencement of any signing or decryption operations, the device is initialized with (or creates) the public and private keys. The device contains the public modulus n and the secret key components d_1, d_2, and z, and k, where k is a prime number of medium-size (e.g., $0 < k < 2^{128}$) chosen at random, $z = k\phi(n)$, d_1 is a random number such that $0 < d_1 < z$ and $\gcd(d_1, z) = 1$, and $d_2 = (e^{-1} \bmod \phi(n))(d_1^{-1} \bmod z) \bmod z$. In this invention, d_1 and d_2 replace the usual RSA secret exponent d. Techniques for generating the initial RSA primes (e.g., p and q) and modulus (n) are well known in the background art. At step 305, the device computes a random prime k' of medium size (e.g., $0 < k' < 2^{128}$). (Algorithms for efficiently generating prime numbers are known in the art.)

At step 303, the device (token) receives a message M to sign (or to decrypt). At step 310, the device updates z by computing $z \leftarrow k'z$. At step 315, the device updates z again by computing $z \leftarrow z / k$. (There should be no remainder from this operation, since k divides z.) At step 320, k is replaced with k' by performing $k \leftarrow k'$. Because k' will not be used in subsequent operations, its storage space may be used to hold R (produced at step 325). At step 325, the device selects a random R where $0 < R < z$ and $\gcd(R, z) = 1$. At step 330, the device updates d_1 by computing $d_1 \leftarrow d_1R \bmod z$. At step 335, the device finds the inverse of R by computing $R' \leftarrow R^{-1} \bmod z$ using, for example, the extended Euclidean algorithm. Note that R is no longer needed after this step, so its storage space may be erased and used to hold R'. At step 340, the device updates d_2 by computing $d_2 \leftarrow d_2R' \bmod z$. At step 345, the device computes $S_0 = M^{d_1} \bmod n$, where M is the input message to be signed (or the message

to be decrypted). Note that M is no longer needed after this step, so its storage space may be used for S_0. At step 350, the device computes $S = S_0^{d_2} \bmod n$, yielding the final signature (or plaintext if decrypting a message). Leak-resistant RSA has similar security characteristics as normal RSA; standard message padding, post-processing, and key sizes may be used. Public key operations are also performed normally (e.g., $M = S^e \bmod n$).

A simpler RSA leak resistance scheme may be implemented by splitting the exponent d into two halves d_1 and d_2 such that $d_1 + d_2 = d$. This can be achieved during key generation by choosing d_1 to be a random integer where $0 \le d_1 \le d$, and choosing $d_2 \leftarrow d - d_1$. To perform private key operations, the device needs d_1 and d_2, but it does not need to contain d. Prior to each private key operation, the cryptographic device identifies which of d_1 and d_2 is larger. If $d_1 > d_2$, then the device computes a random integer r where $0 \le r \le d_1$, adds r to d_2 (i.e., $d_2 \leftarrow d_2 + r$), and subtracts r from d_1 (i.e., $d_1 \leftarrow d_1 - r$). Otherwise, if $d_1 \le d_2$, then the device chooses a random integer r where $0 \le r \le d_2$, adds r to d_1 (i.e., $d_1 \leftarrow d_1 + r$), and subtracts r from d_2 (i.e., $d_2 \leftarrow d_2 - r$). Then, to perform the private key operation on a message M, the device computes $s_1 = M^{d_1} \bmod n$, $s_2 = M^{d_2} \bmod n$, and computes the signature $S = s_1 s_2 \bmod n$. While this approach of splitting the exponent into two halves whose sum equals the exponent can also be used with Diffie-Hellman and other cryptosystems, dividing the exponent into the product of two numbers mod ϕ(p) is usually preferable since the assumption that information about d_1+d_2 will not leak is less conservative than the assumption that information about $x_1 x_2 \bmod \phi(p)$ will not leak. In the case of RSA, updates mod $\phi(n)$ cannot be done safely, since $\phi(n)$ must be kept secret.

When the Chinese Remainder Theorem is required for performance, it is possible to use similar techniques to add leak resistance by maintaining multiples of the secret primes (p and q) that are updated every time (e.g., multiplying by the new multiple then dividing by the old multiple). These techniques also protect the exponents (d_p and d_q) as multiples of their normal values. At the end of the operation, the result S is corrected to compensate for the adjustments to d_p, d_q, p, and q.

An exemplary embodiment maintains state information consisting of the values n, B_i, B_p, k, p_k, q_k, d_{pk}, d_{qk}, p_{Inv}, and f. To convert a traditional RSA CRT private key (consisting of p, q, d_p, and d_q with $p < q$) into the new representation, a random value for k is chosen, where $0 < k < 2^{64}$. The value B_i is chosen at random where $0 < B_i < n$, and R_1 and R_2 are chosen at

random where $0 < R_1 < 2^{64}$ and $0 < R_2 < 2^{64}$. (Of course, constants such as 2^{64} are chosen as example values. It is possible, but not necessary, to place constraints on random numbers, such as requiring that they be prime.) The leak-resistant private key state is then initialized by setting $n \leftarrow pq$, $B_f \leftarrow B_i^{-d} \bmod n$, $p_k \leftarrow (k)(p)$, $q_k \leftarrow (k)(q)$, $d_{pk} \leftarrow d_p + (R_1)(p) - R_1$, $d_{qk} \leftarrow d_q + (R_2)(q) - R_2$, $p_{Inv} \leftarrow k(p^{-1} \bmod q)$, and $f \leftarrow 0$.

To update the system state, first a random value α may be produced where $0 < \alpha < 2^{64}$. Then compute $p_k \leftarrow ((\alpha)(p_k)) / k$, $q_k \leftarrow ((\alpha)(q_k)) / k$, $p_{Inv} \leftarrow ((\alpha)(p_{Inv})) / k$, $k \leftarrow \alpha$. The exponents d_{pk} and d_{qk} may be updated by computing $d_{pk} \leftarrow d_{pk} \pm (R_3 p_k - R_3 k)$ and $d_{qk} \leftarrow d_{qk} \pm (R_4 q_k - R_4 k)$, where R_3 and R_4 can be random or constant values (even 1). The blinding factors B_i and B_f may be updated by computing $B_i = B_i^2 \bmod n$ and $B_f = B_f^2 \bmod n$, by computing new blinding factors, by exponentiating with a value other than 2, etc. Update processes should be performed as often as practical, for example before or after each modular exponentiation process. Before the update begins, a failure counter f is incremented, and when the update completes f is set to zero. If f ever exceeds a threshold value indicating too many consecutive failures, the device should temporarily or permanently disable itself. Note that if the update process is interrupted, memory values should not be left in intermediate states. This can be done by using complete reliable memory updates. If the total set of variable changes is too large for a single complete update, it is possible to store α first then do each variable update reliably which keeping track of how many have been completed.

To perform a private key operation (such as decryption or signing), the input message C is received by the modular exponentiator. Next, the value is blinded by computing $C' \leftarrow (C)(B_i) \bmod n$. The blinded input message is then used to compute modified CRT intermediates by computing $m_{pk} \leftarrow (C')^{d_{pk}} \bmod p_k$ and $m_{qk} \leftarrow (C')^{d_{qk}} \bmod q_k$. Next in the exemplary embodiment, the CRT intermediates are multiplied by k, e.g. $m_{pk} \leftarrow (k)(m_{pk}) \bmod p_k$ and $m_{qk} \leftarrow (k)(m_{qk}) \bmod q_k$. The CRT difference is then computed as $m_{pqk} = (m_{pk} [+ q_k] - m_{qk}) [\bmod q_k]$, where the addition of q_k and/or reduction mod q_k are optional. (The addition of q_k ensures that the result is non-negative.) The blinded result can be computed as

$$M' = \frac{(m_{pk})k + p_k\left[\left(\frac{(p_{Inv})(m_{pqk})}{k}\right) \bmod q_k\right]}{k^2}$$

, then the final result M is computed as $M = (M')B_f \bmod n$.

As one of ordinary skill in the art will appreciate, variant forms of the invention are possible. For example, the computational processes can be re-ordered or modified without significantly changing the invention. Some portions (such as the initial and blinding steps) can be skipped. In another example, it is also possible to use multiple blinding factors (for example, instead of or in addition to the value k).

In some cases, other techniques may also be appropriate. For example, exponent vector codings may be rechosen frequently using, for example, a random number generator. Also, Montgomery arithmetic may be performed mod j where j is a value that is changed with each operation (as opposed to traditional Montgomery implementations where j is constant with $j = 2^k$). The foregoing shows that the method and apparatus of the present invention can be implemented using numerous variations and modifications to the exemplary embodiments described herein, as would be known by one skilled in the art.

C. Leak-Resistant ElGamal Public Key Encryption and Digital Signatures

Still other asymmetric cryptographic protocols that may be improved using the techniques of the invention. For example, ElGamal and related cryptosystems are widely used for digital signatures and public key encryption. If information about the secret exponents and parameters leaks from an ElGamal implementation, security can be compromised. Consequently, leak-resistant implementations of ElGamal would be useful.

The private key in the ElGamal public key encryption scheme is a randomly selected secret a where $1 \le a \le p-2$. The non-secret parameters are a prime p, a generator α, and $\alpha^a \bmod p$. To encrypt a message m, one selects a random k (where $1 \le k \le p-2$) and computes the ciphertext (γ, δ) where $\gamma = \alpha^k \bmod p$ and $\delta = m(\alpha^a \bmod p)^k \bmod p$. Decryption is performed by computing $m = \delta(\gamma^{p-1-a}) \bmod p$. (See the Handbook of Applied Cryptography by A. Menezes, P. van Oorschot, and S. Vanstone, 1997, pages 294-298, for a description of ElGamal public-key encryption).

To make the ElGamal public-key decryption process leak-resistant, the secret exponent $(p - 1 - a)$ is stored in two halves a_1 and a_2, such that $a_1 a_2 = (\phi(p)-a) \bmod \phi(p)$. When generating ElGamal parameters for this leak-resistant implementation, it is recommended, but not required, that p be chosen with $\frac{p-1}{2}$ prime so that $\phi(p)/2$ is prime. The variables a_1 and a_2 are normally chosen initially as random integers between 0 and $\phi(p)$.

Alternatively, it is possible to generate a first, then choose a_1 and a_2, as by selecting a_1 relatively prime to $\phi(p)$ and computing $a_2 = (a^{-1} \bmod \phi(p))(a_1^{-1} \bmod \phi(p)) \bmod \phi(p)$.

Figure 4 illustrates an exemplary leak-resistant ElGamal decryption process. At step 405, the decryption device receives an encrypted message pair (γ, δ). At step 410, the device selects a random r_1 where $1 \le r_1 < \phi(p)$ and $\gcd(r_1, \phi(p)) = 1$. At step 415, the device updates a_1 by computing $a_1 \leftarrow a_1 r_1 \bmod \phi(p)$, over-writing the old value of a_1 with the new value. At step 420, the device computes the inverse of r_1 by computing $r_2 = (r_1)^{-1} \bmod \phi(p)$. Because r_1 is not used after this step, its storage space may be used to hold r_2. Note that if $\frac{p-1}{2}$ is prime, then r_2 may also be found by finding $r_2' = r_1^{(p-1)/2-2} \bmod \frac{p-1}{2}$, and using the CRT to find $r_2 \pmod{p-1}$. At step 425, the device updates a_2 by computing $a_2 \leftarrow a_2 r_2 \bmod \phi(p)$. At step 430, the device begins the private key (decryption) process by computing $m' = \gamma^{a_1} \bmod p$. At step 435, the device computes $m = \delta (m')^{a_2} \bmod p$ and returns the message m. If verification is successful, the result equals the original message because:

$$\begin{aligned}(\delta)(m')^{a_2} \bmod p &= \left(m(\alpha^a)^k\right)\left(\gamma^{a_1} \bmod p\right)^{a_2} \bmod p \\ &= \left(m\alpha^{ak}\right)\left(\gamma^{a_1 a_2 \bmod \phi(p)}\right) \bmod p \\ &= \left(m\alpha^{ak}\right)\left((\alpha^k \bmod p)^{-a \bmod \phi(p)}\right) \bmod p \\ &= \left(m\alpha^{ak}\right)\left(\alpha^{-ak}\right) \bmod p \\ &= m\end{aligned}$$

As with the ElGamal public key encryption scheme, the private key for the ElGamal digital signature scheme is a randomly-selected secret a, where $1 \le a \le p-2$. The public key is also similar, consisting of a prime p, a generator α, and public parameter y where $y = \alpha^a \bmod p$. To sign a message m, the private key holder chooses or precomputes a random secret integer k (where $1 \le k \le p-2$ and k is relatively prime to $p-1$) and its inverse, $k^{-1} \bmod \phi(p)$. Next, the signer computes the signature (r, s), where $r = \alpha^k \bmod p$, $s = \left((k^{-1} \bmod \phi(p))[\mathrm{H}(m) - ar]\right) \bmod \phi(p)$, and $\mathrm{H}(m)$ is the hash of the message. Signature verification is performed using the public key (p, α, y) by verifying that $1 \le r < p$ and by verifying that $y^r r^s \bmod p = \alpha^{\mathrm{H}(m)} \bmod p$.

To make the ElGamal digital signing process leak-resistant, the token containing the private key maintains three persistent variables, a_k, w, and r. Initially, $a_k = a$ (the private exponent), $w = 1$, and $r = \alpha$. When a message m is to be signed (or during the

Petitioner Apple Inc. - Exhibit 1004, p. 5037

precomputation before signing), the token generates a random number b and its inverse b^{-1} mod $\phi(p)$, where b is relatively prime to $\phi(p)$ and $0 < b < \phi(p)$. The token then updates a_k, w, and r by computing $a_k \leftarrow (a_k)(b^{-1}) \bmod \phi(p)$, $w \leftarrow (w)(b^{-1}) \bmod \phi(p)$, and $r \leftarrow (r^b) \bmod p$. The signature (r, s) is formed from the updated value of r and s, where $s = (w(\mathrm{H}(m) - a_k r)) \bmod \phi(p)$. Note that a_k, w, and r are not randomized prior to the first operation, but should be randomized before exposure to possible attack, since otherwise the first operation may leak more information than subsequent ones. It is thus recommended that a dummy signature or parameter update with $a_k \leftarrow (a_k)(b^{-1}) \bmod \phi(p)$, $w \leftarrow (w)(b^{-1}) \bmod \phi(p)$, and $r \leftarrow (r^b) \bmod p$ be performed immediately after key generation. Valid signatures produced using the exemplary tamper-resistant ElGamal process may be checked using the normal ElGamal signature verification procedure.

It is also possible to split all or some the ElGamal variables into two halves as part of the leak resistance scheme. In such a variant, a is replaced with a_1 and a_2, w with w_1 and w_2, and r with r_1 and r_2. It is also possible to reorder the operations by performing, for example, the parameter updates as a precomputation step prior to receipt of the enciphered message. Other variations and modifications to the exemplary embodiments described herein will be evident to one skilled in the art.

D. Leak-Resistant DSA

Another commonly used asymmetric cryptographic protocol is the Digital Signature Algorithm (DSA, also known as the Digital Signature Standard, or DSS), which is defined in "Digital Signature Standard (DSS)," Federal Information Processing Standards Publication 186, National Institute of Standards and Technology, May 19, 1994 and described in detail in the Handbook of Applied Cryptography, pages 452 to 454. DSA is widely used for digital signatures. If information about the secret key leaks from a DSA implementation, security can be compromised. Consequently, leak-resistant implementations of DSA would be useful.

In non-leak-proof systems, the private key consists of a secret parameter a, and the public key consists of (p, q, α, y), where p is a large (usually 512 to 1024 bit) prime, q is a 160-bit prime, α is a generator of the cyclic group of order q mod p, and $y = \alpha^a \bmod p$. To sign a message whose hash is $\mathrm{H}(m)$, the signer first generates (or precomputes) a random integer k and its inverse k^{-1} mod q, where $0 < k < q$. The signer then computes the signature (r, s), where $r = (\alpha^k \bmod p) \bmod q$, and $s = (k^{-1} \bmod q)(\mathrm{H}(m) + ar) \bmod q$.

In an exemplary embodiment of a leak-resistant DSA signing process, the token containing the private key maintains two variables in nonvolatile memory, a_k and k, which are initialized with $a_k = a$ and $k = 1$. When a message m is to be signed (or during the precomputation before signing), the token generates a random integer b and its inverse b^{-1} mod q, where $0 < b < q$. The token then updates a_k and k by computing $a_k \leftarrow (a_k b^{-1} \bmod q)(k) \bmod q$, followed by $k \leftarrow b$. The signature (r, s) is formed from the updated values of a_k and k by computing $r = \alpha^k \bmod p$ (which may be reduced mod q), and $s = [(b^{-1}H(m) \bmod q) + (a_k r) \bmod q] \bmod q$. As indicated, when computing s, $b^{-1}H(m) \bmod q$ and $(a_k r) \bmod q$ are computed first, then combined mod q. Note that a_k and k should be randomized prior to the first operation, since the first update may leak more information than subsequent updates. It is thus recommended that a dummy signature (or parameter update) be performed immediately after key generation. Valid signatures produced using the leak-resistant DSA process may be checked using the normal DSA signature verification procedure.

IV. Other Algorithms and Applications

Still other cryptographic processes can be made leak-proof or leak-resistant, or may be incorporated into leak-resistant cryptosystems. For example, cryptosystems such as those based on elliptic curves (including elliptic curve analogs of other cryptosystems), secret sharing schemes, anonymous electronic cash protocols, threshold signatures schemes, etc. be made leak resistant using the techniques of the present invention.

Implementation details of the schemes described may be adjusted without materially changing the invention, for example by re-ordering operations, inserting steps, substituting equivalent or similar operations, etc. Also, while new keys are normally generated when a new system is produced, it is often possible to add leak resistance retroactively while maintaining or converting existing private keys.

Leak-resistant designs avoid performing repeated mathematical operations using non-changing (static) secret values, since they are likely to leak out. However, in environments where it is possible to implement a simple function (such as an exclusive OR) that does not leak information, it is possible use this function to implement more complex cryptographic operations.

While the exemplary implementations assume that the leak functions can reveal any information present in the system, designers may often safely use the (weaker) assumption

that information not used in a given operation will not be leaked by that operation. Schemes using this weaker assumption may contain a large table of precomputed subkey values, from which a unique or random subset are selected and/or updated for each operation. For example, DES implementations may use indexed permutation lookup tables in which a few table elements are exchanged with each operation.

While leak resistance provides many advantages, the use of leak resistance by itself cannot guarantee good security. For example, leak-resistant cryptosystems are not inherently secure against error attacks, so operations should be verified. (Changes can even be made to the cryptosystem and/or leak resistance operations to detect errors.) Similarly, leak resistance by itself does not prevent attacks that extract the entire state out of a device (e.g., $L=L_{MAX}$). For example, traditional tamper resistance techniques may be required to prevent attackers from staining ROM or EEPROM memory cells and reading the contents under a microscope. Implementers should also be aware of interruption attacks, such as those that involve disconnecting the power or resetting a device during an operation, to ensure that secrets will not be compromised or that a single leaky operation will not be performed repeatedly. (As a countermeasure, devices can increment a counter in nonvolatile memory prior to each operation, and reset or reduce the counter value when the operation completes successfully. If the number of interrupted operations since the last successful update exceeds a threshold value, the device can disable itself.) Other tamper resistance mechanisms and techniques, such as the use of fixed-time and fixed-execution path code or implementations for critical operations, may need to be used in conjunction with leak resistance, particularly for systems with a relatively low self-healing rate (e.g., L_{MAX} is small).

Leak-resistant algorithms, protocols, and devices may be used in virtually any application requiring cryptographic security and secure key management, including without limitation: smartcards, electronic cash, electronic payments, funds transfer, remote access, timestamping, certification, certificate validation, secure e-mail, secure facsimile, telecommunications security (voice and data), computer networks, radio and satellite communications, infrared communications, access control, door locks, wireless keys, biometric devices, automobile ignition locks, copy protection devices, payment systems, systems for controlling the use and payment of copyrighted information, and point of sale terminals.

The foregoing shows that the method and apparatus of the present invention can be implemented using numerous variations and modifications to the exemplary embodiments described herein, as would be known by one skilled in the art. Thus, it is intended that the scope of the present invention be limited only with regard to the claims below.

WHAT IS CLAIMED IS:

1. A method for implementing RSA with the Chinese Remainder Theorem for use in a cryptographic system, with resistance to leakage attacks against said cryptographic system, comprising the steps of:

(a) obtaining a representation of an RSA private key corresponding to an RSA public key, said private key characterized by secret factors p and q;

(b) storing said representation of said private key in a memory;

(c) obtaining a message for use in an RSA cryptographic operation;

(d) computing a first modulus, corresponding to a multiple of p, where the value of said multiple of p and the value of said multiple of p divided by p are both unknown to an attacker of said cryptographic system;

(e) reducing said message modulo said first modulus;

(f) performing modular exponentiation on the result of step (e);

(g) computing a second modulus, corresponding to a multiple of q, where the value of said multiple of q and the value of said multiple of q divided by q are both unknown to an attacker of said cryptographic system;

(h) reducing said message modulo said second modulus;

(i) performing modular exponentiation on the result of step (h);

(j) combining the results of said steps (e) and (h) to produce a result which, if operated on with an RSA public key operation using said RSA public key, yields said message; and

(k) repeating steps (c) through (j) a plurality of times using different values for said multiple of p and for said multiple of q.

2. The method of claim 1 where:

(i) said step (b) includes storing an exponent d_p of said RSA private key in said memory as a plurality of parameters;

(ii) an arithmetic function of at least one of said plurality of parameters is congruent to d_p, modulo (p-1);

(iii) none of said parameters comprising said stored d_p is equal to d_p;

(iv) an exponent used in said step (f) is at least one of said parameters;

(v) at least one of said parameters in said memory changes with said repetitions of said steps (c) through (j).

3. The method of claim 2 where said plurality of parameters includes a first parameter equal to said d_p plus a multiple of phi(p), and also includes a second parameter equal to a multiple of phi(p), where phi denotes Euler's totient function.

4. The method of claim 1 where the value of said multiple of p divided by p is equal to the value of said multiple of q divided by q.

5. The method of claim 1 where said multiple of p and said multiple of q used in said steps (c) through (j) are updated and modified in said memory after said step (b).

6. The method of claim 1 performed in a smart card.

7. The method of claim 1 where at least two of said steps are performed in an order other than (a) through (k)

8. A method for implementing RSA for use in a cryptographic system, with resistance to leakage attacks against said cryptographic system, comprising the steps of:

(a) obtaining an RSA private key corresponding to an RSA public key, said RSA public key having an RSA modulus n;

(b) storing said private key in a memory in a form whereby a secret parameter of said key is stored as an arithmetic combination of phi(x) and a first at least one key masking parameter, where

(i) an operand x in said phi(x) is an exact multiple of at least one factor of said modulus n of said RSA public key; and

(ii) said first key masking parameter is unknown to an attacker of said cryptosystem;

(iii) a representation of said first key masking parameter is stored in said memory;

(iv) phi denotes Euler's totient function;

(c) receiving a message;

(d) deriving an RSA input from said message;

(e) performing modular exponentiation to raise said RSA input to a power dependent on said secret parameter, modulo an RSA modulus stored in said memory, to produce an RSA result such that said RSA result raised to the

power of the public exponent of said RSA public key, modulo the modulus of said RSA public key, equals said RSA input;

(f) updating said secret parameter in said memory by:

(i) modifying said first key masking parameter to produce a new key masking parameter, where said modification is performed in a manner such that an attacker with partial useful information about said first key masking parameter has less useful information about said new key masking parameter; and

(ii) using said new key masking parameter to update said secret parameter in said memory;

(g) repeating steps (d) through (f) a plurality of times, where the power used for each of said modular exponentiation steps (e) is different.

9. The method of claim 8 where said operand x in said phi(x) corresponds to said RSA modulus n of said RSA public key.

10. The method of claim 8 where said operand x in said phi(x) corresponds to a prime factor of said RSA modulus n of said RSA public key, and where said modular exponentiation of said step (e) is performed using the Chinese Remainder Theorem.

11. A method for implementing exponential key exchange for use in a cryptographic system, with resistance to leakage attacks against said cryptographic system, comprising the steps of:

(a) obtaining, and storing in a memory, exponential key exchange parameters g and p, and a plurality of secret exponent parameters on which an arithmetic relationship may be computed to produce an exponent x;

(b) using a key update transformation to produce a plurality of updated secret exponent parameters while maintaining said arithmetic relationship thereamong;

(c) receiving a public value y from a party with whom said key exchange is desired;

(d) using said updated secret exponent parameters to perform a cryptographic computation yielding an exponential key exchange result $z = y^x \bmod p$;

(e) using said result z to secure an electronic communication with said party; and

(f) performing said steps (b), (c), (d), and (e) in a plurality of transactions.

12. The method of claim 11 where each of said transactions involves a different said party.

13. The method of claim 11 where said arithmetic relationship is such that said exponential key exchange result is a product of certain of said secret exponent parameters, both before and after said step (b).

14. The method of claim 11 where said key update transformation includes choosing a random key update value r; and where said step (b) includes multiplying one of said secret exponent parameters by r and another of said secret exponent parameters by an inverse of r, said multiplication being performed modulo phi(p), where phi is Euler's totient function.

15. The method of claim 11 where said key update transformation includes choosing a random key update value r; and where said step (b) includes adding r to one of said secret exponent parameters and subtracting r from another of said secret exponent parameters.

16. The method of claim 15 where said secret exponent parameters include two values x_1 and x_2 such that x_1+x_2 is congruent to x, modulo phi(p), where phi is Euler's totient function, and where said step of performing said cryptographic computation yielding said exponential key exchange result includes computing $z_1 = y\hat{}x_1 \bmod p$, $z_2 = y\hat{}x_2 \bmod p$, and $z=z_1z_2 \bmod p$.

17. A cryptographic token configured to perform cryptographic operations using a secret key in a secure manner, comprising:

(a) an interface configured to receive power from a source external to said token;

(b) a memory containing said secret key;

(c) a processor:

(i) configured to receive said power delivered via said interface;

(ii) configured to perform said processing using said secret key from said memory;

(d) said token having a power consumption characteristic:

(i) that is externally measurable; and

(ii) that varies over time in a manner measurably correlated with said cryptographic operations; and

(e) a source of unpredictable information usable in said cryptographic operations to make determination of said secret key infeasible from external measurements of said power consumption characteristic.

18. The cryptographic token of claim 17, in the form of a secure microprocessor.

19. The cryptographic token of claim 17, in the form of a smart card.

20. The cryptographic token of claim 19, wherein said cryptographic operations performed by said smart card enable a holder thereof to decrypt an encrypted communication received via a computer network.

21. The cryptographic token of claim 19, wherein said smart card is configured to store value in an electronic cash scheme.

22. The cryptographic token of claim 21, wherein said cryptographic operations include authenticating that a balance of said stored value has been decreased.

23. The cryptographic token of claim 17, wherein said cryptographic operations include asymmetric private key operations.

24. The cryptographic token of claim 23 wherein said cryptographic operations include exponential key agreement operations.

25. The cryptographic token of claim 23, wherein said cryptographic operations include DSA signing operations.

26. The cryptographic token of claim 23, wherein said cryptographic operations include ElGamal private key operations.

27. The cryptographic token of claim 23, wherein said asymmetric private key operations include RSA private key operations.

28. The cryptographic token of claim 27 wherein said private key operations include Chinese Remainder Theorem operations.

29. The cryptographic token of claim 17, wherein said cryptographic operations include symmetric encryption operations.

30. The cryptographic token of claim 17, wherein said cryptographic operations include symmetric decryption operations.

31. The cryptographic token of claim 17, wherein said cryptographic operations include symmetric authentication operations using said secret key.

32. The cryptographic token of claim 17, wherein said cryptographic operations include authenticating a payment.

33. The cryptographic token of claim 17, wherein said cryptographic operations include securing a broadcast communications signal.

34. The cryptographic token of claim 33, wherein said cryptographic operations include decrypting a satellite broadcast.

35. A method for securely managing and using a private key in a computing environment where information about said private key may leak to attackers, comprising the steps of:

(a) using a first private key, complementary to a public key, to perform first asymmetric cryptographic operation;

(b) reading at least a portion of said first private key from a memory;

(c) transforming said read portion of said first private key to produce a second private key:

(i) said second private key usable to perform a subsequent asymmetric cryptographic operation in a manner that remains complementary to said public key, and

(ii) said transformation enabling said asymmetric cryptographic operations to be performed in a manner such that information leaked during said

first asymmetric cryptographic operation does not provide incrementally useful information about said second private key;

(d) obtaining a datum;

(e) using said second private key to perform said subsequent asymmetric cryptographic operation on said datum.

36. The method of claim 35 where said asymmetric cryptographic operation includes a digital signing operation.

37. The method of claim 36 where said signing operation is an RSA operation.

38. The method of claim 36 where said signing operation is an DSA operation.

39. The method of claim 36 where said signing operation is an ElGamal operation.

40. The method of claim 35 where said asymmetric cryptographic operation includes a decryption operation.

41. The method of claim 40 where said decryption operation is an RSA operation.

42. The method of claim 40 where said decryption operation is an ElGamal operation.

43. The method of claim 35 where at least two of said steps are performed in an order different than (a), (b), (c), (d), (e).

44. The method of claim 35 further comprising the step, after at least said step (c), of replacing said private key in said memory with said second private key.

45. The method of claim 35, performed in a smart card.

46. The method of claim 35, further comprising the steps of: prior to at least said step (c), incrementing a counter stored in a nonvolatile memory and verifying that said counter has not exceeded a threshold value; and after at least said step (c) has completed successfully, decreasing a value of said counter.

47. A method for performing cryptographic transactions while protecting a stored cryptographic key against compromise due to leakage attacks, comprising the steps of:

(a) retrieving a stored private cryptographic key stored in a memory, said stored key having been used in a previous cryptographic transaction;

(b) using a first cryptographic function to derive from said stored key an updated key, about which useful information about said stored key obtained through monitoring of leaked information is effectively uncorrelated to said updated key;

(c) replacing said stored key in said memory with said updated key;

(d) using an asymmetric cryptographic function, cryptographically processing a datum with said updated key; and

(e) sending said cryptographically processed datum to an external device having a public key corresponding to said stored key.

48. The method of claim 47 where said stored key includes a first plurality of parameters, and where said updated key includes a second plurality of parameters.

49. The method of claim 48 where no secret value within said first plurality of parameters is included within said second plurality of parameters.

50. The method of claim 49 where said first plurality of parameters is different than said second plurality of parameters, yet a predetermined relationship among said first plurality of parameters is also maintained among said second plurality of parameters.

51. The method of claim 50 where said relationship among said plurality of parameters is an arithmetic function involving at least two of said plurality of parameters.

52. The method of claim 51 where said arithmetic function is the sum of said parameters.

53. The method of claim 51 where said relationship includes a bitwise combination of said parameters.

54. The method of claim 53 where said bitwise combination is an exclusive OR.

55. The method of claim 47 where said step (b) includes using pseudorandomness to derive said updated key.

56. A method for implementing a private key operation for an asymmetric cryptographic system with resistance to leakage attacks against said cryptographic system, comprising the steps of:

(a) encoding a portion of a private key as at least two component parts, such that an arithmetic function of said parts yields said portion;

(b) modifying said component parts to produce updated component parts, but where said arithmetic function of said updated parts still yields said private key portion;

(c) obtaining a message for use in an asymmetric private key cryptographic operation;

(d) separately applying said component parts to said message to produce an intermediate result;

(e) deriving a final result from said intermediate result such that said final result is a valid result of applying said private key to said message; and

(f) repeating steps (b) through (e) a plurality of times.

57. The method of claim 56 where said private key portion includes an exponent, and where said intermediate result represents the result of raising said message to the power of said exponent, modulo a second key portion.

58. The method of claim 57 where said private key operation is configured for use with an RSA cryptosystem.

59. The method of claim 57 where said private key operation is configured for use with an ElGamal cryptosystem.

60. The method of claim 56 where said private key operation is configured for use with a DSA cryptosystem.

61. The method of claim 60 where said private key is represented by secret parameters a_k and k whose product, modulo a predetermined DSA prime q for said private key, yields said private key portion.

62. The method of claim 56 implemented in a smart card.

63. The method of claim 56 where said private key is configured for use with an elliptic curve cryptosystem.

64. A method for performing cryptographic transactions in a cryptographic token while protecting a stored cryptographic key against compromise due to leakage attacks, including the steps of:

(a) retrieving said stored key from a memory;

(b) cryptographically processing said key, to derive an updated key, by executing a cryptographic update function that:

(i) prevents partial information about said stored key from revealing useful information about said updated key, and

(ii) also prevents partial information about said updated key from revealing useful information about said stored key;

(c) replacing said stored key in said memory with said updated key;

(d) performing a cryptographic operation using said updated key; and

(e) repeating steps (a) through (d) a plurality of times.

65. The method of claim 64 where said cryptographic update function of said step (b) includes a one-way hash operation.

66. The method of claim 64 where said cryptographic operation of said step (d) is a symmetric cryptographic operation; and comprising the further step of sending a result of said cryptographic operation to a party capable of rederiving said updated key.

67. The method of claim 64 further comprising the step, prior to said step (a), of receiving from a second party a symmetric authentication code and a parameter; and said where said step (b) includes iterating a cryptographic transformation a number of times determined from said parameter; and where said step (d) includes performing a symmetric message authentication code verification operation.

Petitioner Apple Inc. - Exhibit 1004, p. 5051

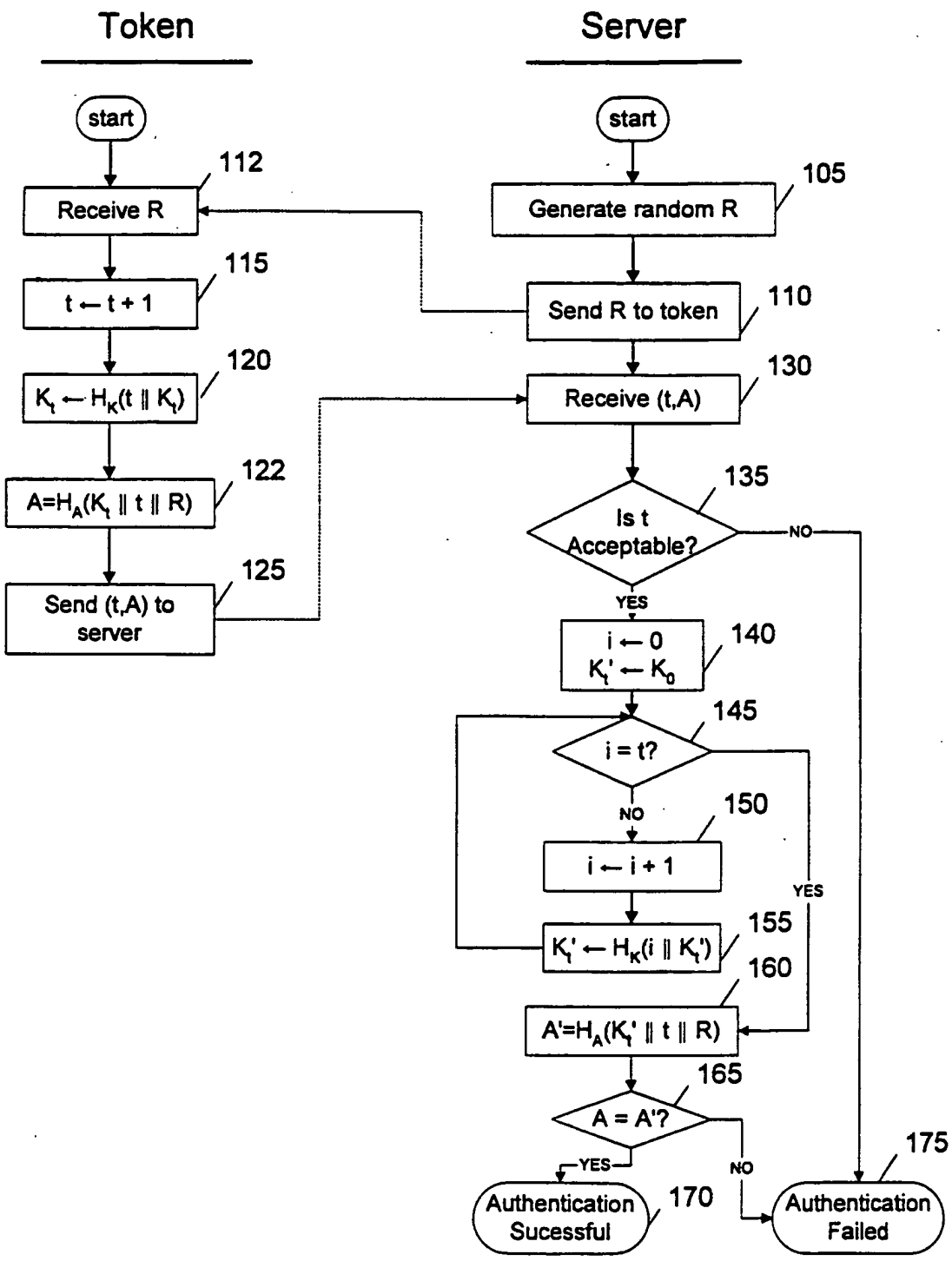
68. he method of claim 66 where said step (d) of performing said cryptographic operation includes using said updated key to encrypt a datum.

69. The method of claim 66 where said updated key contains unpredictable information such that said updated key is not stored in its entirety anywhere outside of said cryptographic token; and where the result of said step (d) is independent of said unpredictable information.

70. The method of claim 64 where said step (c) of replacing said stored key includes:
(i) explicitly erasing a region of said memory containing said stored key; and
(ii) storing said updated key in said region of memory.

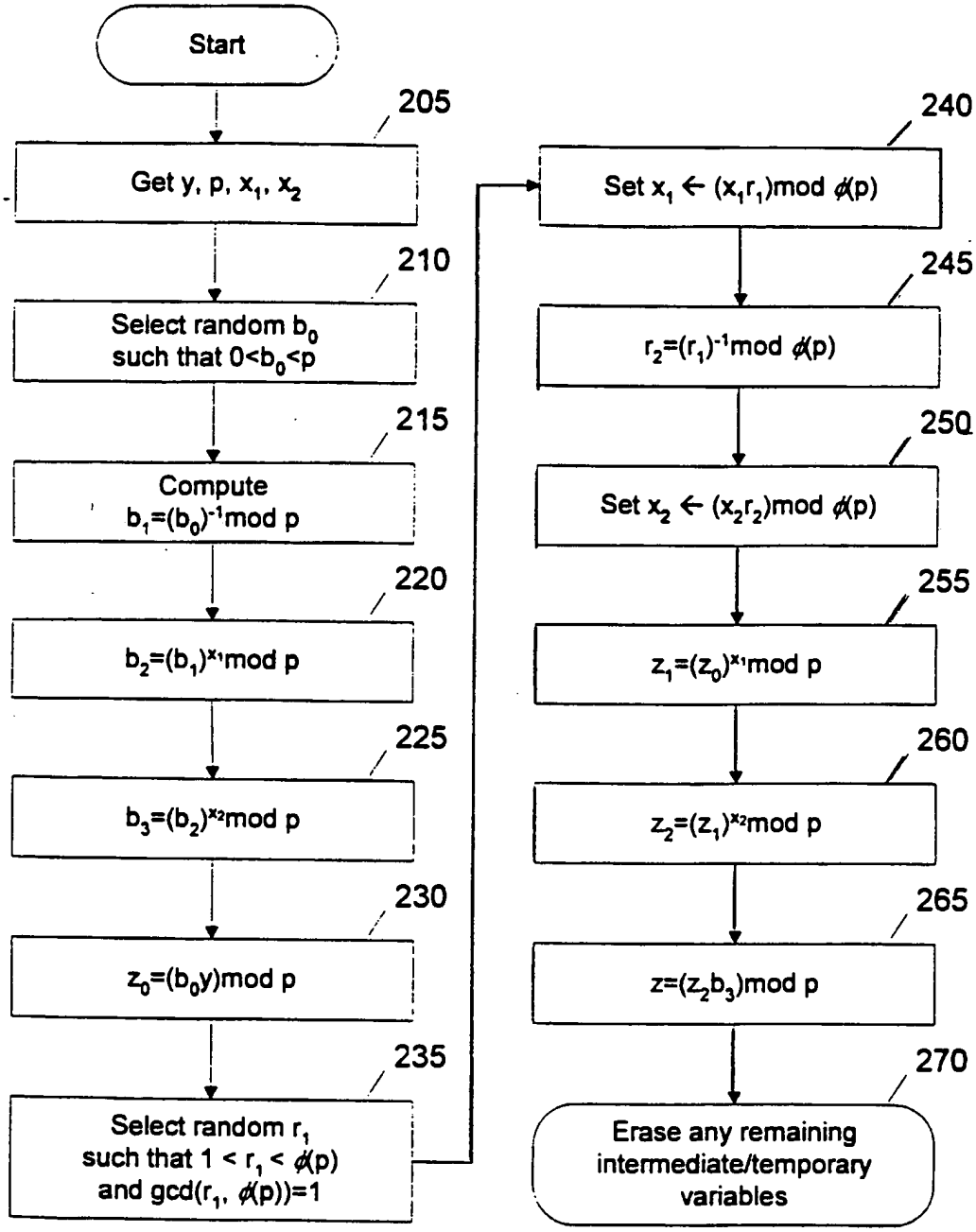
71. The method of claim 64 performed within a smart card.

FIG. 1



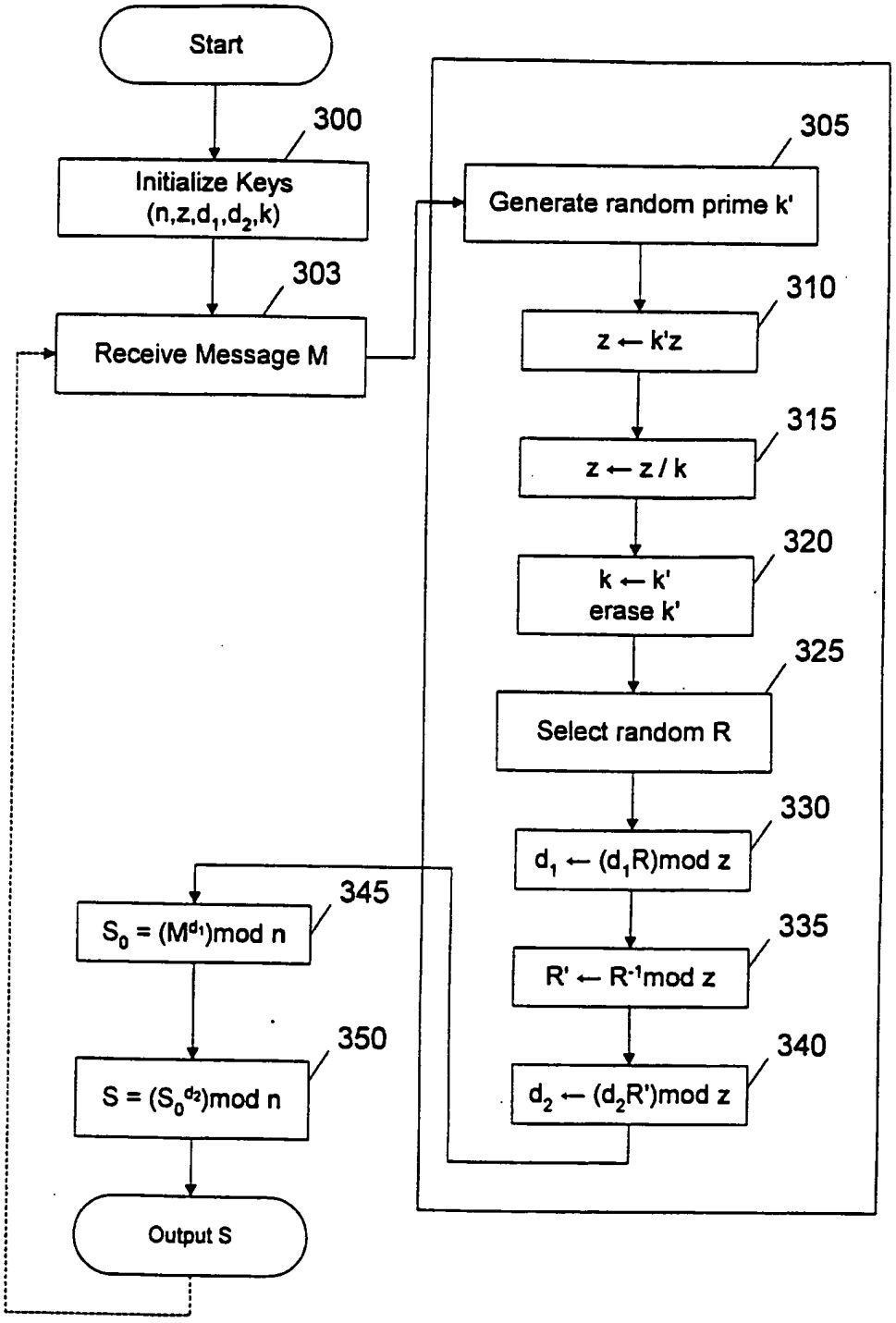
Petitioner Apple Inc. - Exhibit 1004, p. 5053

FIG. 2



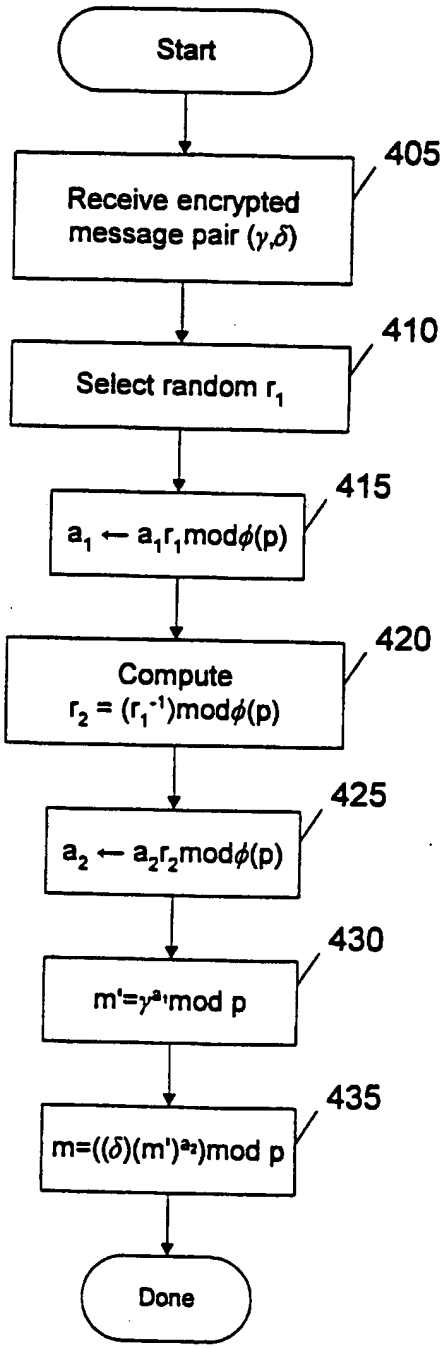
Petitioner Apple Inc. - Exhibit 1004, p. 5054

FIG. 3



Petitioner Apple Inc. - Exhibit 1004, p. 5055

FIG. 4



Petitioner Apple Inc. - Exhibit 1004, p. 5056

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27896

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :HO4 L 9/30
US CL :380/30,49

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 380/30,49

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 4,799,258 A (DAVIES et al) 17 January 1989, abstract, col.4, lines 43-50, col.7,lines 15-33, col.8,lines 12-19	17-23,25- 45
Y	US 5,546,463 A (CAPUTO et al) 13 August 1996, abstract, col.2, lines, 60-65, col.5, lines 39-50,53-58, col.6, lines 7-12	17-23,25-45
A,P	US 5,848,159 A (COLLINS et al.) 08 December 1998, abstract, col.1, lines 56-67, col.4, lines 33-44, col.5, lines 52-67, col.6, lines 24-30	1-16,46-71

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means.		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
30 MARCH 1999	06 MAY 1999
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-0040	Authorized officer GAIL HAYES Telephone No. (703) 305-9711

Form PCT/ISA/210 (second sheet)(July 1992) *

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS
search terms:token,smart card,tamper proof, tamper resistant,leak-resistant,RSA, public key, private key, chinese remainder theorem, diffie hellman,dsa,des

Petitioner Apple Inc. - Exhibit 1004, p. 5058

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
H04N 7/167, G06F 1/00

A1

(11) International Publication Number: **WO 99/48296**

(43) International Publication Date: 23 September 1999 (23.09.99)

(21) International Application Number: PCT/US99/05734

(22) International Filing Date: 16 March 1999 (16.03.99)

(30) Priority Data:
60/078,053 16 March 1998 (16.03.98) US

(71) Applicant: INTERTRUST TECHNOLOGIES CORPORATION [US/US]; 460 Oakmead Parkway, Sunnyvale, CA 94086 (US).

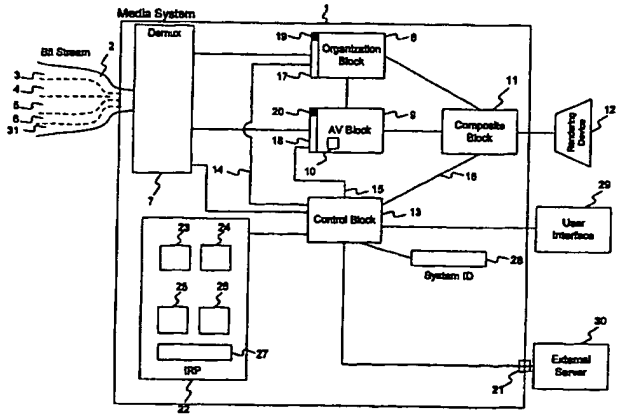
(72) Inventors: SHAMOON, Talal, G.; 533 Bryant Street #5, Palo Alto, CA 94301 (US). HILL, Ralph, D.; 224 Dover Street, Los Gatos, CA 94032 (US). RADCLIFFE, Chris, D.; 3654 Farm Hill Boulevard, Redwood City, CA 94061 (US). HWA, John, P.; 503 Lower Vinters Circle, Fremont, CA 94539 (US).

(74) Agents: GARRETT, Arthur, S. et al.; Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P., 1300 I Street, Washington, DC 20005–3315 (US).

(81) Designated States: CA, CN, JP, KR, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHODS AND APPARATUS FOR CONTINUOUS CONTROL AND PROTECTION OF MEDIA CONTENT



(57) Abstract

A novel method and apparatus for protection of streamed media content is disclosed. The apparatus includes control means for governance of content streams or objects, decryption means for decrypting content streams or objects under control of the control means, and feedback means for tracking actual use of content streams or objects. The control means may operate in accordance with rules received as part of the streamed content, or through a side–band channel. The rules may specify allowed uses of the content, including whether or not the content can be copied or transferred, and whether and under what circumstances received content may be "checked out" of one device and used in a second device. The rules may also include or specify budgets, and a requirement that audit information be collected and/or transmitted to an external server. The apparatus may include a media player designed to call plugins to assist in rendering content. A "trust plugin" and its use are disclosed so that a media player designed for use with unprotected content may render protected content without the necessity of requiring any changes to the media player. The streamed content may be in a number of different formats, including MPEG–4, MP3, and the RMFF format.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

METHODS AND APPARATUS FOR CONTINUOUS CONTROL AND PROTECTION OF MEDIA CONTENT

FIELD OF THE INVENTION

This invention relates generally to computer and/or electronic security. More particularly, this invention relates to systems and methods for protection of information in streamed format.

BACKGROUND

Streaming digital media consists generally of sequences of digital information received in a "stream" of packets, and designed to be displayed or rendered. Examples include streamed audio content, streamed video, etc.

Digital media streams are becoming an increasingly significant means of content delivery, and form the basis for several adopted, proposed or de facto standards. The acceptance of this format, however, has been retarded by the ease with which digital media streams can be copied and improperly disseminated, and the consequent reluctance of content owners to allow significant properties to be distributed through streaming digital means. For this reason, there is a need for a methodology by which digital media streams can be protected.

SUMMARY OF THE INVENTION

Consistent with the invention, this specification describes a new architecture for protection of information provided in streamed format. This architecture is described in the context of a generic system which resembles a system to render content encoded pursuant to the MPEG-4 specification (ISO/IEC 14496.1), though with certain modifications, and with the proviso that the described system may differ from the MPEG-4 standard in certain respects. A variety of different embodiments is described, including an MPEG-4 embodiment and a system designed to render content encoded pursuant to the MP3 specification (ISO/IEC TR 11172).

According to aspects of the invention, this architecture involves system design aspects and information format aspects. System design aspects include the incorporation of content protection functionality, control functionality, and feedback enabling control functionality to monitor the activities of the system. Information format aspects include the incorporation of rule/control information into information streams, and the protection of content through mechanisms such as encryption and watermarking.

Petitioner Apple Inc. - Exhibit 1004, p. 5061

Systems and methods consistent with the present invention perform content protection and digital rights management. A streaming media player consistent with the present invention includes a port designed to accept a digital bit stream. The digital bit stream includes content, which is encrypted at least in part, and a secure container including control information designed to control use of the content, including at least one key suitable for decryption of at least a portion of the content. The media player also includes a control arrangement including a means for opening secure containers and extracting cryptographic keys, and means for decrypting the encrypted portion of the content.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an embodiment of the invention and, together with the description, serve to explain the advantages and principles of the invention. In the drawings,

FIG. 1 shows a generic system consistent with the present invention;

FIG. 2 shows an exemplary Header 201 consistent with the present invention;

FIG. 3 shows a general encoding format consistent with the present invention;

FIG. 4 illustrates one manner for storing a representation of a work consistent with the present invention;

FIG. 5 shows an example of a control message format;

FIG. 6 is a flow diagram illustrating one embodiment of the steps which take place using the functional blocks of FIG. 1;

FIG. 7 illustrates a form wherein the control messages may be stored in Control Block 13;

FIG. 8 shows MPEG-4 System 801 consistent with the present invention;

FIG. 9 shows an example of a message format;

FIG. 10 illustrates an IPMP table consistent with the present invention;

FIG. 11 illustrates a system consistent with the present invention;

FIG. 12 illustrates one embodiment of the DigiBox format;

FIG. 13 shows an example of a Real Networks file format (RMFF);

FIG. 14 shows an RNPFF format consistent with the present invention;

FIG. 15 illustrates the flow of changes to data in the Real Networks file format in an architecture consistent with the present invention;

FIG. 16 illustrates a standard Real Networks architecture;

FIG. 17 shows an exemplary architecture in which a trust plugin operates within the overall Real Networks architecture;

Petitioner Apple Inc. - Exhibit 1004, p. 5062

FIG. 18 shows a bit stream format consistent with the principles of the present invention;

FIG. 19 shows one embodiment of protection applied to the MP3 format;

FIG. 20 illustrates one embodiment of an MP3 player designed to process and render protected content;

FIG. 21 illustrates the flow of data in one embodiment in which a protected MPEG-4 file may be created consistent with the present invnetion;

FIG. 22 illustrates the flow of data in one embodiment in which control may be incorporated into an existing MPEG-4 stream consistent with the present invention;

FIG. 23 shows a system consistent with the principles of the present invention;

FIG. 24 shows a system consistent with the principles of the present invention;

FIG. 25 shows an example of an aggregate stream consistent with the present invention;

FIG. 26 illustrates a Header CMPO 2601 consistent with the present invention;

FIG. 27 shows exemplary Content Management Protection Objects consistent with the principles of the present invention; and

FIG. 28 shows an example of a CMPO Data Structure 2801 consistent with the present invention.

DETAILED DESCRIPTION

Reference will now be made in detail to implementations consistent with the principles of the present invention as illustrated in the accompanying drawings.

The following U.S. patents and applications, each of which is assigned to the assignee of the current application, are hereby incorporated in their entirety by reference: Ginter, et al., "Systems and Methods for Secure Transaction Management and Electronic Rights Protection," U.S. Patent Application Serial No. 08/964,333, filed on November 4, 1997 ("Ginter '333"); Ginter, et al., "Trusted Infrastructure Support Systems, Methods and Techniques for Secure electronic commerce, Electronic Transactions, Commerce Process Control Automation, Distributed Computing, and Rights Management, " U.S. Patent Application Serial No. 08/699,712, filed on August 12, 1996 ("Ginter '712") ; Van Wie, et al, "Steganographic Techniques for Securely Delivering Electronic Digital Rights Management Information Over Insecure Communications Channels, U.S. Patent Application Serial No. 08/689,606, filed on August 12, 1996 ("Van Wie") ; Ginter, et. al "Software Tamper Resistance and Secure Communication," U.S. Patent Application Serial No. 08/706,206, filed on August 30, 1996 ("Ginter, '206"); Shear, et al, "Cryptographic Methods, Apparatus & Systems for Storage Media Electronic Rights Management in

Petitioner Apple Inc. - Exhibit 1004, p. 5063

Closed & Connected Appliances, " U.S. Patent Application Serial No. 08/848,077, filed on May 15, 1997 ("Shear"); Collberg et al, "Obfuscation Techniques for Enhancing Software Security, " U.S. Patent Application Serial No. 09/095,346, filed on June 9, 1998 ("Collberg"); Shear, "Database Usage Metering and Protection System and Method," U.S. Patent No. 4,827,508, issued on May 2, 1989 ("Shear Patent").

FIG. 1 illustrates Media System 1, which is capable of accepting, decoding, and rendering streamed multimedia content. This is a generic system, though it includes elements based on the MPEG-4 specification. Media System 1 may include software modules, hardware (including integrated circuits) or a combination. In one embodiment, Media System 1 may include a Protected Processing Environment (PPE) as described in the Ginter '333 application.

In FIG. 1, Bit Stream 2 represents input information received by System 1. Bit Stream 2 may be received through a connection to an external network (e.g., an Internet connection, a cable hookup, radio transmission from a satellite broadcaster, etc.), or may be received from a portable memory device, such as a DVD player.

Bit Stream 2 is made up of a group of related streams of information, including Organization Stream 3, Audio Stream 4, Video Stream 5, Control Stream 6, and Info Stream 31. Each of these streams is encoded into the overall Bit Stream 2. Each of these represents a category of streams, so that, for example, Video Stream 5 may be made up of a number of separate Video Streams.

These streams correspond generally to streams described in the MPEG-4 format as follows:

Organization Stream 3 corresponds generally to the BIFS stream and the OD ("Object Descriptor") stream.

Audio Stream 4 and Video Stream 5 correspond generally to the Audio and Video streams.

Control Stream 6 corresponds generally to the IPMP stream.

Audio Stream 4 includes compressed (and possibly encrypted) digital audio information. This information is used to create the sound rendered and output by Media System 1. Audio Stream 1 may represent multiple audio streams. These multiple streams may act together to make up the audio output, or may represent alternative audio outputs.

Video Stream 5 includes compressed (and possibly encrypted) digital video information. This information is used to create the images and video rendered and output by Media System 1. Video Stream 5 may represent multiple video streams. These multiple streams may act together to make up the video output, or may represent alternative

Petitioner Apple Inc. - Exhibit 1004, p. 5064

video outputs.

Organization Stream 3 includes organizational information and metadata related to the work to be rendered. This information may include a tree or other organizational device which groups audio and video streams into objects. This information may also include metadata associated with the entire work, the objects, or the individual streams.

Control Stream 6 includes control information, divided generally into header information and messages. The header information includes an identifier for each discrete message. The content of the messages, which will be described further below, may include cryptographic keys and rules governing the use of content.

Info Stream 31 carries additional information associated with the content in other components of Bit Stream 2, including but not limited to graphics representing cover art, text for lyrics, coded sheet music or other notation, independent advertising content, concert information, fan club information, and so forth. Info Stream 31 can also carry system management and control information and/or components, such as updates to software or firmware in Media System 1, algorithm implementations for content-specific functions such as watermarking, etc.

Each of these streams is made up of packets of information. In one exemplary embodiment, each packet is 32 bytes in length. Since a single communications channel (e.g., a cable, a bus, an infrared or radio connection) contains packets from each of the streams, packets need to be identified as belonging to a particular stream. In a preferred embodiment, this is done by including a header which identifies a particular stream and specifies the number of following packets which are part of that stream. In another embodiment, each packet may include individual stream information.

Exemplary Header 201 is shown in FIG. 2. This header may generally be used for the Organization, Audio and Video Streams. A header for the Control Stream is described below. Header 201 includes Field 202, which includes a bit pattern identifying Header 201 as a header. Field 203 identifies the particular type of stream (e.g., Audio Stream, Organization Stream, Control Stream, etc.) Field 204 contains an Elementary Stream Identifier (ES_ID), which is used to identify the particular stream, and may be used in cases where multiple streams of a particular stream type may be encountered at the same time. Field 207 contains a time stamp, which is used by the system to synchronize the various streams, including rendering of the streams. Composite Block 11 may, for example, keep track of the elapsed time from the commencement of rendering. Time Stamp 207 may be used by Composite Block 11 to determine when each object is supposed to be rendered. Time Stamp 207 may therefore specify an elapsed time from commencement of rendering,

Petitioner Apple Inc. - Exhibit 1004, p. 5065

and Composite Block 11 may use that elapsed time to determine when to render the associated object.

Field 205 contains a Governance Indicator. Field 206 identifies the number of following packets which are part of the identified stream. In each case, the relevant information is encoded in a binary format. For example, Field 202 might include an arbitrary sequence of bits which is recognized as indicating a header, and Field 203 might include two bits, thereby allowing encoding of four different stream types.

Returning to FIG. 1, System 1 includes Demux 7, which accepts as input Bit Stream 2 and routes individual streams (sometimes referred to as Elementary Streams or "ESs") to appropriate functional blocks of the system.

Bit Stream 2 may be encoded in the format illustrated in FIG. 3. In this figure, Header 301 is encountered in the bit stream, with Packet 302 following, and so on through Packet 308.

When Demux 7 encounters Header 301, Demux 7 identifies Header 301 as a header and uses the header information to identify Packets 302-305 as organization stream packets. Demux 7 uses this information to route these packets to Organization Block 8. Demux 7 handles Header 306 in a similar manner, using the contained information to route Packets 307 and 308 to AV ("Audio Video") Block 9.

AV Block 9 includes Decompressor 10, which accepts Elementary Streams from Audio Stream 4 and Video Stream 5 and decompresses those streams. As decompressed, the stream information is placed in a format which allows it to be manipulated and output (through a video display, speakers, etc.). If multiple streams exist (e.g., two video streams each describing an aspect of a video sequence), AV Block 9 uses the ES_ID to assign each packet to the appropriate stream.

Organization Block 8 stores pointer information identifying particular audio streams and video streams contained in a particular object, as well as metadata information describing, for example, where the object is located, when it is to be displayed (e.g., the time stamp associated with the object), and its relationship to other objects (e.g., is one video object in front of or behind another video object). This organization may be maintained hierarchically, with individual streams represented at the lowest level, groupings of streams into objects at a higher level, complete scenes at a still higher level, and the entire work at the highest level.

FIG. 4 illustrates one manner in which Organization Block 8 may store a representation of a work. In this Figure, Tree 401 represents an entire audiovisual work. Branch 402 represents a high-level organization of the work. This may include, for

Petitioner Apple Inc. - Exhibit 1004, p. 5066

example, all of the video or possibly the audio and video associated with a particular scene.

Sub-Branch 403 represents a group of related video objects. Each such object may include an entire screen, or an individual entity within the screen. For example, Sub-Branch 403 may represent a background which does not change significantly from one shot to the next. If the video is moving between two points of reference (e.g., a conversation, with the camera point of view changing from one face to the other), Sub-Branch 404 could represent a second background, used in the second point of view.

Nodes 405 and 406 may represent particular video objects contained within the related group. Node 405 could, for example, represent a distant mountain range, while Node 406 represents a tree immediately behind one of the characters.

Each of the nodes specifies or contains a particular ES_ID, representing the stream containing the information used by that node. Node 405, for example, contains ES_ID 407, which identifies a particular video stream which contains compressed (and possibly encrypted) digital information representing the mountain range.

Composite Block 11 accepts input from Organization Block 8 and from AV Block 9. Composite Block 11 uses the input from Organization Block 8 to determine which specific audiovisual elements will be needed at any given time, and to determine the organization and relationship of those elements. Composite Block 11 accepts decompressed audiovisual objects from AV Block 9, and organizes those objects as specified by information from Organization Block 8. Composite Block 11 then passes the organized information to Rendering Device 12, which might be a television screen, stereo speakers, etc.

Control Block 13 stores control messages which may be received through Control Stream 6 and/or may be watermarked into or steganographically encoded in other streams, including Audio Stream 4 and Video Stream 5. One control message format is illustrated by FIG. 5, which shows Control Message 501. Control Message 501 is made up of Header 502 and Message 503. Header 502 consists of Field 508, which includes a bit pattern identifying the following information as a header; Stream Type Field 509, which identifies this as a header for the organization stream; ID Field 504, which identifies this particular control message; Pointer Field 505, which identifies those ESs which are controlled by this message; Time Stamp Field 507, which identifies the particular portion of the stream which is controlled by this control message (this may indicate that the entirety of the stream is controlled); and Length Field 506, which specifies the length (in bytes) of Message 503. Message 503 may include packets following Header 502, using the general format shown in FIG. 3. In the example shown, Control Message 501 carries the unique ID 111000,

Petitioner Apple Inc. - Exhibit 1004, p. 5067

encoded in ID Field 504. This control message controls ESs 14 and 95, as indicated by Pointer Field 505. The associated Message contains 1,024 bytes, as indicated by Length Field 506.

In an alternate embodiment, the association of control to content may be made in Organization Block 8, which may store a pointer to particular control messages along with the metadata associated with streams, objects, etc. This may be disadvantageous, however, in that it may be desirable to protect this association from discovery or tampering by users. Since Control Block 13 will generally have to be protected in any event, storing the association in this block may make protection of Organization Block 8 less necessary.

Control Block 13 implements control over System 1 through Control Lines 14, 15 and 16, which control aspects of Organization Block 8, AV Block 9 and Composite Block 11, respectively. Each of these Control Lines may allow two-way communication.

Control Lines 14 and 15 are shown as communicating with AV Block Stream Flow Controller 18 and with Organization Block Stream Flow Controller 17. These Stream Flow Controllers contain functionality controlled by Control Block 13. In the embodiment illustrated, the Stream Flow Controllers are shown as the first stage in a two-stage pipeline, with information being processed by the Stream Flow Controller and then passed on to the associated functional block. This allows isolation of the control functionality from the content manipulation and display functionality of the system, and allows control to be added in without altering the underlying functionality of the blocks. In an alternate embodiment, the Stream Flow Controllers might be integrated directly into the associated functional blocks.

Stream Flow Controllers 17 and 18 contain Cryptographic Engines 19 and 20, respectively. These Cryptographic Engines operate under control of Control Block 13 to decrypt and/or cryptographically validate (e.g., perform secure hashing, message authentication code, and/or digital signature functions) the encrypted packet streams received from Demux 7. Decryption and validation may be selective or optional according to the protection requirements for the stream.

Cryptographic Engines 19 and 20 may be relatively complex, and may, for example, include a validation calculator that performs cryptographic hashing, message authentication code calculation, and/or other cryptographic validation processes. In addition, as is described further below, additional types of governance-related processing may also be used. In one alternative embodiment, a single Stream Flow Controller may be used for both Organization Stream 3 and Audio/Video Streams 4-5. This may reduce the cost of and space used by System 1. These reductions may be significant, since System 1

Petitioner Apple Inc. - Exhibit 1004, p. 5068

may contain multiple AV Blocks, each handling a separate Audio or Video Stream in parallel. This alternative may, however, impose a latency overhead which may be unacceptable in a real-time system.

If the Stream Flow Controllers are concentrated in a single block, they may be incorporated directly into Demux 7, which may handle governance processing prior to routing streams to the functional blocks. Such an embodiment would allow for governed decryption or validation of the entirety of Bit Stream 2, which could occur prior to the routing of streams to individual functional blocks. Encryption of the entirety of Bit Stream 2 (as opposed to individual encryption of individual ESs) might be difficult or impossible without incorporating stream controller functionality into Demux 7, since Demux 7 might otherwise have no ability to detect or read the header information necessary to route streams to functional blocks (that header information presumably being encrypted).

As is noted above, each of the individual streams contained in Bit Stream 2 may be individually encrypted. An encrypted stream may be identified by a particular indicator in the header of the stream, shown in FIG. 2 as Governance Indicator 205.

When a header is passed by Demux 7 to the appropriate functional block, the stream flow controller associated with that block reads the header and determines whether the following packets are encrypted or otherwise subject to governance. If the header indicates that no governance is used, the stream flow controller passes the header and the packets through to the functional blocks with no alteration. Governance Indicator 205 may be designed so that conventionally encoded content (e.g., unprotected MPEG-4 content) is recognized as having no Governance Indicator and therefore passed through for normal processing.

If a stream flow controller detects a set governance indicator, it passes the ES_ID associated with that stream and the time stamp associated with the current packets to Control Block 13 along Control Line 14 or 15. Control Block 13 then uses the ES_ID and time stamp information to identify which control message(s) are associated with that ES. Associated messages are then invoked and possibly processed, as may be used for governance purposes.

A simple governance case is illustrated by FIG. 6, which shows steps which take place using the functional blocks of FIG. 1. In Step 601, Demux 7 encounters a header, and determines that the header is part of the AV stream. In Step 602, Demux 7 passes the header to AV Stream Controller 18. In Step 603, AV Stream Controller 18 reads the header and determines that the governance indicator is set, thereby triggering further processing along Path 604. In Step 605, AV Stream Controller 18 obtains the ES_ID and

Petitioner Apple Inc. - Exhibit 1004, p. 5069

time stamp from the header and transmits these to Control Block 13, along Control Line 15. In Step 606, Control Block 13 looks up the ES_ID and determines that the ES_ID is associated with a particular control message. In Step 611, Control Block 13 uses the time stamp information to choose among control messages, if there is more than one control message associated with a particular ES. In Step 607, Control Block 13 accesses the appropriate control message, and obtains a cryptographic key or keys for decryption and/or validation. In Step 608, Control Block 13 passes the cryptographic key(s) along Control Line 15 to AV Stream Controller 18. In Step 609, AV Stream Controller 18 uses the cryptographic key as an input to Cryptographic Engine 20, which decrypts and/or validates the packets following the header as those packets are received from Demux 7. In Step 610, the decrypted packets are then passed to AV Block 9, which decompresses and processes them in a conventional manner.

Time stamp information may be useful when it is desirable to change the control message applicable to a particular ES. For example, it may be useful to encode different portions of a stream with different keys, so that an attacker breaking one key (or even a number of keys) will not be able to use the content. This can be done by associating a number of control messages with the same stream, with each control message being valid for a particular period. The time stamp information would then be used to choose which control message (and key) to use at a particular time. Alternatively, one control message may be used, but with updated information being passed in through the Control Stream, the updates consisting of a new time stamp and a new key.

In an alternative embodiment, Control Block 13 may proactively send the appropriate keys to the appropriate stream flow controller by using time stamp information to determine when a key will be will be needed. This may reduce overall latency.

Control Line 16 from FIG. 1 comes into play once information has been passed from Organization Block 8 and AV Block 9 to Composite Block 11, and the finished work is prepared for rendering through Rendering Device 12. When Composite Block 11 sends an object to Rendering Device 11, Composite Block 11 sends a start message to Control Block 13. This message identifies the object (including any associated ES_IDs), and specifies the start time of the display (or other rendering) of that object. When an object is no longer being rendered, Composite Block 11 sends an end message to Control Block 13, specifying that rendering of the object has ended, and the time at which the ending occurred. Multiple copies of a particular object may be rendered at the same time. For this reason, start and stop messages sent by Composite Block 11 may include an assigned instance ID, which specifies which instance of an object is being rendered.

Petitioner Apple Inc. - Exhibit 1004, p. 5070

Control Block 13 may store information relating to start and stop times of particular objects, and/or may pass this information to external devices (e.g., External Server 30) through Port 21. This information allows Control Block 13 to keep track not only of which objects have been decrypted, but of which objects have actually been used. This may be used, since System 1 may decrypt, validate, and/or decompress many more objects than are actually used. Control Block 13 can also determine the length of use of objects, and can determine which objects have been used together. Information of this type may be used for sophisticated billing and auditing systems, which are described further below.

Control Line 16 may also be used to control the operation of Composite Block 11. In particular, Control Block 13 may store information specifying when rendering of a particular object is valid, and may keep track of the number of times an object has been rendered. If Control Block 13 determines that an object is being rendered illegally (i.e., in violation of rules controlling rendering), Control Block 13 may terminate operation of Composite Block 11, or may force erasure of the illegal object.

In an alternate embodiment, the level of control provided by Control Line 16 may at least in part be provided without requiring the presence of that line. Instead, Control Block 13 may store a hash of the organization information currently valid for Organization Block 8. This hash may be received through Control Stream 6, or, alternatively, may be generated by Control Block 13 based on the information contained in Organization Block 8.

Control Block 13 may periodically create a hash of the information currently resident in Organization Block 8, and compare that to the stored hash. A difference may indicate that an unauthorized alteration has been made to the information in Organization Block 8, thereby potentially allowing a user to render information in a manner violative of the rules associated with that information. In such an event, Control Block 13 may take appropriate action, including deleting the information currently resident in Organization Block 8.

If System 1 is designed so that Organization Block 8 controls the use of content by Composite Block 11, so that content cannot be rendered except as is specified by the organization information, Control Block 13 may be able to control rendering of information through verifying that the current Organization Block contents match the hash which has been received by Control Block 13, thereby eliminating at least one reason for the presence of Control Line 16.

Control Block 13 may also be responsible for securely validating the origin, integrity, authenticity, or other properties of received content, through cryptographic

Petitioner Apple Inc. - Exhibit 1004, p. 5071

validation means such as secure hashing, message authentication codes, and/or digital signatures.

System 1 may also include an Inter-Rights Point, indicated as IRP 22. IRP 22 is a protected processing environment (e.g., a PPE) in which rules/controls may be processed, and which may store sensitive information, such as cryptographic keys. IRP 22 may be incorporated within Control Block 13, or may be a separate module. As is illustrated, IRP 22 may include CPU 23 (which can be any type of processing unit), Cryptographic Engine 24, Random Number Generator 25, Real Time Clock 26, and Secure Memory 27. In particular embodiments, some of these elements may be omitted, and additional functionality may be included.

Governance Rules

Control messages stored by Control Block 13 may be very complex. FIG. 7 illustrates the form in which the control messages may be stored in Control Block 13, consisting of Array 717. Column 701 consists of the address at which the control messages are stored. Column 702 consists of the identifier for each control message. This function may be combined with that of Column 701, by using the location information of Column 701 as the identifier, or by storing the message in a location which corresponds to the identifier. Column 703 consists of the ES_IDs for each stream controlled by the control message. Column 704 consists of the message itself. Thus, the control message stored at location 1 has the ID 15, and controls stream 903.

In a simple case, the message may include a cryptographic key, used to decrypt the content associated with the stream(s) controlled by the message. This is illustrated by Cryptographic Key 705 from FIG. 7. Cryptographic keys and/or validation values may also be included to permit cryptographic validation of the integrity or origin of the stream.

In a more complex case, the message may include one or more rules designed to govern access to or use of governed content. Rules may fall into a number of categories.

Rules may require that a particular aspect of System 1, or a user of System 1, be verified prior to decryption or use of the governed content. For example, System 1 may include System ID 28, which stores a unique identifier for the system. A particular rule contained in a control message may specify that a particular stream can only be decrypted on a system in which System ID 28 contains a particular value. This is illustrated at row 2 in FIG. 7, in which the message is shown as consisting of a rule and commands. The rule may be implicit, and therefore may not be stored explicitly in the table (e.g. the table may store only the rule, the rule - specific functions (commands) invoked by the rule, or only the functions).

Petitioner Apple Inc. - Exhibit 1004, p. 5072

In this case, when Stream Controller 18 encounters a Header for stream 2031 containing a set governance indicator, Stream Controller 18 passes the associated ES_ID (2031) to Control Block 13. Control Block 13 then uses the ES_ID to identify Control Message 20 which governs stream 2031. Control Message 20 includes Rule 706, which includes (or invokes) Commands 707, and an Authorized System ID 708. Authorized System ID 708 may have been received by System 1, either as part of Control Message 20, or as part of another control message (e.g., Control Message 9), which Control Message 20 could then reference in order to obtain access to the Authorized System ID. Such a case might exist, for example, if a cable subscriber had pre-registered for a premium show. The cable system might recognize that registration, and authorize the user to view the show, by sending to the user an ID corresponding to the System ID.

When Rule 706 is invoked, corresponding Commands 707 access System ID 28 and obtain the system ID number. The commands then compare that number to Authorized System ID 708, specified by Rule 706. If the two numbers match, Commands 707 release Cryptographic Key 709 to Stream Controller 18, which uses Cryptographic Key 709 to decrypt the stream corresponding to ES_ID 2031. If the two numbers do not match, Commands 707 fail to release Cryptographic Key 709, so that Stream Controller 18 is unable to decrypt the stream.

In order to carry out these functions, in one embodiment, Control Block 13 includes, or has access to, a processing unit and memory. The processing unit is preferably capable of executing any of the commands which may be included or invoked by any of the rules. The memory will store the rules and association information (ID of the control message and IDs of any governed ESs).

Since the functions being carried out by Control Block 13 are sensitive, and involve governance of content which may be valuable, Control Block 13 may be partially or completely protected by a barrier which resists tampering and observation. As is described above, the processing unit, secure memory, and various other governance-related elements may be contained in IRP 22, which may be included in or separate from Control Block 13.

Control Block 13 may also carry out somewhat more complex operations. In one example, a control message may require that information from System 1 not only be accessed and compared to expected information, but stored for future use. For example, a control message might allow decryption of a Stream, but only after System ID 28 has been downloaded to and stored in Control Block 13. This would allow a control message to check the stored System ID against System ID 28 on a regular basis, or perhaps on every

Petitioner Apple Inc. - Exhibit 1004, p. 5073

attempted re-viewing of a particular Stream, thereby allowing the control message to insure that the Stream is only played on a single System.

Control Block 13 may also obtain information dynamically. For example, System 1 may include User Interface 29, which can include any type of user input functionality (e.g., hardware buttons, information displayed on a video screen, etc.) A particular rule from a control message may require that the user enter information prior to allowing decryption or use of a stream. That information may, for example, be a password, which the Rule can then check against a stored password to insure that the particular user is authorized to render the stream.

Information obtained from the user might be more complicated. For example, a rule might require that the user input payment or personal information prior to allowing release of a cryptographic key. Payment information could, for example, constitute a credit card or debit card number. Personal information could include the user's name, age, address, email address, phone number, etc. Entered information could then be sent through Port 21 to External Server 30 for verification. Following receipt of a verification message from External Server 30, the Rule could then authorize release of a cryptographic key. Alternatively, Control Block 13 may be designed to operate in an "off-line" mode, storing the information pending later hookup to an external device (or network). In such a case, Control Block 13 might require that a connection be made at periodic intervals, or might limit the number of authorizations which may be obtained pending the establishment of an external connection.

In a somewhat more complex scenario, a control message may include conditional rules. One particular example is illustrated by row 4 of the table shown in FIG. 7, in which Control Message 700 is shown as controlling streams 49-53. Control Message 700 further consists of Rule 710, Commands 711 and Cryptographic Keys 712-716. There could, of course, be a number of additional cryptographic keys stored with the message.

In this case, Rule 710 specifies that a user who agrees to pay a certain amount (or provide a certain amount of information) may view Stream 49, but all other users are required to view Stream 50, or a combination of Streams 49 and 50. In this case, Stream 49 may represent a movie or television program, while Stream 50 represents advertisements. In one embodiment, different portions of Stream 49 may be decrypted with different keys so that, for example, a first portion is decrypted with Key 712, a second portion is decrypted with Key 713, a third portion is decrypted with Key 714, and so on. Rule 710 may include all keys used to decrypt the entirety of Stream 49. When the user initially attempts to access the video encoded in Stream 49, Rule 710 could put up a

Petitioner Apple Inc. - Exhibit 1004, p. 5074

message asking if the user would prefer to use pay for view mode or advertising mode. If the user selects pay for view mode, Rule 710 could store (or transmit) the payment information, and pass Cryptographic Key 712 to Stream Controller 18. Stream Controller 18 could use Cryptographic Key 712 to decrypt the first stream until receipt of a header indicating that a different key is needed to decrypt the following set of packets. Upon request by Stream Controller 18, Control Block 13 would then check to determine that payment had been made, and then release Cryptographic Key 713, which would be used to decrypt the following packets, and so on. Rule 710 could additionally release Cryptographic Key 716, corresponding to Organization Stream 52, which corresponds to video without advertisements.

If, on the other hand, the user had chosen the advertising mode, Rule 710 could release Cryptographic Key 712 to Stream Controller 18 to allow decryption of Stream 49. Rule 710 could also authorize decryption of Stream 50 which contains the advertisements. Rule 710 could further release Cryptographic Key 715 to Organization Block 8. Cryptographic Key 715 matches Organization Stream 51. Organization Stream 51 references the video from Stream 49, but also references advertisements from Stream 50. Rule 710 would refuse to release Cryptographic Key 716, which corresponds to Organization Stream 52, which corresponds to the video without advertisements.

In operation, Control Block 13 could monitor information from Composite Block 11 over Control Line 16. That information could include the identity of each object actually rendered, as well as a start and stop time for the rendering. Control Block 13 could use this information to determine that an advertisement had actually been rendered, prior to releasing Cryptographic Key 713 for decryption of the second portion of video from Stream 49. This feedback loop allows Control Block 13 to be certain that the advertisements are not only being decrypted, but are also being displayed. This may be necessary because Composite Block 11 may be relatively unprotected, thereby allowing an unscrupulous user to remove advertisements before viewing.

A variety of additional relatively complex scenarios are possible. For example, rules from Control Block 13 could customize the programming for a particular geographic location or a particular type of viewer, by using information on the location or the viewer to control conditional decryption or use. This information could be stored in System 1 or entered by the user.

In another example, shown at row 5 of Array 717, Rule 719 may specify Budget 718, which may include information relating to the number of uses available to the user, the amount of money the user has to spend, etc. In operation, Rule 719 may require that

Petitioner Apple Inc. - Exhibit 1004, p. 5075

Budget 718 be securely stored and decremented each time a budgeted activity occurs (e.g., each time the associated work is played). Once the budget reaches zero, Rule 719 may specify that the work may no longer be played, or may display a message to the user indicating that the user may obtain additional budget by, for example, entering a credit card number or password, or contacting an external server.

In another example, a rule may control the ability of a user to copy a work to another device. The rule may, for example, specify that the user is authorized to use the governed work on more than one device, but with only one use being valid at any time. The rule may specify that an indication be securely stored regarding whether the user has "checked out" the work. If the user copies the work to another device (e.g., through Port 21), the rule may require that the work only be transmitted in encrypted form, and that the relevant control messages be transmitted along with it. The rule can further require that an indicator be securely set, and that the indicator be checked each time the user attempts to use or copy the work. If the indicator is set, the rule might require that the work not be decrypted or used, since the user only has the right to use the work on one device at a time, and the indicator establishes that the work is currently "checked out" to another device and has not been checked back in.

The receiving device may include the same type of indicator, and may allow the user to use the work only as long as the indicator is not set. If the user desires to use the work on the original device, the two devices may communicate, with the indicator being set in the second and reset in the first. This allows the work to be stored in two locations, but only used in one.

In another embodiment, the same result may be reached by copying the relevant control message from one device to the other, then erasing it from the original device. Because the control message includes keys used for decryption, this would insure that the work could only be used in one device at a time.

In one embodiment, this technique may be used to communicate digital media files (e.g., music, video, etc.) from a personal computer to a consumer electronics device without allowing the user to make multiple choices for simultaneous use. Thus, a larger, more sophisticated device (e.g., a personal computer), could download a file, then "check out" the file to a portable device lacking certain functions present in the personal computer (e.g., a hand-held music player).

Rules may also be used to specify that an initial user may transfer the file to another user, but only by giving up control over the file. Such rules could operate similarly to the

Petitioner Apple Inc. - Exhibit 1004, p. 5076

technique described above for transferring a file from one device to another, or could require that the original file be entirely erased from the original device after the transfer.

Rules in Control Block 13 may be added or updated through at least two channels. New rules may be obtained through Control Stream 6. If a control message contains an identifier corresponding to a control message already present in Control Block 13, that control message (including contained rules) may overwrite the original control message. A new rule may, for example, be identical to an existing rule, but with a new time stamp and new keys, thereby allowing decryption of a stream which had been encrypted with multiple keys. System 1 may be designed so that certain rules may not be overwritable. This may be enforced by designating certain positions in Array 717 as non-overwritable, or by providing a flag or other indicator to show that a particular rule cannot be overwritten or altered. This would allow for certain types of superdistribution models, including allowing a downstream distributor to add rules without allowing the downstream distributor to remove or alter the rules added by upstream distributors.

In addition, new rules may be encoded into Organization Stream 3, Audio Stream 4, or Video Stream 5, in the form of a watermark or steganographic encoding.

New rules may also be obtained through Port 21. Port 21 may connect to an external device (e.g., a smart card, portable memory, etc.) or may connect to an external network (e.g., External Server 30). Rules may be obtained through Port 21 either in an ad hoc manner, or as a result of requests sent by Control Block 13. For example, Control Message 14 (FIG. 7, row 6) may include a rule specifying that a new rule be downloaded from a particular URL, and used to govern Stream 1201.

Control messages, including rules, may be encoded using secure transmission formats such as DigiBoxes. A DigiBox is a secure container means for delivering a set of business rules, content description information, content decryption information and/or content validation information. One or more DigiBoxes can be placed into the headers of the media content or into data streams within the media.

FIG. 12 illustrates one embodiment of the DigiBox format and the manner in which that format is incorporated into a control message. Control Message 1201 is made up of Control Message Header 1202 and Control Message Contents 1203. As is described elsewhere, Control Message Header 1202 may include information used by Demux 7 (FIG. 1) to appropriately route the message to Control Block 13.

Control Message Contents 1203 of Control Message 1201 consists of DigiBox 1204, and may also include additional information. DigiBox 1204 consists of DigiBox Header 1205, Rules 1206 and Data 1207. Rules 1206 may include one or more rules. Data

Petitioner Apple Inc. - Exhibit 1004, p. 5077

1207 may include various types of data, including ES_ID 1208, Cryptographic Key 1209, and Validation Data 1210. Data 1207 may also include cryptographic information such as a specification of the encryption algorithm, chaining modes used with the algorithm, keys and initialization vectors used by the decryption and chaining.

Initialization vectors contained within Data 1207 are similar to cryptographic keys, in that they constitute input to the original encryption process and therefore are necessary for decryption. In one well-known prior art embodiment, the initialization vectors may be generated by starting with a base initialization vector (a 64 bit random number) and xor'ing in the frame number or start time for the content item.

Validation Data 1210 contained within Data 1207 may include cryptographic has or authentication values, cryptographic keys for calculating keyed authentication values (e.g., message authentication codes), digital signatures, and/or public key certificates used in validating digital certificates.

Thus, the DigiBox may incorporate the information described above as part of the control message, including the rules, the stream ID and the cryptographic keys and values.

In an alternative embodiment, DigiBox Header 1205 may be designed so that it can be read by Demux 7 and routed to Control Block 13. In such an embodiment, DigiBox 1204 would itself constitute the entirety of the control message, thus obviating the need to nest DigiBox 1204 within Control Message 1201.

Some or all of the contents of DigiBox 1204 will generally be encrypted. This may include Rules 1206, Data 1207, and possibly some or all of Header 1205. System 1 may be designed so that a DigiBox may only be decrypted (opened) in a protected environment such as IRP 22. In an alternate embodiment, Control Block 13 may directly incorporate the functionality of IRP 22, so that the DigiBox may be opened in Control Block 13 without the necessity of routing the DigiBox to IRP 22 for processing. In one embodiment, the cryptographic key used to decrypt DigiBox 1204 may be stored in IRP 22 (or Control Block 13), so that the DigiBox can only be opened in that protected environment.

Rules 1206 are rules governing access to or use of DigiBox Data 1207. In one embodiment, these rules do not directly control the governed streams. Since Cryptographic Key 1209 can only be accessed and used through compliance with Rules 1206, however, Rules 1206 in fact indirectly control the governed streams, since those streams can only be decrypted through use of the key, which can only be obtained in compliance with the rules. In another embodiment, Data 1207 may include additional rules, which may be extracted from the DigiBox and stored in a table such as Array 717 of FIG. 7.

Petitioner Apple Inc. - Exhibit 1004, p. 5078

The rules governing access to or use of a DigiBox may accompany the DigiBox, (as shown in FIG. 12) or may be separately transmitted, in which event Rules 1206 would contain a pointer or reference to the rules used to access Data 1207. Upon receipt of a DigiBox, Control Block 13 may receive rules separately through Control Stream 6, or may request and receive rules through Port 21.

Pipelined Implementation

One potential drawback to the system illustrated in FIG.1 consists of the fact that the system introduces complexity and feedback into a pipelined system designed to render content in real time. The rendering pipeline generally consists of Demux 7, Organization Block 8 and AV Block 9, Composite Block 11 and Rendering Device 12. Because content is received in a streamed fashion, and must be rendered in real time, pipelined processing must occur in a highly efficient manner, under tight time constraints. A failure to process within the time available may mean that output to Rendering Device 12 may be interrupted, or that incoming Bit Stream 2 may overflow available buffers, thereby causing the loss of some portion of the incoming data.

An alternative embodiment of System 1 is designed to address these problems, although at a possible cost in the ability to use standard system components and a possible cost in overall system security. This alternative embodiment is illustrated in FIG. 11, which shows System 1101.

System 1101 is similar to System 1 from FIG. 1 in many respects. It receives Bit Stream 1102, which consists of Organization Stream 1103, Audio Stream 1104, Video Stream 1105 and Control Stream 1106. These streams are received by Demux 1107, which passes Organization Stream 1103 to Organization Block and passes Audio Stream 1104 and Video Stream 1105 to AV Block 1109. Organization Block 1108 and AV Block 1109 operate similarly to their counterparts in FIG. 1, and pass information to Composite Block 1110, which organizes the information into a coherent whole and passes it to Rendering Device 1111. Streams sent to Organization Block 1108 are decrypted and/or validated by Stream Flow Controller 1112, and streams sent to AV Block 1109 are decrypted and/or validated by Stream Flow Controller 1113.

System 1101 differs from System 1, however, in that control and feedback are distributed, and integrated directly into the processing and rendering pipeline. System 1101 thus lacks a separate control block, and also lacks a feedback path back from the Composite Block 1110.

In System 1101, control is exercised directly at Organization Block 1108 and AV Block 1109. As in System 1, cryptographic keys are received through Control Stream 1106

Petitioner Apple Inc. - Exhibit 1004, p. 5079

(in an alternative embodiment, the keys could be incorporated directly into header or other information in Organization Stream 1103 or Audio/Video Streams 1104 and 1105). Those keys are included in a data format which includes information regarding the stream type of the encrypted content and, if multiple stream types are possible, an identifier for the particular controlled stream.

When Demux 1107 encounters a key in Control Stream 1106, it reads the information relating to the stream type, and routes the key to the appropriate stream flow controller. If Demux 1107 encounters a key designated for decryption or validation of Organization Stream 1103, for example, it routes that key to Stream Flow Controller 1112.

Stream Flow Controller 1112 stores received keys in Storage Location 1114. Storage Location 1114 stores the keys and also stores an indicator of the controlled stream ID.

Stream Flow Controller 1112 includes Cryptographic Engine 1115, which uses the received keys to decrypt and/or validate encrypted and/or protected portions of Organization Stream 1103. The keys may themselves be received in an encrypted manner, in order to provide some degree of security. In such a case, Stream Flow Controller may use a variety of techniques to decrypt the key, including using stored information as a key, or as a key seed. That stored information could, for example, constitute a "meta-key" provided earlier through Bit Stream 1102 or through a separate port.

Stream Flow Controller 1113, associated with AV Block 1109, contains a corresponding Storage Location 1116 and Cryptographic Engine 1117, and operates in a manner similar to the operation described for Stream Flow Controller 1112.

This implementation avoids the latency penalty which may be inherent in the necessity for communication between stream flow controllers and a separate control block.

This alternate implementation may also eliminate the feedback channel from the composite block (FIG.1, Control Line 16). This feedback channel may be used in order to insure that the content being passed from Composite Block 11 to Rendering Device 12 is content that has been authorized for rendering. In the alternate embodiment shown in FIG.11, this feedback channel does not exist. Instead, this implementation relies on the fact that Composite Block 1110 depends upon information from Organization Block 1108 to determine the exact structure of the information being sent to Rendering Device 1111. Composite Block 1110 cannot composite information in a manner contrary to the organization dictated by Organization Block 1108.

In one embodiment, this control by Organization Block 1108 may be sufficient to obviate the need for any feedback, since Organization Block 1108 may be designed so that

Petitioner Apple Inc. - Exhibit 1004, p. 5080

it accepts information only through Stream Controller 1112, and Stream Controller 1112 may be designed so that it only decrypts or validates information under the control of rules stored in Storage Location 1114.

In such an embodiment, security may be further increased by incorporating Secure Memory 1118 into Organization Block 1108. Secure Memory 1118 may store a copy or hash of the organization tree validly decrypted by Stream Controller 1112, and in current use in Main Organization Block Memory 1119. Organization Block 1108 may be used to periodically compare the organization tree stored in Main Organization Block Memory 1119 to the tree stored in Secure Memory 1118. If a discrepancy is spotted, this may indicate that an attacker has altered the organization tree stored in Main Organization Block 1119, thereby possibly allowing for the rendering of content in violation of applicable rules. Under such circumstances, Organization Block 1108 may be used to take protective measures, including replacing the contents of Main Organization Block Memory 1119 with the contents of Secure Memory 1118.

MPEG-4 Implementation

The generic system described above may be embodied in an MPEG-4 system, as illustrated in FIG. 8, which shows MPEG-4 System 801.

MPEG-4 System 801 accepts MPEG-4 Bit Stream 802 as input. MPEG-4 Bit Stream 802 includes BIFS Stream 803, OD Stream 804, Audio Stream 805, Video Stream 806 and IPMP Stream 807. These streams are passed to Demux 808, which examines header information and routes packets as appropriate, to BIFS 809, AVO 810, OD 811 or IPMP System 812.

IPMP System 812 receives IPMP messages through IPMP Stream 807. Those messages may include header information identifying the particular message, as well as an associated IPMP message. The IPMP message may include control information, which may include a cryptographic key, validation information, and/or may include complex governance rules, as are described above.

Stream Controllers 813, 814 and 815 act to decrypt, validate, and/or govern streams passed to BIFS 809, AVO 810 and OD 811, respectively.

OD 811 holds object descriptors, which contain metadata describing particular objects. This metadata includes an identifier of the particular Elementary Stream or streams which include the object, and may also include a pointer to a particular IPMP message which governs the object. Alternatively, the relationship between IPMP messages and particular objects or streams may be stored in a table or other form within IPMP System 812.

Petitioner Apple Inc. - Exhibit 1004, p. 5081

IPMP System 812 may exercise control over other functional blocks through Control Lines 816, 817, 818 and 819, each of which may transmit control/governance signals from IPMP System 812 and information or requests from other functional blocks to IPMP System 812. The information requests may include an ES_ID and a time stamp, which IPMP System 812 may use to determine which particular message (e.g., key) should be used and when.

In an alternative embodiment, IPMP System 812 may exercise control over Composite and Render 821 by receiving a hash of the currently valid BIFS tree (possibly through IPMP stream 807), and periodically checking the hash against the BIFS tree stored in BIFS 809. Because BIFS 809 controls the manner in which Composite and Render 821 renders information, if IPMP System 812 confirms that the current BIFS tree is the same as the authorized tree received through BIFS Stream 803, IPMP System 812 can confirm that the proper content is being rendered, even without receiving feedback directly from Composite and Render 821. This may be necessary, since BIFS 809 may communicate with Port 822, which may allow a user to insert information into BIFS 809, thereby creating a possibility that a user could insert an unauthorized BIFS tree and thereby gain unauthorized access to content.

When a stream controller receives encrypted or otherwise governed information, it may send the ES_ID and time stamp directly to IPMP System 812. Alternatively, it may send this information to OD 811, which may reply with the ID of the IPMP message which governs that object or stream. The stream controller can then use that IPMP message ID to request decryption, validation, and/or governance from IPMP System 812. Alternatively, OD 811 can pass the IPMP ID to IPMP System 812, which can initiate contact with the appropriate stream controller.

IPMP System 812 may obtain IPMP information through two channels other than IPMP Stream 807. The first of these channels is Port 820, which may be directly connected to a device or memory (e.g., a smart card, a DVD disk, etc.) or to an external network (e.g., the Internet). An IPMP message may contain a pointer to information obtainable through Port 812, such as a URL, address on a DVD disk, etc. That URL may contain specific controls needed by the IPMP message, or may contain ancillary required information, such as, for example, information relating to the budget of a particular user.

IPMP System 812 may also obtain IPMP information through OD updates contained in OD Stream 804. OD Stream 804 contains metadata identifying particular objects. A particular OD Message may take the format shown in FIG. 9. In this figure, OD Message 901 includes Header 902, which identifies the following packets as part of the OD

Petitioner Apple Inc. - Exhibit 1004, p. 5082

stream, and indicates the number of packets. OD Message 901 further consists of Message 903, which includes a series of Pointers 904 and associated Metadata 905. Each Pointer 904 identifies a particular Elementary Stream, and the associated metadata is applicable to that stream. Finally, OD Message 901 may contain an IPMP Pointer 906, which identifies a particular IPMP message.

In aggregate, the information contained in OD Message 901 constitutes an object descriptor, since it identifies and describes each elementary stream which makes up the object, and identifies the IPMP message which governs the object. OD Message 901 may be stored in OD 811, along with other messages, each constituting an object descriptor.

Object descriptors stored in OD 811 may be updated through OD Stream 804, which may pass through a new object descriptor corresponding to the same object. The new object descriptor then overwrites the existing object descriptor. This mechanism may be used to change the IPMP message which controls a particular object, by using a new object descriptor which is identical to the existing object descriptor, with the exception of the IPMP pointer.

OD Stream 804 can also carry IPMP_DescriptorUpdate messages. Each such message may have the same format as IPMP messages carried on the IPMP stream, including an IPMP ID and an IPMP message.

IPMP_DescriptorUpdate messages may be stored in a table or array in OD 811, or may be passed to IPMP System 812, where they may overwrite existing stored IPMP messages, or may add to the stored messages.

Since IPMP information may be separately conveyed through the OD stream or the IPMP stream, MPEG-4 System 801 may be designed so that it only accepts information through one or the other of these channels.

In another embodiment, the existence of the two channels may be used to allow multi-stage distribution, with governance added at later stages, but with no risk that later alterations may override governance added at an earlier stage.

Such a system is illustrated in FIG. 10. In this Figure, IPMP System 812 includes IPMP Table 1002, which has slots for 256 IPMP messages. This table stores the IPMP_ID implicitly, as the location at which the information is stored, shown in Column 1003. The IPMP message associated with IPMP_ID 4, for example, is stored at slot 4 of IPMP Table 1002.

Each location in IPMP Table 1002 includes Valid Indicator 1004 and Source Indicator 1005. Valid Indicator 1004 is set for a particular location when an IPMP message is stored at that location. This allows IPMP System 812 to identify slots which are

Petitioner Apple Inc. - Exhibit 1004, p. 5083

unfilled, which otherwise might be difficult, since at start-up the slots may be filled with random information. This also allows IPMP System 812 to identify messages which are no longer valid and which may be replaced. Valid Indicator 1004 may store time stamp information for the period during which the message is valid with IPMP System 812 determining validity by checking the stored time stamp information against the currently valid time.

Source Indicator 1005 is set based on whether the associated IPMP message was received from IPMP Stream 807 or from OD Stream 804.

These indicators allow IPMP System 812 to establish a hierarchy of messages, and to control the manner in which messages are added and updated. IPMP System 812 may be designed to evaluate the indicators for a particular location once a message is received corresponding to that location. If the valid indicator is set to invalid, IPMP System 812 may be designed to automatically write the IPMP message into that slot. If the valid indicator is set to valid, IPMP System 812 may then be designed to check the source indicator. If the source indicator indicates that the associated message was received through OD Stream 804, IPMP System 1812 may be designed to overwrite the existing message with the new message. If, however, the source indicator indicates that the associated message was received through IPMP Stream 807, IPMP System 812 may be designed to check the source of the new message. That check may be accomplished by examining the header associated with the new message, to determine if the new message was part of OD Stream 804 or part of IPMP Stream 807. Alternatively, IPMP System 812 may derive this information by determining whether the message was received directly from Demux 808 or through OD 811.

If the new message came through IPMP Stream 807, IPMP System 812 may be designed to store the new message in Table 1002, overwriting the existing message. If the new message came through OD Stream 804, on the other hand, IPMP System 812 may be designed to reject the new message.

This message hierarchy can be used to allow for a hierarchy of control. A studio, for example, may encode a movie in MPEG-4 format. The studio may store IPMP messages in the IPMP stream. Those messages may include a requirement that IPMP System 812 require that a trailer for another movie from the same studio be displayed prior to the display of the feature movie. IPMP System 812 could be used to monitor the beginning and end of rendering of the trailer (using feedback through Control Line 819) to ensure that the entire trailer plays, and that the user does not fast-forward through it.

Petitioner Apple Inc. - Exhibit 1004, p. 5084

The movie studio could encrypt the various elementary streams, including the IPMP stream. The movie studio could then provide the movie to a distributor, such as a cable channel. The movie studio could provide the distributor with a key enabling the distributor to decrypt the OD stream (or could leave the OD stream unencrypted), and the ability to insert new messages in that stream. The cable channel could, for example, include a rule in the OD stream specifying that the IPMP system check to determine if a user has paid for premium viewing, decrypt the movie if premium viewing has been paid for, but insert advertisements (and require that they be rendered) if premium viewing has not been paid for).

The cable channel would therefore have the ability to add its own rules into the MPEG-4 Bit Stream, but with no risk that the cable channel would eliminate or alter the rules used by the movie studio (e.g., by changing the trailer from a movie being promoted by the studio to a rival movie being promoted by the cable channel). The studio's rules could specify the types of new rules which would be allowed through the OD stream, thereby providing the studio a high degree of control.

This same mechanism could be used to allow superdistribution of content, possibly from one user to another. A user could be provided with a programming interface enabling the insertion of messages into the OD stream. A user might, for example, insert a message requiring that a payment of $1.00 be made to the user's account before the movie can be viewed. The user could then provide the movie to another user (or distribute it through a medium whereby copying is uncontrolled, such as the Internet), and still receive payment. Because the user's rules could not overrule the studio's rules, however, the studio could be certain that its rules would be observed. Those might include rules specifying the types of rules a user would be allowed to add (e.g., limiting the price for redistribution).

MPEG-4 System 801 may also be designed to include a particular type of IPMP system, which may be incompatible with IPMP systems that may be designed into other MPEG-4 systems. This may be possible because the MPEG-4 standard does not specify the format of the information contained in the IPMP stream, thereby allowing different content providers to encode information in differing manners.

IPMP System 812 in MPEG-4 System 801 may be designed for an environment in which differing IPMP formats exist. That system may scan the IPMP stream for headers that are compatible with IPMP System 812. All other headers (and associated packets) may be discarded. Such a mechanism would allow content providers to incorporate the same IPMP message in multiple formats, without any concern that encountering an unfamiliar format would cause an IPMP system to fail. In particular, IPMP headers can

Petitioner Apple Inc. - Exhibit 1004, p. 5085

incorporate an IPMP System Type Identifier. Those identifiers could be assigned by a central authority, to avoid the possibility that two incompatible systems might choose the same identifier.

IPMP System 801 might be designed to be compatible with multiple formats. In such a case, IPMP System 801 might scan headers to locate the first header containing an IPMP System Identifier compatible with IPMP System 801. IPMP System 801 could then select only headers corresponding to that IPMP System Identifier, discarding all other headers, including headers incorporating alternate IPMP System Identifiers also recognized by the IPMP system.

Such a design would allow a content provider to provide multiple formats, and to order them from most to least preferred, by including the most preferred format first, the second most preferred format second, and so on. Since IPMP System 801 locks onto the first compatible format it finds, this ordering in IPMP Stream 801 would insure that the IPMP system chose the format most desired by the content provider.

Even if different IPMP formats are used, content will probably be encoded (and encrypted) using a single algorithm, since sending multiple versions of content would impose a significant bandwidth burden. Thus, ordinarily it will be necessary for content to be encrypted using a recognized and common encryption scheme. One such scheme could use the DES algorithm in output feedback mode.

This method of screening IPMP headers, and locking onto a particular format may also be used to customize an MPEG-4 bit Stream for the functional capabilities of a particular MPEG-4 system. Systems capable of rendering MPEG-4 content may span a considerable range of functionality, from high-end home theaters to handheld devices. Governance options suitable for one type of system may be irrelevant to other systems.

For example, MPEG-4 System 801 may include a connection to the Internet through Port 820, whereas a second MPEG-4 system (for example a handheld Walkman-like device) may lack such a connection. A content provider might want to provide an option to a viewer, allowing the viewer to see content for free in return for providing information about the viewer. The content provider could insert a rule asking the user whether the user wants to view the content at a cost, or enter identification information. The rule could then send the information through a port to the Internet, to a URL specified in the rule. A site at that URL could then evaluate the user information, and download advertisements targeted to the particular user.

Although this might be a valuable option for a content provider, it obviously makes no sense for a device which is not necessarily connected to the Internet. It would make no

Petitioner Apple Inc. - Exhibit 1004, p. 5086

sense to present this option to the user of a non-connected device, since even if that user entered the information, the rule would have no way to provide the information to an external URL or download the advertisements. In such a case, the content provider might prefer to require that the user watch preselected ads contained in the original MPEG-4 bit stream.

Header information in the IPMP stream could be used to customize an MPEG-4 bit stream for particular devices. As with the IPMP System Type information, IPMP Header information could include MPEG-4 System Types. These could include 8 or 16-bit values, with particular features represented by bit maps. Thus, the presence of a bit at position 2, for example, could indicate that a device includes a persistent connection to the Internet.

An IPMP system could then evaluate the headers, and lock on to the first header describing functionality less than or equal to the functionality contained in the MPEG-4 device in which the IPMP system is embedded. If the header constituted a complete match for the functionality of the MPEG-4 device, the IPMP system could then cease looking. If the header constitutes less than a complete match (e.g., a header for a system which has an Internet connection, but lacks a digital output port, when the system includes both), the IPMP system can lock on to that header, but continue to scan for closer matches, locking on to a closer match if and when one is found.

The IPMP messages identified by a particular header would be those suited for the particular functionality of the MPEG-4 device, and would allow for customization of the MPEG-4 bit stream for that functionality. In the context of the example given above, the IPMP system for an MPEG-4 device containing an Internet connection would lock on to a particular header, and would download the IPMP messages characterized by that header. Those messages would prompt the user for information, would provide that information to the URL, and would authorize decryption and rendering of the movie, with the advertisements inserted at the appropriate spot.

In the case of an MPEG-4 device without an Internet connection, on the other hand, the IPMP system would lock onto a set of headers lacking the bit indicating an Internet connection, and would download the rules associated with that header. Those rules might not provide any option to the user. The rules might allow decryption of the content, but would also specify decryption of an additional ES from the MPEG-4 stream. That additional ES would contain the advertisements, and the IPMP system would require decryption and rendering of the advertisements, checking Control Line 819 to make certain that this had occurred. In the case of the system with the Internet connection, however, the rules allowing decryption and requiring rendering of the ES containing the advertisements

Petitioner Apple Inc. - Exhibit 1004, p. 5087

would never be loaded, since those rules would be contained within messages identified by the wrong type of header. The advertisement ES would therefore never be decrypted and would be ignored by the MPEG-4 device.

FIG. 21 illustrates one manner in which a protected MPEG-4 file may be created. In this figure, CreateBox 2101 represents a DigiBox creation utility, which accepts keys and rules. In one embodiment, CreateBox 2101 may pass these keys and rules to IRP 2102 and receive DigiBox 2103 from IRP 2102. In another embodiment, IRP 2102 may be incorporated into CreateBox 2101, which accepts keys and rules and outputs DigiBox 2103.

DigiBox 2103 contains governance rules, initialization vectors and keys. DigiBox 2103 is passed from CreateBox 2101 to Bif Encoder 2104. Bif Encoder 2104 may be conventional, with the exception that it is designed to accept and process DigiBoxes such as DigiBox 2103. Bif Encoder 2104 also accepts a .txt file containing a scene graph, and initial object descriptor commands.

Bif Encoder 2104 outputs a .bif file, containing the scene graph stream (in compressed binary form) and a .od file, containing the initial object descriptor commands, the object descriptor stream, and DigiBox 2103.

Bif Encoder 2104 passes the .bif file and the .od file to Mux 2105. Mux 2105 also accepts compressed audio and video files, as well as a .scr file that contains the stream description. Mux 2105 creates IPMP streams, descriptors and messages, encrypts the content streams, interleaves the received streams, and outputs Protected MPEG-4 Content File 2106, consisting of Initial Object Descriptor 2107 and Encrypted Content 2108. Initial Object Descriptor 2107 contains DigiBox 2103, as well as other information. Encrypted Content 2108 may include a scene graph stream (i.e., a BIFS stream), an object descriptor stream, IPMP streams, and encrypted content streams.

If DigiBox 2103 contains all keys and rules necessary to render all of the content, it may be unnecessary for Mux 2105 to create any IPMP streams. If additional keys or rules may be necessary for at least a portion of the content, Mux 2105 may incorporate those rules and keys into one or more additional DigiBoxes, and incorporate those DigiBoxes either in the IPMP stream or in the OD update stream.

FIG. 22 illustrates one manner in which control may be incorporated into an existing MPEG-4 stream. In this figure, Unprotected MPEG-4 Content File 2201 includes Initial Object Descriptor 2202 and Content 2203. The content may include a scene description stream (or BIF stream), an object descriptor stream, a video stream, an audio stream, and possibly additional content streams.

Petitioner Apple Inc. - Exhibit 1004, p. 5088

Unprotected MPEG-4 Content File 2201 is passed to Repackager 2204, which also accepts keys and rules. Repackager 2204 passes the keys and rules to IRP 2205, and receives DigiBox 2206 in return, containing keys, rules and initialization vectors. In an alternate embodiment, IRP 2205 may be incorporated directly into Repackager 2204.

Repackager 2204 demuxes Unprotected MPEG-4 Content File 2201. It inserts DigiBox 2206 into the Initial Object Descriptor and encrypts the various content streams. Repackager 2204 also adds the IPMP stream, if this is necessary (including if additional DigiBoxes are necessary).

Repackager 2204 outputs Protected MPEG-4 Content File 2207, consisting of Initial Object Descriptor 2208 (including DigiBox 2206) and Encrypted Content 2209 (consisting of various streams, including the IPMP streams, if necessary).

Real Networks Implementation

In one embodiment, the elements described above may be used in connection with information encoded in compliance with formats established by Real Networks, Inc.

The Real Networks file format (RMFF) is illustrated in FIG. 13. This format includes a block of headers at the beginning (Header 1301), followed by a collection of content packets (Content 1302), followed by an index used for seek and goto operations (Index 1303). Each file can contain several streams of different types. For each stream, there is a "Media Properties Header" (1304) used to describe the format of the media content (e.g., compression format) and provide stream specific information (e.g., parameters for the decompressor).

Real Networks streams can be protected by inserting a DigiBox into Header 1301 and encrypting the data packets contained in Content 1302. The altered format is illustrated in FIG.14, which shows Header 1401, including Media Properties Headers 1402 and 1403, which in turn contain DigiBoxes 1404 and 1405, respectively. The format also includes encrypted Content 1406 and Index 1407.

In one embodiment, the declared type of the data is changed from the standard Real Networks format to a new type (e.g., RNWK_Protected.) The old type is then saved. Changing the type forces the Real Networks player to load a "Trust Plugin," since this Plugin is registered as the only decoder module that can process streams of type "RNWK-Protected." The Trust Plugin opens the DigiBox, gets approval from the user, if it is needed, determines the original content type, loads a decoder plugin for the original content, and then decrypts and/or validates the content, passing it to the content decoder plugin to be decompressed and presented to the user.

Petitioner Apple Inc. - Exhibit 1004, p. 5089

In one embodiment, the specific alterations made to the Real Networks file format are the following:

- Increase the preroll time to force larger buffers on playback. In a current embodiment, an increase of 3 seconds is used. Larger buffers are needed because of the extra steps needed to decrypt the content.
- Modify each stream-specific header by changing the mime type to "RNWK-Protected", saving the old mime type in the decoder specific information and adding a content identifier and DigiBox to the decoder specific information. The DigiBox contains the key, initialization vector (IV), version information, and watermarking instructions. The key, IV and content identifier are generated automatically, or can be provided as command-line parameters. The same key, IV and content identifier are used for every stream.
- Content packets are selectively encrypted. In one embodiment, content packets whose start time in milliseconds is in the first half-second of each 5 seconds (i.e., starttime % 5000 < 500) are encrypted. This encrypts approximately one-tenth of the content reducing encryption and decryption costs, and damages the content, sufficiently to prevent resale. The encryption algorithm can be DES using output-feedback mode or any similar algorithm. The initialization vector is computed for each packet by xoring the stream's IV with the packet's start time in milliseconds. Some information unique to the stream should also be xored into the IV. In one embodiment, the same IV is used for multiple packets whenever two or more streams have packets with the same start time. This usually happens for the first packet in each stream since they usually have start time 0. Other than the first packet, it is rare to have two packets have the same start time.

In one embodiment, these changes to the Real Networks file format are accomplished as is shown in FIG. 15. As is illustrated, RMFF file 1501 is formatted in the standard Real Networks RMFF format. This file is passed to Packager 1502. Also passed to Packager 1502 is Rights File 1503. Packager 1503 generates Protected RMFF File 1504, which includes various alterations as described above and as listed in FIG. 15, including the incorporation of one or more DigiBoxes in the header, encryption of the content, modification of the mime type, etc.

In one embodiment, the trust plugin described above is illustrated in FIGs. 16 and 17. FIG. 16 illustrates the standard Real Networks architecture. File 1601 (e.g., a streaming audio file in Real Networks format) is provided to Real Networks G2 Client

Petitioner Apple Inc. - Exhibit 1004, p. 5090

Core 1602. File 1601 may be provided to RealNetworks G2 Client Core 1602 from Server 1603, or through Direct Connection 1604.

Upon receipt of File 1601, Real Networks G2 Client Core 1602 accesses a rendering plugin appropriate to File 1601, based on information which is obtained from the header associated with File 1601. Rendering Plugins 1605 and 1606 are shown. If File 1601 is of a type which cannot be rendered by either Rendering Plugin 1605 or Rendering Plugin 1606, Real Networks G2 Client Core 1602 may attempt to access an appropriate plugin, e.g., by asking for the user's assistance or by accessing a site associated with the particular file type.

Rendering Plug-In 1605 or 1606 processes File 1601 in a conventional manner. This processing most likely includes decompression of File 1601, and may include other types of processing useful for rendering the content. Once this processing is complete (keeping in mind that the content is streamed, so that processing may be occurring on one set of packets at the same time that another set of packets is being rendered), File 1601 is passed back to Real Networks G2 Client Core 1602, which then passes the information to Rendering Device 1607. Rendering Device 1607 may, for example, be a set of stereo speakers, a television receiver, etc.

FIG. 17 illustrates the manner in which a trust plugin operates within the overall Real Networks architecture. Much of the architecture illustrated in FIG. 17 is the same as that illustrated in FIG. 16. Thus, File 1701 is provided to Real Networks G2 Client Core 1702 through Server 1703 or through Direct Connection 1704. The file is processed by Real Networks G2 Client Core 1702, using plugins, including Rendering Plugins 1705 and 1706, and is then passed to Rendering Device 1707.

FIG. 17 differs from FIG. 16 in its incorporation of Trust Plugins 1708 and 1709, and IRP 1710. When initially registered with Real Networks G2 Client Core 1702, Trust Plugins 1708 and 1709 inform Real Networks G2 Client Core 1702 that they can process content of type RNWK-Protected. Whenever Real Networks G2 Client Core 1702 encounters a stream of this type, it is then enabled to create an instance of the trust plugin to process the stream, e.g., Trust Plugin 1708. It then passes the stream to the trust plugin.

The stream passed to Trust Plugin 1708 may be in the format shown in FIG. 14. In such a case, Trust Plugin 1708 extracts DigiBox 1404 from Media Properties Header 1402. It also extracts the content id and original mime type from Media Properties Header 1402. The Trust Plugin first checks to see if any other stream with the same content identifier has been opened. If so, then DigiBox 1404 is not processed further. Instead, the key and IV from the box for this other stream are used. This avoids the time cost of opening a second

Petitioner Apple Inc. - Exhibit 1004, p. 5091

box. Also, this ensures that a user is only asked to pay once even if there are multiple protected streams. By sharing content ids, keys, and IVs, several files can be played with the user only paying once. This is useful when SMIL is used to play several RMFF files as a single presentation.

In an alternate and possibly more secure embodiment, this check is not performed, and the key and IV from the current DigiBox are used even if another stream with the content identifier has already been opened.

If no other stream has been identified with the same content identifier, Trust Plugin 1708 passes DigiBox 1404 to IRP 1710. IRP 1710 may be a software process running on the same computer as Real Networks G2 Client Core and Trust Plugin 1708. IRP 1710 may run in a protected environment or may incorporate tamper resistance techniques designed to render IRP 1710 resistant to attack.

IRP 1708 may process DigiBox 1404 and extract a cryptographic key and an IV, which may then be passed to Trust Plugin 1708. Trust Plugin 1708 may then use this information to decrypt Encrypted Contents 1406.

Trust Plugin 1708 uses the original mime type information extracted from Media Properties Header 1402 to create an instance of the rendering plugin to be used for the content (e.g., Rendering Plugin 1705). Once this is done, Trust Plugin 1708 behaves like an ordinary rendering plugin to the Real Networks G2 Client Core 1702, in that Real Networks G2 Client Core 1702 passes streamed information to Trust Plugin 1708, which decrypts that information and passes it to Rendering Plugin 1705. From the perspective of Real Networks G2 Client Core 1702, Trust Plugin 1708 constitutes the appropriate rendering pluin, and the core is not aware that the information is being passed by Trust Plugin 1708 to a second plugin (e.g., Rendering Plugin 1705).

Similarly, from the point of view of Rendering Plugin 1705, Trust Plugin 1708 behaves like Real Networks G2 Client Core 1702. Thus although Rendering Plugin 1705 receives decrypted stream information from Trust Plugin 1708, Rendering Plugin 1705 operates exactly as if the information had been received directly from Real Networks G2 Client Core 1702. In this manner, content formatted for Rendering Plugin 1705 may instead be first processed by Trust Plugin 1708, without requiring any alteration to Real Networks G2 Client Core 1702 or Rendering Plugin 1705.

Trust Plugin 1708 may also perform other processing that may be helpful for security purposes. For example, Trust Plugin 1708 may watermark the decrypted file prior to passing it to Rendering Plugin 1705, keeping in mind that the watermark algorithm must be such that it will survive decompression of the file by Rendering Plugin 1705.

Petitioner Apple Inc. - Exhibit 1004, p. 5092

MP3 Embodiment

The techniques described above can also be applied to MP3 streaming content.

The MP-3 specification does not define a standard file format, but does define a bit stream, which is illustrated in FIG.18. In FIG. 18, MP-3 Bit Stream 1801 includes Content 1802. Content 1802 is divided into frames, shown as Frame 1803, Frame 1804 and Frame 1805. The dots between Frame 1804 and 1805 symbolize the fact that Content 1802 may include a large number of frames.

Each frame includes its own small header, shown in FIG. 18 as Headers 1806, 1807 and 1808.

Many MP3 players support a small trailer defined by the ID3 V1 specification, shown as Trailer 1809. This is a 128 byte trailer for carrying fields like artist, title and year, shown as Fields 1810, 1811 and 1812. The ID3 V1 trailer is ignored by players not designed to read such trailers, since it does not appear to be valid MP3 data.

FIG. 19 shows one embodiment of protection applied to the MP3 format. This protected format constitutes File 1908 and includes the following items:

- Unencrypted MP3 Content 1912. This is the first information encountered by a player, and will be rendered by any standard MP3 player. It can include a message to the user indicating that the content is protected and providing instructions as to how the content can be accessed (e.g., a URL for a trust plugin, instructions on payment mechanisms, etc.) Unencrypted MP3 Content 1912 may include a "teaser," consisting of an initial portion of the content (e.g., 30 seconds), which is rendered at no cost, thereby allowing a user to sample the content prior to making a decision to purchase it.
- Encrypted MP-3 Content 1901, which may include thousands of MP-3 frames. In one embodiment, the first eight frames out of every 32 frames are encrypted. Thus, one-quareter of the frames are rendered unuseable unless a player is able to decrypt them. In practice, this may render the content un-sellable or unuseable, without imposing excessive encryption or decryption costs. To further reduce encryption and decryption costs, only 32 bytes in each frame are encrypted. In a current embodiment, these are the first 32 bytes after the header and CRC information. In a different embodiment, a different 32 bytes may be encrypted in every frame. In a current embodiment, the content is encrypted with the DES using algorithm output-feedback mode. The initial IV for the file is randomly generated and then xored with the frame number to generate a unique IV for each frame.

Many alternate embodiments may exist, including encrypting more or less information, and using different encryption algorithms.

- ID3 V1 Trailer 1902, including 128 bytes.

Petitioner Apple Inc. - Exhibit 1004, p. 5093

- Content ID 1903, including 16 bytes. This is used by the player application to avoid opening DigiBoxes which it has already opened.
- DigiBox 1904, which may comprise approximately 18K bytes. It includes Key 1909, IV 1910 and Watermarking Instructions 1911. Watermarking Instructions 1911 may be used in a process of watermarking the associated content.
- Address 1905, which contains the address in the file of Content ID 1903 and consists of 4 bytes.
- Trust ID 1906, which identifies this trusted MP-3 file and consists of 16 bytes.
- ID3 V1 Trailer 1907, which is a copy of Trailer 1902.

A conventional MP3 player encountering File 1908 would be unable to render Content 1901, since at least a portion of that content is encrypted. Such a player would most likely read through to Trailer 1902 and cease processing at that point. A conventional player looking for the ID3 trailer information will seek to the end and find it.

FIG. 20 illustrates one embodiment of an MP3 player designed to process and render protected content. This figure shows MP3 Player 2001, which includes Buffer 2006 and Decompressor 2007, and renders content to Rendering Device 2008. In one embodiment, this is a modified version of a player distributed by Sonique.

Player 2001 obtains Protected MP3 File 2002 through any standard interface. Protected MP3 File 2002 may have the format illustrated in FIG. 19.

When Player 2001 is asked to play Protected MP3 File 2002, Player 2001 first calls Trust Plug-In 2003, which includes Approval Function 2009 and Decrypt Function 2005. Trust Plugin 2003 calls Approval Function 2009 to determine if Protected MP3 File 2002 is protected and whether authorization exists to play the file. Approval Function 2009 is first given a pointer to Protected MP3 File 2002. It then checks Protected MP3 File 2002 for the presence of Trust ID 1906. If Trust ID 1906 is not found, Approval Function 2009 returns an indicator that the file is not protected. Player 2001 then proceeds to render the file as a normal MP3 file.

If Trust ID 1906 is found, Approval Function 2009 checks Content ID 1903 to see if it matches the Content ID of a file that has already been opened.

If Protected MP3 File 2002 has not been previously opened, DigiBox 1904 is retrieved by Approval Function 2009, and is passed to IRP 2004, which may include software running in a protected environment, or incorporating tamper resistance. IRP 2004 attempts to open DigiBox 1904 in compliance with the rules associated with that DigiBox. One such rule may require, for example, that the user indicate assent to pay for use of the content. If DigiBox 1904 cannot be opened (e.g., the user refuses to pay) a value is

Petitioner Apple Inc. - Exhibit 1004, p. 5094

returned to Approval Function 2009 indicating that the file is protected and may not be played.

If DigiBox 1904 is opened in compliance with applicable rules, the key and IV are retrieved and passed to Decrypt Function 2005. The key and IV are stored with the content id for later re-use and Decrypt Function 2005 is initialized. This may improve overall system performance, since it reduces the number of times a DigiBox must be opened. Each such action may introduce significant latency.

On the other hand, storing this information in unprotected memory may reduce overall system security. Security may be enhanced either by not storing this information (thereby requiring that each DigiBox be opened, even if the corresponding file has already been opened through another DigiBox), or by storing this information in a protected form or in a secure location.

The stored key, IV and content id are referenced when Approval Function 2009 first checks Content ID 1903 to determine if it matches the Content ID of an already opened file. If the new Content ID matches a stored Content ID, Decrypt Function 2005 is reinitialized using the stored key and IV corresponding to the matching content id an value indicating that this is a protected file for which play is authorized is returned to Approval Function 2009.

Once Protected MP3 File 2002 has been opened, each time Player 2001 needs a packet, Player 2001 reads it into Buffer 2006, strips off the header and CRC and passes the remaining data and a frame number to Decrypt Function 2005, which decrypts the frame if necessary, and returns it to Player 2001.

In a current embodiment, although audio content is encrypted, headers or trailers are not encrypted. This allows the Player 2001 to process information in headers or trailers without intervention from Approval Function 2009 or Decrypt Function 2005. This allows Player 2001 to place information such as playing time, artist and title into a playlist display, and initialize Decompressor 2007, without any action required from Trust Plugin 2003.

Commerce Appliance Embodiment

This section will describe an embodiment, comprising a Commerce Appliance architecture designed to allow persistent control of digital works in consumer electronics devices. Although this is described as a separate embodiment, it should be understood that the features of this embodiment may be combined with, or supplant, the features of any of the embodiments provided elsewhere in this description.

In one embodiment, this section will describe modifications to the MPEG-4 standard designed to support the association of persistent rules and controls with MPEG-4

Petitioner Apple Inc. - Exhibit 1004, p. 5095

content, as well as elements necessary for a Commerce Appliance to use such content. This is intended, however, merely as an example.

In one embodiment, shown in FIG. 23, each Commerce Appliance 2301 includes a CMPS ("Content Management and Protection System") 2302. Each CMPS is responsible for governing the use of controlled content, including decrypting the content and ensuring that the content is only used as permitted by associated rules.

Each governed digital work is associated with one or more CMPOs (Content Management Protection Object), e.g., CMPOs 2303. Each CMPO may specify rules governing the use of the digital work, and may include keys used to decrypt the work.

CMPOs may be organized in an hierarchical fashion. In one embodiment, a content aggregator (e.g., a cable channel, a web site, etc.) may specify a Channel CMPO ("CCMPO") used to associate certain global rules with all content present on that channel. Each independent work may in turn have an associated Master CMPO ("MCMPO") used to associate rules applicable to the work as a whole. Each object (or Elementary Stream, in MPEG-4) may have associated with it a CMPO containing rules governing the particular object.

In one exemplary application, Commerce Appliance 2301 may be an MPEG-4 player containing CMPS 2302. Upon receipt of a user command to play a particular work, CMPS 2302 may download a MCMPO associated with the work and obtain rules, which may include conditions required for decryption and viewing of the work. If the rules are satisfied, CMPS 2302 may use keys from the MCMPO to decrypt any Elementary Streams ("ES"), and may pass the decrypted ESs into the buffers. Composition and rendering of the MPEG-4 work may thereafter proceeds according to the MPEG-4 standard, except that any storage location or bus which may contain the work in the clear must be secure, and CMPS 2302 may have the ability to govern downstream processing, as well as to obtain information regarding which AVOs were actually released for viewing.

In a variation, the process of obtaining and governing the work may include downloading a CCMPO which applies rules governing this and other works. If rules contained in the CCMPO are satisfied, CMPS 2302 may obtain a key used to decrypt the MCMPO associated with the particular work to be viewed.

In another variation, a CMPO may be associated with each ES. In this variation, the MCMPO supplies one or more keys for decryption of each CMPO, and each CMPO may in turn supply a key for decryption of the associated ES.

Commerce Appliance 2301 is a content-rendering device which includes the capability of supporting distributed, peer management of content related rights by securely

Petitioner Apple Inc. - Exhibit 1004, p. 5096

applying rules and controls to govern the use of content. Commerce Appliance 2301 may include general-purpose functions devoted to acquisition and managed rendering of content (e.g., a DVD (and/or any other optical disk format) player is able to play a DVD (and/or any other optical disk format) disk and output content to a television.) Commerce Appliance 2301 may make use of any of the means for protecting and using digital content on high capacity optical disk, in one non-limiting example, a DVD disk, as described in the aforementioned Shear patent application.

Commerce Appliance 2301 also includes special-purpose functions relating to other management and protection of content functions. These special-purpose functions may be supported by one or more embedded or otherwise included CMPS 2302 in the form of a single CMPS or a cooperative CMPS arrangement, and may include a user interface (e.g., User Interface 2304) designed to display control-related information to the user and/or to receive control-related information and directions from the user. Commerce Appliance 2301 may also be designed so that it is networkable with other Commerce Appliances (e.g., a set-top box connected to a DVD player and a digital television) and/or with other devices, such as a computer arrangement, which may also include one or more CMPSs.

An important form of Commerce Appliance specifically anticipates secure coupling on a periodic or continual fashion with a computer managed docking environment (e.g., a standalone computer or other computer managed device which itself may be a Commerce Appliance) where the one or more CMPSs of the Commerce Appliance interoperate with the docking environment to form a single user arrangement whose performance of certain functions and/or certain content usage events is enabled by such inter-operation through, at least in part, cooperation between CMPSs and content usage management information of the Commerce Appliance and the trust environment capabilities of the docking environment, (e.g., further one or more CMPSs and content usage management information, such as, for example, information provided by use of CI).

An exemplary Commerce Appliance may be designed to comply with the emerging MPEG-4 standard for the formatting, multiplexing, transmission, compositing, and rendering of video and other types of information.

Commerce Appliance 2301 may be any computing device, one non-limiting example of which is a Personal Computer (PC) that includes MPEG-4 software (and/or hardware) for rendering content. In accordance with the present invention, the PC may also use one or more CMPSs as described herein.

The commerce appliance function is not restricted to streamed channel content but may include various browser-type applications consisting of aggregated composite content

Petitioner Apple Inc. - Exhibit 1004, p. 5097

such as still imagery, text, synthetic and natural video and audio and functional content such as applets, animation models and so on. these devices include browsers, set-top boxes, etc.

Content Management and Protection System (CMPS)

Each commerce appliance includes one or more CMPS (e.g., CMPS 2302). The CMPS is responsible for invocation and application of rules and controls, including the use of rules and controls to govern the manner in which controlled content is used.

Particular functions of CMPS 2302 include the following:

(a) Identification and interpretation of rules.

CMPS 2302 must determine which rules are to be applied, and must determine how those rules are to be interpreted in light of existing state information. In one embodiment, this requires that CMPS 2302 obtain and decrypt one or more CMPOs 2303 associated with a work.

(b) Identification of content associated with particular rules.

CMPS 2302 must determine which content is governed by particular one or more rules. This may be accomplished by obtaining information from one or more CMPOs 2303 and/or other CI. In one embodiment, a CCMPO may identify a set of works, a MCMPO may identify a particular work and a CMPO may identify a particular ES or Audio Visual Object ("AVO").

(c) Decryption of content as allowed by the rules.

CMPS 2302 may be designed so that all content is routed through CMPS 2302 for decryption, prior to reinsertion into the data flow required by the relevant standard. In the case of MPEG-4, for example, the output from Demux 2305 may be fed into CMPS 2302. CMPS 2302 may then decrypt the content and, if relevant rules and controls are satisfied, feed the content into the MPEG-4 buffers. From that point, the data flow associated with the content may be as described by MPEG-4.

(d) Control of content based on rules.

CMPS 2302 may be used to control usage of content after the initial decryption, for example, through the use of secure event management as described in the incorporated Ginter '333 patent application. In the case of MPEG-4 systems, this may require that CMPS 2302 exercise control over hardware and/or software which performs the following functions: demuxing (performed by Demux 2305), decompression/buffering/decode into AVOs (performed by Scene Descriptor Graph 2306, AVO Decode 2307 and Object Descriptors 2308), scene rendering (performed in Composite and Render 2309).

Petitioner Apple Inc. - Exhibit 1004, p. 5098

CMPS 2302 may also be used to control use and consequences according to: (1) generational copy protection rules such as the CGMS and/or SGMS standards; (2) various Conditional Access control methods, such as those proposed and/or implemented by NDS as described in MPEG-4 document M2959, DAVIC "Copyright Control Framework" document, and in other publications; (3) a Rights Management Language, such as those proposed in the Ginter '333 patent application and/or as described by U.S. Patent No. 5,638, 443 to Stefik, et al.; (4) use policies described in accordance with AT&T's Policy Maker, as described by Blaze, Feigenbaum, and Lacy; (5) the CCI layer bits for IEEE 1394 serial bus transmission as specified by the DTDG subgroup of the DVD Copy Protection Technical Working Group and/or as implemented by the Hitachi, Intel, Matsushita, Sony and Toshiba proposed standard (hereafter "the five company proposal"); (6) controls transmitted using any secure container technology such as, for example, IBM Cryptolope; (7) any other means for specifying use rules and consequences.

(e) Monitoring use of content.

CMPS 2302 may be used to monitor content to: (i) ensure that rules are being complied with; (ii) ensure that no attempts are being made to tamper with the system or protected content; and (iii) record information used by rules, including usage information needed for payment purposes.

(f) Updating user budgets.

CMPS 2302 may be used to update user or other budgets to reflect usage.

(g) Exhaust information.

CMPS 2302 may be used to output payment and usage information ("exhaust information") to external processes, including one or more Commerce Utility Systems.

(h) Hardware identification and configuration.

(i) Obtaining new, additional, and/or augmented rules from an external process, one non-limiting example of which is a Rights and Permission Clearinghouse as described in the incorporated Shear patent application.

(j) Receiving keys, digital credentials, such as certificates, and/or administrative information, from certifying authorities, deployment managers, clearinghouses, and/or other trusted infrastructure services.

(k) Securely sending and/or receiving user and/or appliance profiling and/or attribute information.

(l) Securely identifying a user or a member of a class of users who requests content and/or CMPO and/or CMPS usage.

Petitioner Apple Inc. - Exhibit 1004, p. 5099

(m) Securely certifying or otherwise guaranteeing the authenticity of application code, for example certifying within CMPO 2301 and/or CMPS 2302 that application code containing rules and/or other application information, such as information written in Java code for conditional execution within a Commerce Appliance, and/or that executes at least in part outside of CMPO 2301 and/or CMPS 2302, has not been altered and/or has been delivered by a guaranteed (e.g., trusted) party.

(n) Securely processing independently delivered CI, such as described in the incorporated Ginter '333 patent application, to perform content usage control that protects the rights of plural, independent parties in a commerce value chain.

(o) Securely performing watermarking (including, for example fingerprinting) functions, for example as described in the Ginter '333 patent application and as incorporated herein, for example including interpreting watermarking information to control content usage and/or to issue an event message, wherein such event message may be reported back to a remote authority, such as, for example, a MCMPO rights clearinghouse management location.

CMPS 2302 may be used to identify and record the current hardware configuration of the Commerce Appliance and any connected devices (e.g., which loudspeakers are available, identification of attached monitors, including whether particular monitors have digital output ports, etc.) If attached devices (such as loudspeakers) also include CMPSs, the CMPSs may be used to communicate for purposes of coordination (e.g., a CMPS in a set-top box and/or loudspeaker arrangement may communicate with a CMPS in a downstream digital television or other display device to establish which CMPS will be responsible for governance or the nature of cooperative governance through a virtual rights process, said process optionally involving a rights authority server that may find, locate, provide, aggregate, distribute, and/or manage rights processes, such as described in the aforementioned Shear patent application, for employing plural CMPSs, for example, for a single user content processing and usage arrangement).

The present invention includes arrangements comprising plural Commerce Appliances and/or CMPSs in one or more user locations, non-limiting examples of which include a home, apartment, loft, office, and/or vehicle, such as a car, truck, sports utility vehicle, boat, ship, or airplane, that may communicate among themselves at least occasionally and may comprise a virtual network that operates in a logically cooperative manner, through at least in part the use of such CMPSs, to ensure optimal commercial flexibility and efficiency and the enforcement of rights of commerce value chain participants, including financial and copyright rights of providers, infrastructure rights of

Petitioner Apple Inc. - Exhibit 1004, p. 5100

appliance providers, societal rights of government and/or societal bodies, and privacy rights of all parties, including consumers. Information related to interaction among such a network of value chain participants, including content usage auditing, content usage consequence, and CI specification, can be securely, variably reported to parties having right to such information, through, at least in part, use of such CMPSs, for example, as described in the aforementioned Ginter '712 patent application regarding the information reporting functioning of VDE nodes.

In one embodiment, shown in FIG. 24, CMPS 2401 consists of special-purpose hardware and resident software or firmware. These include the following:

(a) One or more processors or microcontrollers e.g. CPU 2402. CPU 2402 controls the overall processing of CMPS 2401, including execution of any necessary software.

(b) One or more external communications ports, e.g., Port 2403. Port 2403 communicates with External Network 2404, which may include LANs, WANs or distributed networks such as the Internet. External communications ports may also include one or more IEEE 1394 serial bus interfaces.

(c) Memory 2405. Types of memories which may be included in Memory 2405-- and examples of the information they may store -- are the following:

i. ROM 2406. ROM 2406 may include any information which is permanently stored in CMPS 2401, such as (1) CMPS Operating System 2407 and/or CMPS BIOS 2408, (2) Rules/Controls 2409 which are permanently stored in the CMPS; (3) Control Primitives 2410 which may be used to build rules or controls; (4) Keys 2411 associated with the CMPS, including a Public/Private Key Pair; (5) one or more Certificates 2412 designed to identify CMPS 2401 and/or the device, including version information; (6) Hardware Signature Information 2413 used to check for tampering (e.g., a hashed signature reflecting the expected hardware state of the device).

ii. RAM 2414. RAM 2414 may hold current state information needed by CMPS 2401, as well as information temporarily stored by CMPS 2401 for later use. Information stored in RAM 2414 may include the following: (1) Software 2415 currently executing in CPU 2402; (2) CMPOs 2416 which are currently active; (3) Content Object Identification 2417 of those content objects which are currently active (in an MPEG 4 system this would constitute, for example, an identification of active AVOs); (4) Rules 2418 which are currently active; (5) State Information 2419 regarding the current state of use of content, including an identification of any higher-order organization (in an MPEG-4 system this would constitute an identification of the scene descriptor tree and the current

Petitioner Apple Inc. - Exhibit 1004, p. 5101

state of composition and rendering); (6) Stored Exhaust Information 2420 relating to use and/or the user, designed for external transmission; (7) Updated Budget Information 2421; (8) Content 2422; (9) Active Content Class Information 2423; and (10) Active User Identification 2424, including identification characteristic information.

iii. NVRAM 2425 (e.g., flash memory). This type of memory may hold information which is persistent but changeable, including at least some: (1) Budget Information 2426; (2) User Information 2427, such as identification, credit card numbers; preferred clearinghouses and other Commerce Utility Systems; (3) User Preferences 2428, such as preferences, profiles, and/or attribute information; and (4) Appliance Information 2429, such as attribution and/or state information.

The types of information described above and stored in CMPS Memory 2405 may be stored in alternative of the above memory types, for example, certain budget information may be located in ROM, information regarding specific one or more clearinghouses may be stored in ROM, certain active information may be moved into NVRAM, etc.

Budget information may include stored budgets made up of, for example:

(1) electronic cash;

(2) pre-authorized uses (e.g., based on a prepayment, the user has the right to watch 12 hours of programming).

(3) Security budgets related to patterns reflecting abnormal and/or unauthorized usage, for example, as described in the incorporated Shear patent, wherein such budgets restrict and/or report certain cumulative usage conduct.

(4) electronic credit, including credit resulting from usage events such as attention to promotional material and/or the playing of multiple works from one or more classes of works (e.g., certain publisher's works) triggering a credit or cash refund event and/or a discount on future playing of one or more of such publisher's works, such as other works provided by such publisher.

User information may include the following types of information for one or more authorized users of the Commerce Appliance:

(1) Name, address, telephone number, social security number or other identifier

(2) Information used to authenticate the user, which may include a user selected password and/or biometric data, such as fingerprints, retinal data, etc.

(3) User public/private key pair

Petitioner Apple Inc. - Exhibit 1004, p. 5102

(4) User attribute and/or profiling information.

iv. Removable Memory 2430. This may include any type of removable memory storage device, such as smart cards, floppy disks or DVD disks. If the commerce appliance is designed to play content received on removable memory devices (e.g., a DVD player), that capability may be used for purposes of the CMPS.

Memory 2405 may include a protected database, in which certain control, budget, audit, security, and/or cryptographic information is stored in secure memory, with complete information stored in an encrypted fashion in unsecure memory.

(d) Encryption/Decryption Engine 2431. CMPS 2401 must include a facility for decrypting received information, including content and CMPOs and/or other. CMPS 2401 may also include a facility for encrypting information if such information is to be transmitted outside the secure boundaries of CMPS 2401. This may include exhaust sent to clearinghouses or other external repositories; and content sent across unsecured buses for usage, such as content sent across IEEE 1394 Serial Bus 2432 to a computer central processing arrangement or to a viewing device such as a monitor, wherein a receiving CMPS may be employed to control such content's usage, including, for example, decrypting such content, as appropriate. Encryption/Decryption Engine 2431 may include a Random Number Generator 2433 used for the creation of keys or key pairs that can be used to identify and assure the uniqueness of CMPSs and support the opening of secure communication channels between such secure content control secure encryption/decryption arrangements.

(e) Secure Clock/Calendar 2434. CMPS 2401 may include Secure Clock/Calendar 2434 designed to provide absolute information regarding the date and time of day, information regarding elapsed absolute time, and/or relative timing information used to determine the elapsed time of operations performed by the system. Secure Clock/Calendar 2434 may include Battery Back Up 2435. It may further include Sync Mechanism 2436 for synchronization with outside timing information, used to recover the correct time in the event of a power loss, and/or to check for tampering.

(f) Interface 2437 to blocks used for content rendering and display. This interface is used for controlling rendering and display, based on rules, and for obtaining feedback information, which may be used for budgeting purposes or for providing information to outside servers (e.g., information on which content was actually displayed, which choices the user invoked, etc.) In the case of an MPEG-4 player such as is shown in

Petitioner Apple Inc. - Exhibit 1004, p. 5103

FIG. 23, this may include control over Commerce Appliance circuitry which handles, for example, buffering, the scene descriptor graph, AVO decode, object descriptors and composite and rendering (e.g., Control Lines 2310, 2311 and 2312).

Feedback Path 2313 from Composite and Render block 2309 may allow CMPS 2302 to determine whether and when content has actually been released to the viewer. For example, Composite and Render block 2309 can issue a start event to CMPS 2302 when an AVO object is released for viewing, and can issue a stop event to CMPS 2302 when the AVO object is no longer being viewed.

Feedback from Composite and Render block 2309 may also be used to detect tampering, by allowing CMPS 2302 to match the identification of the objects actually released for viewing with the identification of the objects authorized for release. Start and end time may also be compared with the expected elapsed time, with a mismatch possibly indicative of the occurrence of an unauthorized event.

In one embodiment, the following protocol may be used for feedback data:

start <id>, T, <instance number><clock time><rendering options>

Sent if elementary stream <id> is reachable in the SD-graph at time *T*, but not at time *T-1*.

end <id>, T, <instance number><clock time><rendering options>

T constitutes presentation time, clock time constitutes the wall clock time, including day and date information, and rendering options may include such information as QoS and rate of play (e.g., fast forward).

Sent if elementary stream **<id>** is reachable in the SD-graph at time *T-1* but not at time *T*. A SD-graph stream is reachable if, during traversal of the SD-graph for display update, the renderer encounters a node that the SD-graph update stream <id> created or modified. This implies that all nodes in the tree need an update history list. This list need not be as large as the number of streams. Further, it can be labeled to indicate if the CMPS will be watching for stream, if not labeled it will not record them. An AV elementary stream is reachable if the stream's content was rendered.

For SD-graph update streams, the object instance number is ignored. For AV streams, the instance number can be used to disambiguate the case where the display shows two or more instances of the same data stream simultaneously. Instance numbers do not have to count up. In this case, they are simply a unique id that allows the CMPS to match a start event with an end event.

In a second embodiment, CMPS 2302 may include some special purpose hardware in combination with general purpose hardware which is also used for other functions of the

Petitioner Apple Inc. - Exhibit 1004, p. 5104

device. In this embodiment, care must be taken to ensure that commercially trusted CMPS functions are performed in a secure and tamper-resistant manner, despite the use of general purpose hardware. Each of the elements recited above may include dedicated CMPS functions and general purpose device functions:

(a) CPU/microcontroller. This may include one or more devices. If more than one device is included (e.g., a CPU and a DSP, a math coprocessor or a commerce coprocessor), these devices may be included within the same package, which may be rendered tamper-resistant, or the devices may communicate on a secure bus. The CPU may include two modes: a secure CMPS mode, and an unsecure general purpose mode. The secure CMPS mode may allow addressing of secure memory locations unavailable to the processor in general purpose mode. This may be accomplished, for example, by circuitry which remaps some of the available memory space, so that, in unsecure mode, the CPU cannot address secure memory locations.

(b) External communications ports. If the device, for example, a Commerce Appliance, is capable of receiving content or other information through a communications port (e.g., a cable connection, an Internet connection), this communications port can be used for CMPS purposes. In such a case, CMPS accesses to the external communications port is preferably designed to avoid or minimize interference with the use of such port for receipt of content.

(c) Memory. In some applications and embodiments, it is possible to operate a Commerce Appliance without NVRAM, wherein information that may be needed for CMPS operation that would employ NVRAM would be loaded into RAM, as required. ROM, RAM and NVRAM may be shared between CMPS uses and general uses. This can be accomplished in any of the following ways, or in a combination of these ways: (1) Some memory space may be rendered off-limits to general purpose uses, for example by remapping; (2) the entirety of the memory may be rendered secure, so that even portions of the memory being used for non-secure purposes cannot be observed or changed except in a secure and authorized manner; (3) CMPS information may be stored in an encrypted fashion, though this requires at least some RAM to be secure, since the CMPS will require direct access to unencrypted information stored in RAM.

(d) Encryption/decryption engine. Encryption and decryption functions, including key generation, may be handled by special purpose software running on a general purpose processor arrangement, particularly, for example, a floating point processor or DSP arrangement. That processor arrangement may also be used for purposes of decompressing and displaying content and/or for handling watermarking/fingerprinting

Petitioner Apple Inc. - Exhibit 1004, p. 5105

insertion and/or reading. Alternatively, the device may include native encryption and decryption functions. For example, various emerging standards may require at least some degree of encryption and decryption of content designed to be passed across unsecure buses within and among devices such as DVD players, such as the "five company proposal" and other IEEE 1394 related initiatives. Circuitry designed to perform such encryption and decryption may also be usable for CMPS applications.

(e) Secure clock/calendar. The underlying device may already require at least some clock information. MPEG-4, for example, requires the use of clock information for synchronization of Elementary Streams. A secure CMPS clock can also be used for such purposes.

In a third embodiment, CMPS 2302 can be primarily software designed to run on a general purpose device which may include certain minimal security-related features. In such a case, CMPS 2302 may be received in the same channel as the content, or in a side-band channel. An I-CMPO and/or other CI may specify a particular type of CMPS, which Commerce Appliance 2301 must either have or acquire (e.g., download from a location specified by the I-CMPO), or CMPS 2302 may be included, for example, with an I-CMPO.

A software CMPS runs on the CPU of the Commerce Appliance. This approach may be inherently less secure than the use of dedicated hardware. If the Commerce Appliance includes secure hardware, the software CMPS may constitute a downloadable OS and/or BIOS which customizes the hardware for a particular type of commerce application.

In one embodiment, a software CMPS may make use of one or more software tamper resistance means that can materially "harden" software. These means include software obfuscation techniques that use algorithmic means to make it very difficult to reverse engineer some or all of a CMPS, and further make it difficult to generalize from a reverse engineering of a given one or more CMPS. Such obfuscation is preferably independent of source code and object code can be different for different CMPSs and different platforms, adding further complexity and separation of roles. Such obfuscation can be employed "independently" to both CI, such as an CMPO, as well as to some or all of the CMPS itself, thus obscuring both the processing environment and executable code for a process. The approach is also applicable for integrated software and hardware implementation CMPS implementations described above. Other tamper resistance means can also be employed, including using "hiding places" for storing certain state information in obscure and unexpected locations, such as locations in NV memory used for other purposes, and data hiding techniques such as watermarking/fingerprinting.

Petitioner Apple Inc. - Exhibit 1004, p. 5106

Association of CMPS With a Commerce Appliance

A CMPS may be permanently attached to a particular device, or may be partially or fully removable. A removable CMPS may include software which is securely loaded into a Commerce Appliance, and/or removable hardware. A removable CMPS may be personalized to one or more particular users, including user keys, budget information, preferences, etc., thereby allowing different users to use the same Commerce Appliance without commingling budgets and/or other rights, etc.

A CMPS may be designed for operation with certain types of content and/or for operation with certain types of business models. A Commerce Appliance may include more than one type of CMPS. For example, a Commerce Appliance designed to accept and display content pursuant to different standards may include one CMPS for each type of format. In addition, a Commerce Appliance may include a CMPS provided by a particular provider, designed to preferentially display certain types of content and to preferentially bill for such content through a particular channel (e.g., billing to one or more particular credit cards and/or using a particular one or more clearinghouses).

Source of Rules

The CMPS must recognize those rules which are to be applied to particular content. Such rules may be received by the CMPS from a variety of sources, depending on the particular embodiment used:

(a) CMPO. The rules may be included within a CMPO (e.g., CMPO 2303) and/or other CI. The CMPO and/or other CI may be incorporated within a content object or stream (as, e.g., a header on an MPEG-4 ES), and/or may be contained within a dedicated content object or stream encoded and received as per the underlying standard (e.g., an MPEG-4 CMPO ES), and/or may be received outside the normal content stream, in which event it may not be encoded as per the underlying standard (e.g., a CMPS received as an encrypted object through a sideband channel).

(b) CMPS. Rules may be permanently and/or persistently stored within a CMPS, e.g., Rules 2409. A CMPS may include default rules designed to handle certain situations, for example, where no CMPO and/or other necessary CI is received (e.g., content encoded under an earlier version of the standard which did not incorporate CMPOs, including MPEG-4 version 1). Complete rules which are stored within the CMPS may be directly or indirectly invoked by a CMPO and/or other CI. This may occur through the CI identifying particular rules through a pointer, and/or it may occur through the CI identifying itself and the general class of control it requires, with the CMPS then applying particular rules specific to that CMPS.

Petitioner Apple Inc. - Exhibit 1004, p. 5107

Rule "primitives" may also be stored within the CMPS (e.g., Control Primitives 2410). The CMPO and/or other CI may invoke these primitives by including a sequence of macro-type commands, each of which triggers a sequence of CMPS primitives.

(c) User. The user may be given the ability to create rules relating to the particular user's preferences. Such rules will generally be allowed to further restrict the use of content, but not to expand the use of content beyond that which would otherwise be allowed. Examples include: (a) rules designed to require that certain types of content (e.g., adult movies) only be accessible after entry of a password and/or only to certain CMPS users (e.g. adults, not children, as, for example, specified by parents and/or a societal body such as a government agency); (b) rules designed to require that only particular users be allowed to invoke operations requiring payment beyond a certain limit and/or aggregate payment over a certain amount.

The user may be allowed to create templates of rules such as described in the aforementioned Ginter '333 patent application (and incorporated herein). In addition, a CMPS arrangement, and/or a particular CMPO and/or other CI, may restrict the rules the user is allowed to specify. For example, a CI may specify that a user can copy a work, but cannot add rules to the work restricting the ability of a recipient to make additional copies (or to be able to view, but only after a payment to the first user). User supplied one or more rules may govern the use of – including privacy restrictions related to -- payment, audit, profiling, preference, and/or any other kind of information (e.g., information result as a consequence of the use of a CMPS arrangement, including, for example, use of secured content). Such user supplied one or more rules can be associated with the user and/or one or more Commerce Appliances in a user arrangement, whether or not the information is aggregated according to one or more criteria, and whether or not user and/or appliance identification information is removed during aggregation and/or subsequent reporting, distribution, or any other kind of use.

The ability to allow the user to specify rules allows the CMPS to subsume (and thereby replace) V-chips, since a parent can use content rating information to specify precisely what types of information each viewer will be allowed to watch (e.g., violent content can only be displayed after entry of a certain password and/or other identifier, including, for example, insertion of a removable hardware card (smart or rights card) possessed by a user).

(d) External network source. The rules may be stored on an external server. Rules may be addressed and downloaded by the CMPS if necessary (e.g., either the CMPO and/or other CI and/or the CMPS contains a pointer to certain rules location(s), such as one

Petitioner Apple Inc. - Exhibit 1004, p. 5108

or more URLs). In addition, content providers and/or clearinghouses may broadcast rules designed for general applicability. For example, a content provider might broadcast a set of rules providing a discount to any user participating in a promotional event (e.g., by providing certain user information). Such rules could be received by all connected devices, could be received by certain devices identified as of interest by the content provider (e.g., all recent viewers of a particular program, as identified by exhaust information provided by the CMPS to a clearinghouse and/or all members having certain identity characteristics such as being members of one or more classes) and/or could be posted in central locations.

Example Embodiment

In one embodiment, a set of MPEG-4 Elementary Streams may make up a work. The Elementary Streams may be encrypted and multiplexed together to form an Aggregate Stream. One or more CMPOs may be present in such stream, or may otherwise be associated with the stream. Options are as follows:

1. Content may be streamed or may be received as static data structures.

2. A Work may be made up of a single stream or data structure, or of many separately addressable streams or data structures, each of which may constitute an Object.

3. If a Work is made up of separately addressable streams or data structures, those streams or data structures may be multiplexed together into an Aggregate Stream, or may be received separately.

4. If streams or data structures are multiplexed together into an Aggregate Stream, the streams or data structures may be encrypted prior to such multiplexing. The Aggregate Stream itself may be encrypted, whether or not the underlying streams or data structures are encrypted. The following possibilities therefore exist: (a) individual streams/data structures are unencrypted (in the clear), the Aggregate Stream is unencrypted; (b) individual streams/data structures are unencrypted prior to multiplexing, the Aggregate Stream is encrypted following multiplexing; (c) individual streams/data structures are encrypted prior to multiplexing, the Aggregate Stream is not encrypted following multiplexing; or (d) individual streams/data structures are encrypted prior to multiplexing, the Aggregate Stream is encrypted following multiplexing.

5. A CMPO may be associated with a channel (CCMPO), a work (MCMPO) or an individual Object (CMPO).

6. A CMPO may be received prior to the controlled data, may be received contemporaneously with the data, or may be received after the data (in which event use of the data must wait until the CMPO has been received).

7. A CMPO may be received as part of an Aggregate Stream or separately.

Petitioner Apple Inc. - Exhibit 1004, p. 5109

8. If a CMPO is received as part of the Aggregate Stream, it may be multiplexed together with the individual streams or data structures, or may constitute a separate stream or data structure.

9. If a CMPO is multiplexed within the Aggregate Stream, it may be encrypted or nonencrypted. If encrypted, it may be encrypted prior to multiplexing, and/or encrypted after multiplexing, if the entire Aggregate Stream is encrypted.

10. If a CMPO is received as part of the Aggregate Stream, it may be (a) a part of the stream or data structure which holds the content (e.g., a header); (b) a separate stream or data structure encoded pursuant to the same format as the streams or data structures which hold the content (e.g., an MPEG-4 ES) or (c) a separate stream or data structure encoded under a different format designed for CMPOs.

11. If a CMPO is a part of the stream or data structure which holds the content, it may be (a) a header which is received once and then persistently maintained for control of the content; (b) a header which is received at regular intervals within the stream or data structure; or (c) data distributed throughout the stream or data structure.

These various scenarios give rise to different requirements for demultiplexing and decryption of the CMPOs. FIG. 25 illustrates the following embodiment:

1. Aggregate Stream 2501 is made up of multiplexed ESs (e.g., ES 2502 and 2503). A combination of such ESs makes up a single work. Aggregate Stream 2501 is generated by a cable aggregator and received by a user's set-top box as one of a number of channels.

2. CCMPOs 2504 corresponding to each channel are sent along the cable in Header 2505 at regular intervals (e.g., once per second). When the set-top box is turned on, it polls each channel, and downloads all current CCMPOs. These are stored persistently, and are changed only if a new CCMPO is received which differs from prior CCMPOs.

3. When the user selects a channel, the set-top box addresses the associated CCMPO. The CCMPO may specify, for example, that content in this particular channel may only be accessed by subscribers to the channel. A CMPS within the set-top box accesses a user profile persistently stored in NVRAM and determines that the user is a subscriber. The CMPS deems the CCMPO rule to have been satisfied.

4. The CMPS obtains an identifier for the MCMPO associated with the work (video) currently streaming on the channel and a key for the MCMPO. If works are received serially on the channel (e.g., a television channel in which one work is provided at a time), the received MCMPO identifier may include don't care bits so that it can address any MCMPO currently on the channel.

Petitioner Apple Inc. - Exhibit 1004, p. 5110

5. The CMPS begins demuxing of Aggregate Stream 2501 (this may occur in parallel with the preceding step), and obtains the MCMPO, which is encoded into an ES multiplexed within the Aggregate Stream (e.g., MCMPO 2506). Although each ES within Aggregate Stream 2501 has been encrypted, Aggregate Stream 2501 was not encrypted following multiplexing. This allows the CMPS to demultiplex Aggregate Stream 2501 without decrypting the entire Aggregate Stream.

6. The CMPS identifies the ES which constitutes the MCMPO (e.g., ES 2503). The CMPS downloads one complete instance of MCMPO 2506 into an internal buffer, and uses the key received from CCMPO 2504 to decrypt MCMPO 2506.

7. The CMPS determines which rules are applied by MCMPO 2506. MCMPO 2506 might, for example, include a rule stating that the user can view the associated work with advertisements at a low fee, but must pay a higher fee for viewing the work without advertisements.

8. The CMPS generates an options menu, and displays that menu on the screen for the user. The menu specifies the options, including the cost for each option. Additional options may be specified, including payment types.

9. The user uses a remote control pointing device to choose to view the work at a lower cost but with advertisements. The user specifies that payment can be made from an electronic cash budget stored in the CMPS.

10. The CMPS subtracts the specified amount from the budget persistently stored in NVRAM, and generates and encrypts a message to a server associated with the cable. The message transfers the required budget to the server, either by transferring electronic cash, or by authorizing a financial clearinghouse to transfer the amount from the user's account to the cable provider's. This message may be sent immediately, or may be buffered to be sent later (e.g., when the user connects the device to the Internet). This step may be taken in parallel with decryption of the content.)

11. The CMPS obtains from MCMPO 2506 a set of keys used to decrypt the Elementary Streams associated with the work (e.g., ES 2502). The CMPS also obtains identifiers for the specific ESs to be used. Since the user has indicated that advertisements are to be included, the MCMPO identifies ESs associated with the advertisements, and identifies a Scene Descriptor Graph which includes advertisements. A Scene Descriptor Graph which does not include advertisements is not identified, and is not passed through by the CMPS.

12. The CMPS passes the decrypted ESs to the MPEG-4 buffers. The normal process of MPEG-4 decoding, compositing and rendering then takes place. The Composite

Petitioner Apple Inc. - Exhibit 1004, p. 5111

and Render block outputs Start and Stop events for each object released for viewing. The CMPS monitors this information and compares it to the expected events. In particular, the CMPS confirms that the advertisements have been released for viewing, and that each operation has occupied approximately the expected amount of time.

In another embodiment, a set-top box containing a CMPS (e.g., CMPS 2302 from FIG. 23) may have a cable input (e.g., carrying M4 Bit Streams 2314 and CMPOs 2303). The cable may carry multiple channels, each made up of two sub-channels, with one sub-channel carrying MPEG-4 ESs (e.g., M4 Bit Streams 2314), and the other sub-channel carrying CMPOs (e.g., CMPOs 2303). The sub-channel carrying CMPOs 2303 could be routed directly to CMPS 2302, with the ES channel being routed to a decryption block (operating under control of the CMPS, e.g., CR&D 2315), and then to the MPEG-4 buffers (e.g., buffers associated with Scene Descriptor Graph 2306, AVO Decode 2307 and Object Descriptors 2308). In this case, if the ESs are not encrypted, they proceed unchanged through the decryption block and into the buffers. This may occur, for example, if the ESs are being broadcast for free, with no restrictions, and/or if they are public domain information, and/or they were created prior to inclusion of CMPOs in the MPEG-4 standard.

Such an embodiment might include timing synchronization information in the CMPO sub-channel, so that CMPOs can be synchronized with the associated ESs.

The concept of incorporating two separate streams, one consisting of control information and connected directly to the CMPS, and the other consisting of ESs, may support a high degree of modularization, such that the formats of CMPOs, and particular types of CMPS's, may be changed without alteration to the underlying ES format. For example, it may be possible to change the CMPO format without the necessity for reformatting content ESs. To take another example, it may be possible to upgrade a Commerce Appliance by including a new or different CMPS, without the necessity for any changes to any of the circuitry designed to demultiplex, composite and render the content ESs. A user might obtain a CMPS on a smart card or other removable device, and plug that device into a Commerce Appliance. This could be done to customize a Commerce Appliance for a particular application or for particular content.

CMPS Interface to a CE Device

A CMPS may be designed to present a standardized interface between the general-purpose functionality of a consumer electronics device and any relevant CMPOs and/or other CI and protected content. For example, a CMPS could be designed to accept CI and encrypted ESs, and output decrypted ESs into the device's buffers. In such a case, the

Petitioner Apple Inc. - Exhibit 1004, p. 5112

manufacturer of the device would be able to design the device in compliance with the specification (e.g., MPEG-4), without concern about commerce-related extensions to the standard, which extensions might differ from provider to provider. All such extensions would be handled by the CMPS.

Initialization

1. Initialization of the CMPS.

A CMPS may be used to identify the capabilities of the Commerce Appliance in which a CMPS is installed. A CMPS permanently associated with a particular Commerce Appliance may have such information designed-in when the CMPS is initially installed (e.g., stored in ROM 2406 shown in FIG.24). A CMPS which is removable may be used to run an initialization operation in order to obtain information about the device's capabilities. Such information may be stored in a data structure stored in NVRAM 2425. Alternatively, some or all of such information may be gathered each time the device is turned on, and stored in RAM 2414.

For example, a DVD player may or may not contain a connection to an external server and/or process. A CMPO and/or other CI stored on a DVD (and/or any other format optical disk) inserted into a DVD (or any other format optical disk) player may include rules predicated on the possibility of outputting information to a server (e.g., content is free if user identification information is output), or may require a direct connection in order, for example, to download keys used to decrypt content. In such a case, the CMPS arrangement may determine the hardware functionality which is expected by or required by the CMPO, and compare that to the hardware actually present. If the CMPS determines that the CMPO and/or other CI requires a network connection , and that the DVD player does not include such a connection, the CMPS may take a variety of steps, including: (1) if the network connection is required for some options but not others, causing only those options which are possible to be displayed to the user; (2) informing the user that necessary hardware is missing; or (3) causing a graceful rejection of the disk, including informing the user of the reason for the rejection.

To take another example, a CMPO and/or other CI may include a business model which allows the user to choose among quality levels (or other forms of variations of a given work, for example, longer length and/or greater options), with a higher price being charged if the user selects a higher level of quality (e.g., music may be played at low resolution for free, but requires a payment in order to be played at a higher resolution). In such a case, the Commerce Appliance may not include loudspeakers which are capable of outputting sound at the higher resolution. The CMPS arrangement preferably identifies this situation, and either eliminates the higher resolution output as an option for the user, or

Petitioner Apple Inc. - Exhibit 1004, p. 5113

informs the user that this option costs more but provides no additional benefit given the Commerce Appliance's current functionality or given the Commerce Appliance not being docked in a user arrangement that provides higher quality loudspeakers.

If the Commerce Appliance may be hooked up to external devices (e.g., loudspeakers, display, etc.), the CMPS will require some mechanism for identifying and registering such devices. Each device may be used to make standard ID and capability information available at all times, thereby allowing the CMPS to poll all connected devices at regular intervals, including, for example, authenticating CMPS arrangements within one or more of each such connected devices. Using another approach, all devices could be used to output CMPS identification information upon power-on, with later connected devices being used to output such information upon establishment of the connection. Such identification information may take the form, for example, of authentication information provided under the "five company arrangement", such authentication methods are herein incorporated by reference.

As discussed earlier, a Commerce Appliance may be connected to multiple devices each containing its own CMPS arrangement (e.g., a DVD player may be connected to a digital TV) In such cases, the CMPSs must be able to initiate secure communication (e. g., using a scheme, for example, like the "five company proposal" for IEEE 1394 serial bus) and determine how the CMPSs will interact with respect to content communication between CMPSs and, in certain embodiments, regarding cooperative governance of such content such as describing in the incorporated Shear patent application. In one embodiment, the first CMPS arrangement to receive content might govern the control process by downloading an initial CMPO and/or other CI, and display one or more of the rules to the user, etc. The second CMPS arrangement might recognize that it has no further role to play, either as a result of a communication between the two CMPS arrangements, or as a result of changes to the content stream created by the first CMPS arrangement (which decrypted the content, and may have allowed demuxing, composition and rendering, etc.)

The relationship between upstream and downstream CMPSs arrangements may be complicated if one device handles certain aspects of MPEG-4 rendering, and the other handles other aspects. For example, a DVD player might handle demuxing and buffering, transferring raw ESs to a digital TV, which then handles composition and rendering, as well as display. In such a case, there might be no back-channel from the composition and rendering block to the upstream CMPS arrangement. CMPS arrangements are preferably designed to handle stand-alone cases (a DVD (or any other optical disk) player with a CMPS arrangement attached to a dumb TV with no CMPS), multiple CMPS arrangement

cases in which one CMPS arrangement handles all of the processing (a DVD (or other optical disk) player which handles everything through composition and rendering, with a video stream output to the digital TV (in one non-limiting example, via an IEEE 1349 serial bus) (that output stream would be encrypted as per the "five company proposal" for copy protection using IEEE 1394 serial bus transmission)) and/or shared processing between two or more CMPSs arrangements regarding some, or in certain cases, all, of such processing.

2. Initialization of a particular content stream.

The CMPS may be designed so that it can accept initialization information which initializes the CMPS for a particular content stream or channel. This header, which may be a CMPO and/or other CI, may contain information used by the CMPS to locate and/or interpret a particular content stream as well as CI associated with that stream. This initial header may be received through a sideband channel, or may be received as a CI ES such as a CMPO ES.

In one example, shown in FIG. 26, Header CMPO 2601 may include the following information:

(a) Stream/Object/CMPO ID 2602, which identifies the content streams/objects governed by Header CMPO 2601 and/or identification of CMPOs associated with each such content stream or object.

In one embodiment, Header CMPO 2601 identifies other CMPOs which contain rules and keys associated with particular content streams. In another embodiment, Header CMPO 2601 directly controls all content streams, by incorporating the keys and rules associated with such streams. In the latter case, no other CMPOs may be used.

In one embodiment, Header CMPO 2601 may be one or more CMPOs, CCMPOs, MCMPOs, and/or other CI.

(b) One or CMPO Keys 2603 for decrypting each identified CMPO.

(c) Work-Level Control 2604, consisting of basic control information associated with the work as a whole, and therefore potentially applicable to all of the content streams which make up the work. This basic control information may include rules governing the work as a whole, including options to be presented to the user.

(d) In one embodiment of this embodiment, a header CMPO may be updatable to contain User/Site Information 2605 regarding a particular user or site currently authorized to use certain content, as well as one or more rule sets under which the user has gained such authorization. A header CMPO associated with a work currently being viewed may be stored in RAM or NVRAM. This may include updated information. In one

embodiment, the CMPO may also store header CMPOs for certain works viewed in the past. In one embodiment, header CMPOs may be stored in non-secure memory, with information sufficient to identify and authenticate that each header CMPO had not been changed.

In one such header CMPO embodiment of this embodiment, the header CMPO operates as follows:

(a) The header CMPO is received by a CMPS arrangement. In the case of previously unreceived content which has now become available, the header CMPO may be received at an input port. In the case of content which is already available, but is not currently being used (e.g., a set-top box with 500 channels, of which either 0 or 1 are being displayed at any given time), CCMPOs for each channel may be buffered by the CMPS arrangement for possible use if the user invokes particular content (e.g., switches to a particular channel).

In either case, the header CMPO must include information which allows a CMPS arrangement to identify it as a header CMPO.

(b) The CMPS arrangement obtains business-model information held in the clear in the header CMPO. Business-model information may include, for example, a statement that content can be viewed for free if advertisements are included, or if the user authorizes Nielson-type information, user and/or audience measurement information, for example, content may be output to a server or otherwise copied once, but only at a price.

(c) The CMPS arrangement either accepts the business model, if the user has authorized it to accept certain types of models (e.g., the user has programmed the CMPS arrangement to always accept play with advertisements for free), rejects the business model, if the user has instructed that the particular model always be rejected, or displays the business model to the user (e.g., by presenting options on the screen).

(d) If a business model has been accepted, the CMPS arrangement then decrypts the remainder of the header CMPO. If the Commerce Appliance contains a live output connection to an external server (e.g., Internet connection, back-channel on a set-top box, etc.), and if latency problems are handled, decryption of these keys can be handled by communicating with the external server, each side authenticating the other, establishment of a secure channel, and receipt of a key from the server. If the Commerce Appliance is not at least occasionally connected to an external server, decryption may have to be based on one or more keys securely stored in the Commerce Appliance.

(e) Once a header CMPO has been decrypted, the CMPS arrangement acquires information used to identify and locate the streams containing the content, and

keys which are used to decrypt either the CMPOs associated with the content, or to directly decrypt the content itself.

(f) In one embodiment of this header embodiment, the header CMPO may contain a data structure for the storage of information added by the CMPS arrangement. Such information may include the following:

(1) Identification of user and/or Commerce Appliance and/or CMPS arrangement. In this embodiment, such information may be stored in a header CMPO in order to provide an audit trail in the event the work (including the header CMPO) is transferred (this only works if the header CMPO is transferred in a writable form). Such information may be used to allow a user to transfer the work to other Commerce Appliances owned by the user without the payment of additional cost, if such transfers are allowed by rule information associated with the header CMPO. For example, a user may have a subscription to a particular cable service, paid for in advance by the user. When a CMPS arrangement downloads a header CMPO from that cable service, the CMPS arrangement may store the user's identification in the header CMPO. The CMPS arrangement may then require that the updated header CMPO be included if the content is copied or transferred. The header CMPO could include a rule stating that, once the user information has been filled in, the associated content can only be viewed by that user, and/or by Commerce Appliances associated with that user. This would allow the user to make multiple copies of the work, and to display the work on multiple Commerce Appliances, but those copies could not be displayed or used by non-authorized users and/or on non-authorized Commerce Appliances. The header CMPO might also include a rule stating that the user information can only be changed by an authorized user (e.g., if user 1 transfers the work to user 2, user 2's CMPS arrangement can update the user information in the header CMPO, thereby allowing user 2 to view the work, but only if user 2 is also a subscriber to the cable channel).

(2) Identification of particular rules options governing use. Rule sets included in header CMPOs may include options. In certain cases, exercise of a particular option might preclude later exercise of a different option. For example, a user might be given the choice to view an unchanged work for one price, or to change a work and view the changed work for a higher price. Once the user decides to change the work and view the changed work, this choice is preferably stored in the header CMPO, since the option of viewing the original unchanged work at the lower price is no longer available. The user might have further acquired the right, or may now be presented with the option for the right, to further distribute the changed work at a mark-up in cost resulting in third party

derived revenue and usage information flowing to both the user and the original work stakeholder(s).

(3) Historical usage information. The header CMPO may include information relating to the number and types of usages. For example, if the underlying work is copied, the header CMPO may be updated to reflect the fact that a copy has been made, since a rule associated with the work might allow only a single copy (e.g., for backup and/or timeshifting purposes). To take another example, a user might obtain the right to view a work one time, or for a certain number of times. The header CMPO would then be updated to reflect each such use.

Usage information may be used to determine if additional uses are authorized by rules associated with the header CMPO. Such information may also be used for audit purposes. Such information may also be provided as usage information exhaust, reported to an external server. For example, a rule may specify that a work may be viewed for free, but only if historical usage information is downloaded to a server.

Content Management Protection Objects (CMPO)

The Content Management and Protection Object ("CMPO") is a data structure which includes information used by the CMPS to govern use of certain content. A CMPO may be formatted as a data structure specified by a particular standard (e.g., an MPEG-4 ES), or may be formatted as a data structure not defined by the standard. If the CMPO is formatted as a data structure specified by the standard, it may be received in the channel utilized by the standard (e.g., as part of a composite MPEG-4 stream) or may be received through some other, side-band method. If the CMPO is formatted as a data structure not specified by the relevant standard, it is provided and decoded using some side-band method, which may include receipt through the same port as formatted content and/or may include receipt through a separate port.

Content may be controlled at virtually any level of granularity. Three exemplary levels will be discussed herein: "channel," "work," and "object."

A "channel" represents an aggregation of works. The works may be available for selection by the user (e.g., a web site, or a video library) or may be received serially (e.g., a cable television channel).

A "work" represents a single audio-visual, textual or other work, intended to be consumed (viewed, read, etc.) by a user as an integrated whole. A work may, for example, be a movie, a song, a magazine article, a multimedia product such, for example, as sophisticated videogame. A work may incorporate other works, as, for example, in a multimedia work which incorporates songs, video, text, etc. In such a case, rights may be

associated

An "object" represents a separately addressable portion of a work. An object may be, for example, an individual MPEG-4 AVO, a scene descriptor graph, an object descriptor, the soundtrack for a movie, a weapon in a videogame, or any other logically definable portion.

Content may be controlled at any of these levels (as well as intermediate levels not discussed herein). The preferred embodiment mechanism for such control is a CMPO or CMPO arrangement (which comprises one or more CMPOs, and if plural, then plural, cooperating CMPOs). CMPOs and CMPO arrangements may be organized hierarchically, with a Channel CMPO arrangement imposing rules applicable to all contained works, a MCMPO or an SGCMPO imposing rules applicable to all objects within a work, and a CMPO arrangement imposing rules applicable to a particular object.

In one embodiment, illustrated in FIG. 27, a CMPS may download CCMPO 2701. CCMPO 2701 may include one or more Rules 2702 applicable to all content in the channel, as well as one or more Keys 2703 used for decryption of one or more MCMPOs and/or SGCMPOs. MCMPO 2704 may include Rules 2705 applicable to a single work and/or works, one or more classes and/or more users and/or user classes, and may also include Keys 2706 used to decrypt CMPOs. CMPO 2707 may include Rules 2708 applicable to an individual object, as well as Key 2709 used to decrypt the object.

As long as all objects are subject to control at some level, there is no requirement that each object be individually controlled. For example, CCMPO 2701 could specify a single rule for viewing content contained in its channel (e.g., content can only be viewed by a subscriber, who is then might be free to redistribute the content with no further obligation to the content provider). In such a case, rules would not necessarily be used for MCMPOs (e.g. Rules 2705), SGCMPOs, or CMPOs (e.g., Rules 2708). In one embodiment, MCMPOs, SGCMPOs, and CMPOs could be dispensed with, and CCMPO 2701 could include all keys used to decrypt all content, or could specify a location where such keys could be located. In another embodiment, CCMPO 2701 would supply Key 2703 used to decrypt MCMPO 2704. MCMPO 2704 might include keys used to decrypt CMPOs (e.g., Keys 2706), but might include no additional Rules 2705. CMPO 2707 might include Key 2709 used to decrypt an object, but might include no additional Rules 2708. In certain embodiments, there may be no SGCMPOs.

A CMPO may be contained within a content data structure specified by a relevant standard (e.g., the CMPO may be part of a header in an MPEG-4 ES.) A CMPO may be contained within its own, dedicated data structure specified by a relevant standard (e.g., a

CMPO ES). A CMPO may be contained within a data structure not specified by any content standard (e.g., a CMPO contained within a DigiBox).

A CCMPO may include the following elements:

(a) ID 2710. This may take the following form: <channel ID>< CMPO type><CMPO ID><version number>. In the case of hierarchical CMPO organization (e.g., CCMPOs controlling MCMPOs controlling CMPOs), CMPO ID 2711 can include one field for each level of the hierarchy, thereby allowing CMPO ID 2711 to specify the location of any particular CMPO in the organization. ID 2710 for a CCMPO may, for example, be 123-000-000. ID 2712 for a MCMPO of a work within that channel may, for example, be 123-456-000, thereby allowing the specification of 1,000 MCMPOs as controlled by the CCMPO identified as "123." CMPO ID 2711 for a CMPO associated with an object within the particular work may, for example, be 123-456-789, thereby allowing the specification of 1,000 CMPOs as associated with each MCMPO.

This method of specifying CMPO IDs thereby conveys the exact location of any CMPO within a hierarchy of CMPOs. For cases in which higher levels of the hierarchy do not exist (e.g., a MCMPO with no associated CCMPO), the digits associated with that level of the hierarchy may be specified as zeroes.

(b) Rules 2702 applicable to all content in the channel. These may be self-contained rules, or may be pointers to rules obtainable elsewhere. Rules are optional at this level.

(c) Information 2713 designed for display in the event the user is unable to comply with the rules (e.g., an advertisement screen informing the user that a subscription is available at a certain cost, and including a list of content available on the channel).

(d) Keys 2703 for the decryption of each MCMPO controlled by this CCMPO. In one embodiment, the CCMPO includes one or more keys which decrypt all MCMPOs. In an alternate embodiment, the CCMPO includes one or more specific keys for each MCMPO.

(e) A specification of a CMPS Type (2714), or of hardware/software necessary or desirable to use the content associated with this channel.

The contents of a MCMPO may be similar to those of a CCMPO, except that the MCMPO may include rules applicable to a single work, and may identify CMPOs associated with each object.

The contents of each CMPO may be similar to those of the MCMPO, except that the CMPO may include rules and keys applicable to a single object.

The contents of an SGCMPO may be similar to those of the CCMPO, except that the MCMPO may include rules applicable to only certain one or more classes of rights, certain one or more classes of works, and/or to one or more certain classes of users and/or user arrangements (e.g. CMPO arrangements and/or their devices).

In another embodiment, shown in FIG. 28, CMPO Data Structure 2801 may be defined as follows:

CMPO Data Structure 2801 is made up of elements. Each element includes a self-contained item of information. The CMPS parses CMPO Data Structure, one element at a time.

Type Element 2802 identifies the data structure as a CMPO, thereby allowing the CMPS to distinguish it from a content ES. In an exemplary embodiment, this element may include 4 bits, each of which may be set to "1" to indicate that the data structure is a CMPO.

The second element is CMPO Identifier 2803, which is used to identify this particular CMPO and to convey whether the CMPO is part of a hierarchical organization of CMPOs and, if so, where this CMPO fits into that organization.

CMPO Identifier 2803 is divided into four sub-elements, each of three bits. These are shown as sub-elements A, B, C and D. The first sub-element (2803 A) identifies the CMPO type, and indicates whether the CMPO is governed or controlled by any other CMPO:

100: this is a top-level CMPO (associated with a channel or an aggregation of works) and is not controlled by any other CMPO.

010: this is a mid-level CMPO (associated with a particular work) and is not controlled by any other CMPO.

110: this is a mid-level CMPO, and is controlled by a top-level CMPO.

001: this is a low-level CMPO (associated with an object within a work) and is not controlled by any other CMPO. This case will be rare, since a low-level CMPO will ordinarily be controlled by at least one higher-level CMPO.

011: this is a low-level CMPO, and is controlled by a mid-level CMPO, but not by a top-level CMPO.

111: this is a low-level CMPO, and is controlled by a top-level CMPO and by a mid-level CMPO.

The second sub-element of CMPO ID 2803 (sub-element B) identifies a top-level CMPO. In the case of a top-level CMPO, this identifier is assigned by the creator of the CMPO. In the case of a mid-level or low-level CMPO which is controlled by a top-level

CMPO, this sub-element contains the identification of the top-level CMPO which performs such control. In the case of a mid-level or low-level CMPO which is not controlled by a top-level CMPO, this sub-element contains zeroes.

The third sub-element of CMPO ID 2803 (sub-element C) identifies a mid-level CMPO. In the case of a top-level CMPO, this sub-element contains zeroes. In the case of a mid-level CMPO, this sub-element contains the identification of the particular CMPO. In the case of a low-level CMPO which is controlled by a mid-level CMPO, this sub-element contains the identification of the mid-level CMPO which performs such control. In the case of a low-level CMPO which is not controlled by a mid-level CMPO, this sub-element contains zeroes.

The fourth sub-element of CMPO ID 2803 (sub-element D) identifies a low-level CMPO. In the case of a top-level or mid-level CMPO, this sub-element contains zeroes. In the case of a low-level CMPO, this sub-element contains the identification of the particular CMPO.

Following the identifier element is Size Element 2804 indicating the size of the CMPO data structure. This element contains the number of elements (or bytes) to the final element in the data structure. This element may be rewritten if alterations are made to the CMPO. The CMPS may use this size information to determine whether the element has been altered without permission, since such an alteration might result in a different size. For such purposes, the CMPS may store the information contained in this element in a protected database. This information can also be used to establish that the entire CMPO has been received and is available, prior to any attempt to proceed with processing.

Following Size Element 2804 are one or more Ownership/Control Elements containing ownership and chain of control information (e.g., Ownership/Control Elements 2805, 2806 and 2807). In the first such element (2805), the creator of the CMPO may include a specific identifier associated with that creator. Additional participants may also be identified in following elements (e.g., 2806, 2807). For example, Element 2805 could identify the creator of the CMPO, Element 2806 could identify the publisher of the associated work and Element 2807 could identify the author of the work.

A specific End Element 2808 sequence (e.g., 0000) indicates the end of the chain of ownership elements. If this sequence is encountered in the first element, this indicates that no chain of ownership information is present.

Chain of ownership information can be added, if rules associated with CMPO 2801 permit such additions. If, for example, a user purchases the work associated with CMPO 2801, the user's identification may be added as a new element in the chain of ownership

elements (e.g., a new element following 2807, but before 2808). This may be done at the point of purchase, or may be accomplished by the CMPS once CMPO 2801 is encountered and the CMPS determines that the user has purchased the associated work. In such a case, the CMPS may obtain the user identifier from a data structure stored by the CMPS in NVRAM.

Following the ownership element chain are one or more Handling Elements (e.g., 2809, 2810) indicating chain of handling. These elements may contain the identification of any CMPS which has downloaded and decoded CMPO 2801, and/or may contain the identification of any user associated with any such CMPS. Such information may be used for audit purposes, to allow a trail of handling in the event a work is determined to have been circulated improperly. Such information may also be reported as exhaust to a clearinghouse or central server. Chain of handling information preferably remains persistent until reported. If the number of elements required for such information exceeds a specified amount (e.g., twenty separate user identifiers), a CMPS may refuse to allow any further processing of CMPO 2801 or the associated work until the CMPS has been connected to an external server and has reported the chain of handling information.

The last element in the chain of handling elements (e.g., 2811) indicates the end of this group of elements. The contents of this element may, for example, be all zeroes.

Following the chain of handling elements may be one or more Certificate Elements (e.g., 2812, 2813) containing or pointing to a digital certificate associated with this CMPO. Such a digital certificate may be used by the CMPS to authenticate the CMPO. The final element in the digital certificate chain is all zeroes (2814). If no digital certificate is present, a single element of all zeroes exists in this location.

Following the Certificate Elements may be a set of Governed Object Elements (e.g., 2815, 2816, 2817, 2818) specifying one or more content objects and/or CMPOs which may be governed by or associated with CMPO 2801. Each such governed object or CMPO is identified by a specific identifier and/or by a location where such object or CMPO may be found (e.g., these may be stored in locations 2815 and 2817). Following each such identifier may be one or more keys used to decrypt such CMPO or object (e.g., stored in locations 2816 and 2818). The set of identifiers/keys ends with a termination element made up of all zeroes (2819).

Following the set of elements specifying identifiers and/or keys may be a set of Rules Elements (e.g., 2820, 2821, 2822) specifying rules/controls and conditions associated with use of the content objects and/or CMPOs identified in the Governed Objects chain (e.g., locations 2815 and 2817). Exemplary rules are described below. Elements may

contain explicit rules or may contain pointers to rules stored elsewhere. Conditions may include particular hardware resources necessary to use associated content objects or to satisfy certain rules, or particular types of CMPS's which are necessary or preferred for use of the associated content objects.

Following the rules/controls and conditions elements may be a set of Information Elements 2823 containing information specified by the creator of the CMPO. Among other contents, such information may include content, or pointers to content, programming, or pointers to programming.

The CMPO ends with Final Termination Element 2824.

In one embodiment, the rules contained in Rules Elements 2820-2822 of CMPO 2801 may include, for example, the following operations:

(1) Play. This operation allows the user to play the content (though not to copy it) without restriction.

(2) Navigate. This allows the user to perform certain types of navigation functions, including fast forward/rewind, stop and search. Search may be indexed or unindexed.

(3) Copy. Copy may be allowed once (e.g., time-shifting, archiving), may be allowed for a specified number of times and/or may be allowed for limited period of time, or may be allowed for an unlimited period of time, so long as other rules, including relevant budgets, are not violated or exceeded. A CMPS arrangement may be designed so that a Copy operation may cause an update to an associated CMPO (e.g., including an indication that the associated content has been copied, identifying the date of copying and the site responsible for making the copy), without causing any change to any applicable content object, and in particular without requiring that associated content objects be demuxed, decrypted or decompressed. In the case of MPEG-4, for example, this may require the following multi-stage demux process:

(i) the CMPS arrangement receives a Copy instruction from the user, or from a header CMPO.

(ii) CMPO ESs associated with the MPEG-4 stream which is to be copied are separated from the content stream in a first demux stage.

(iii) CMPOs are decrypted and updated by the CMPS arrangement. The CMPOs are then remuxed with the content ESs (which have never been demuxed from each other), and the entire stream is routed to the output port without further alteration.

This process allows a copy operation to take place without requiring that the content streams be demuxed and decrypted. It requires that the CMPS arrangement include

two outputs: one output connected to the digital output port (e.g., FIG. 23 line 2316, connecting to Digital Output Port 2317), and one output connected to the MPEG-4 buffers (e.g., FIG. 23, lines 2310, 2311, 2312), with a switch designed to send content to one output or the other (or to both, if content is to be viewed and copied simultaneously) (e.g., Switch 2319). Switch 2319 can be the only path to Digital Output Port 2317, thereby allowing CMPS 2302 to exercise direct control over that port, and to ensure that content is never sent to that port unless authorized by a control. If Digital Output Port 2317 is also the connector to a digital display device, CMPS 2302 will also have to authorize content to be sent to that port even if no copy operation has been authorized.

In one example embodiment, the receiving device receiving the information through Digital Output Port 2317 may have to authenticate with the sending device (e.g., CMPS 2302). Authentication may be for any characteristic of the device and/or one or more CMPSs used in conjunction with that device. Thus, for example, a sending appliance may not transmit content to a storage device lacking a compatible CMPS.

In another non-limiting example, CMPS 2302 can incorporate session encryption functionality (e.g., the "five company arrangement") which establishes a secure channel from a sending interface to one or more external device interfaces (e.g., a digital monitor), and provided that the receiving interface has authenticated with the sending interface, encrypts the content so that it can only be decrypted by one or more authenticated 1394 device interfaces. In that case, CMPS 2302 would check for a suitable IEEE 1394 serial bus interface , and would allow content to flow to Digital Output Port 2317 only if (a) an authorized Play operation has been invoked, a secure channel has been established with the device and the content has been session-encrypted, or (b) an authorized Copy or Retransmit operation has been invoked, and the content has been treated as per the above description (i.e., the CMPO has been demuxed, changed and remuxed, the content has never been decrypted or demuxed).

This is only possible if CMPOs are separately identifiable at an early demux stage, which most likely requires that they be stored in separate CMPO ESs. If the CMPOs are stored as headers in content ESs, it may be impossible to identify the CMPOs prior to a full demux and decrypt operation on the entirety of the stream.

(4) Change. The user may be authorized to change the content.

(5) Delete. This command allows the user to delete content which is stored in the memory of the Consumer Appliance. This operation operates on the entire work. If the user wishes to delete a portion of a work, the Change operation must be used.

(6) Transfer. A user may be authorized to transfer a work to a third party.

This differs from the Copy operation in that the user does not retain the content or any rights to the content. The Transfer operation may be carried out by combining a Copy operation and a Delete operation. Transfer may require alteration of the header CMPO associated with the work (e.g., adding or altering an Ownership/Control Element, such as Elements 2805-2807 of FIG. 28), so as to associate rights to the work with the third party.

These basic operations may be subject to modifications, which may include:

i. Payment. Operations may be conditioned on some type of user payment. Payment can take the form of cash payment to a provider (e.g., credit card, subtraction from a budget), or sending specified information to an external site (e.g., Nielson-type information).

ii. Quality of Service. Operations may specify particular quality of service parameters (e.g., by specifying a requested QoS in MPEG-4), including: requested level of decompression, requested/required types of display, rendering devices (e.g., higher quality loudspeakers, a particular type of game controller).

iii. Time. Operations may be conditioned such that the operation is only allowed after a particular time, or such that the price for the operation is tied to the time (e.g., real-time information at a price, delayed information at a lower price or free, e.g., allowing controlled copies but only after a particular date).

iv. Display of particular types of content. Operations may be conditioned on the user authorizing display of certain content (e.g., the play operation may be free if the user agrees to allow advertisements to be displayed).

In all of these cases, a rule may be modified by one or more other rules. A rule may specify that it can be modified by other rules or may specify that it is unmodifiable. If a rule is modifiable, it may be modified by rules sent from other sources. Those rules may be received separately by the user or may be aggregated and received together by the user.

Data types which may be used in an exemplary MPEG-4 embodiment may include the following:

a. CMP Data Stream.

The CMP-ds is a new elementary stream type that has all of the properties of an elementary stream including its own CMPO and a reference in the object descriptors. Each CMP-ds stream has a series of one or more *CMP Messages*. A CMP_Message has four parts:

1. **Count**: [1...*n*] CMPS types supported by this IP ES. Multiple CMPS systems may be supported, each identified by a unique *type*. (There may have

to be a central registry of types.)

2. **CMPS_type_identifiers:** [1...*n*] identifiers, each with an offset in the stream and a length. The offset points to the byte in the CMPO where the data for that CMPS type is found. The length is the length in bytes of this data.

3. **Data segments**: One segment for each of the *n* CMPS types encoded in a format that is proprietary to the CMPS supplier.

4. **CMP_Message_URL:** That references another CMP_Message. (This is in keeping with the standard of using URLs to point to streams.)

b. CMPO.

The CMPO is a data structure used to attach detailed CMP control to individual elementary streams. Each **CMPO** contains:

1. **CMPO_ID:** An identifier for the content under control. This identifier must *uniquely* identify an elementary stream.

2. **CMPO_count:** [1...*n*] CMPS types supported by this **CMPO**.

3. **CMPS_type_identifiers:** [1...*n*] identifiers, each with an offset in the stream and a length. The offset points to the byte in the CMPO where the data for that CMPS type is found. The length is the length in bytes of this data.

4. **Data segments:** *n* data segments. Each data segment is in a format that is proprietary to the CMPS supplier.

5. **CMPO_URL:** An optional URL that references an additional CMPO that adds information to the information in this CMPO. (This is a way of dynamically adding support for new CMPSs.)

c. Feedback Event

The feedback events come in two forms: start and end. Each feedback event contains three pieces of information:

1. **Elementary_stream_ID**

2. **Time:** in presentation time

3. **Object_instance_number**

User Interface.

Commerce Appliance 2301 may include User Interface 2304 designed to convey control-related information to the user and to receive commands and information from the user. This interface may include special purpose displays (e.g., a light which comes on if a current action requires payment), special purpose buttons (e.g., a button which accepts the payment or other terms required for display of content), and/or visual information presented on screen.

Example of Operation in an MPEG-4 Context

1. User selects a particular work or channel. The user may, for example, use a remote control device to tune a digital TV to a particular channel.

2. Selection of the channel is communicated to a CMPS arrangement, which uses the information to either download a CCMPO or to identify a previously downloaded CCMPO (e.g., if the CMPS arrangement is contained in a set-top box, the set-top box may automatically download CCMPOs for every channel potentially reachable by the box).

3. The CMPS arrangement uses the CCMPO to identify rules associated with all content found on the channel. For example, the CCMPO may specify that content may only be viewed by subscribers, and may specify that, if the user is not a subscriber, an advertisement screen should be put up inviting the user to subscribe.

4. Once rules specified by the CCMPO have been satisfied, the CCMPO specifies the location of a MCMPO associated with a particular work which is available on the channel. The channel CMPO may also supply one or more keys used for decryption of the MCMPO.

5. The CMPS arrangement downloads the MCMPO. In the case of an MPEG-4 embodiment, the MCMPO may be an Elementary Stream. This Elementary Stream must be identifiable at a relatively early stage in the MPEG-4 decoding process.

6. The CMPS arrangement decrypts the MCMPO, and determines the rules used to access and use the content. The CMPS arrangement presents the user with a set of options, including the ability to view for free with advertisements, or to view for a price without advertisements.

7. The user selects view for free with advertisements, e.g., by highlighting and selecting an option on the screen using a remote control device.

8. The CMPS arrangement acquires one or more keys from the MCMPO and uses those keys to decrypt the ESs associated with the video. The CMPS arrangement identifies two possible scene descriptor graphs, one with and one without advertisements. The CMPS arrangement passes the scene descriptor graph with advertisements through, and blocks the other scene descriptor graph.

9. The CMPS arrangement monitors the composite and render block, and checks to determine that the advertisement AVOs have actually been released for viewing. If the CMPS arrangement determines that those AVOs have not been released for viewing, it puts up an error or warning message, and terminates further decryption.

CMPS Rights Management In Provider And Distribution Chains

In addition to consumer arrangements, in other embodiments one or more CMPSs

may be used in creating, capturing, modifying, augmenting, animating, editing, excerpting, extracting, embedding, enhancing, correcting, fingerprinting, watermarking, and/or rendering digital information to associate rules with digital information and to enforce those rules throughout creation, production, distribution, display and/or performance processes.

In one non-limiting example, a CMPS, a non-exhaustive example of which may include a least a secure portion of a VDE node as described in the aforementioned Ginter et al., patent specification, is incorporated in video and digital cameras, audio microphones, recording, playback, editing, and/or noise reduction devices and/or any other digital device. Images, video, and/or audio, or any other relevant digital information may be captured, recorded, and persistently protected using at least one CMPS and/or at least one CMPO. CMPSs may interact with compression/decompression, encryption/decryption, DSP, digital to analog, analog to digital, and communications hardware and/or software components of these devices as well.

In another non-exhaustive example, computer animation, special effects, digital editing, color correcting, noise reduction, and any other applications that create and/or use digital information may protect and/or manage rights associated with digital information using at least one CMPS and/or at least one CMPO.

Another example includes the use of CMPSs and/or CMPOs to manage digital assets in at least one digital library, asset store, film and/or audio libraries, digital vaults, and/or any other digital content storage and management means.

In accordance with the present applications, CMPSs and/or CMPOs may be used to manage rights in conjunction with the public display and/or performance of digital works. In one non-exhaustive example, flat panel screens, displays, monitors, TV projectors, LCD projectors, and/or any other means of displaying digital information, may incorporate at least one hardware and/or software CMPS instance that controls the use of digital works. A CMPS may allow use only in conjunction with one or more digital credentials, one example of which is a digital certificate, that warrant that use of the digital information will occur in a setting, location, and/or other context for public display and/or performance. Non-limiting examples of said contexts include theaters, bars, clubs, electronic billboards, electronic displays in public areas, or TVs in airplanes, ships, trains and/or other public conveyances. These credentials may be issued by trusted third parties such as certifying authorities, non-exhaustive examples of which are disclosed in the aforementioned Ginter '712 patent application.

Additional MPEG-4 Embodiment Information

This work is based on the MPEG-4 description in the version 1 Systems Committee Draft (CD), currently the most complete description of the evolving MPEG-4 standard.

This section presents the structural modifications to the MPEG-4 player architecture and discusses the data lines and the concomitant functional changes. Figure 23 shows the functional components of the original MPEG-4 player. Content arrives at Player 2301 packaged into a serial stream (e.g., MPEG-4 Bit Stream 2314). It is demultiplexed via a sequence of three demultiplexing stages (e.g., Demux 2305) into elementary streams. There are three principle types of elementary streams: AV Objects (AVO), Scene Descriptor Graph (SDG), and Object Descriptor (OD). These streams are fed into respective processing elements (e.g., AVO Decode 2307, Scene Descriptor Graph 2306, Object Descriptors 2308). The AVOs are the multimedia content streams such as audio, video, synthetic graphics and so on. They are processed by the player's compression/coding subsystems. The scene descriptor graph stream is used to build the scene descriptor graph. This tells Composite and Render 2309 how to construct the scene and can be thought of as the "script." The object descriptors contain description information about the AVOs and the SD-graph updates.

To accommodate a CMPS (e.g., CMPS 2302) and to protect content effectively, the player structure must be modified in several ways:

- Certain data paths must be rerouted to and from the CMPS
- Certain buffers in the SDG, AVO decode and Object descriptor modules must be secured
- Feedback paths from the user and the composite and render units to the CMPS must be added

In order for CMPS 2302 to communicate with the MPEG-4 unit, and for it to effectively manage content we must specify the CMPO structure and association protocols and we must define the communication protocols over the feedback systems (from the compositor and the user.)

The structural modifications to the player are shown in Figure 23. The principal changes are:

- All elementary streams are now routed through CMPS 2302.
- Direct communication path between Demux 2305 and CMPS 2302.
- A required "Content Release and Decrypt" Module 2315 in CMPS 2302.

- The addition of a feedback loop (e.g., Line 2313) from Composite and Render 2309 to CMPS 2302.
- Bi-directional user interaction directly with the CMPS 2302, through Line 2316.

Furthermore, for M4v2P, CMP-objects are preferably associated with all elementary streams. Elementary streams that the author chooses not to protect are still marked by an "unprotected content" CMPO. The CMPOs are the primary means of attaching rules information to the content. Content here not only refers to AVOs, but also to the scene descriptor graph. Scene Descriptor Graph may have great value and will thus need to be protected and managed by CMPS 2302.

The direct path from Demux 2305 to CMPS 2302 is used to pass a CMPS specific header, that potentially contains business model information, that communicates business model information at the beginning of user session. This header can be used to initiate user identification and authentication, communicate rules and consequences, and initiate up-front interaction with the rules (selection of quality-of-service (QoS), billing, etc.) The user's communication with CMPS 2302 is conducted through a *non-standardized* channel (e.g., Line 2316). The CMPS designer may provide an independent API for framing these interactions.

Feedback Path 2313 from Composite and Render block 2309 serves an important purpose. The path is used to cross check that the system actually presented the user with a given scene. Elementary streams that are processed by their respective modules may not necessarily be presented to the user. Furthermore, there are several fraud scenarios wherein an attacker could pay once and view multiple times. The feedback path here allows CMPS 2302 to cross check the rendering and thereby perform a more accurate accounting. This feedback is implemented by forcing the Composite and Render block 2309 to issue a *start event* that signals the initiation of a given object's rendering that is complemented by a *stop event* upon termination. The feedback signaling process may be made optional by providing a CMP-notification flag that may be toggled to indicate whether or not CMPS 2302 should be notified. All CMPOs would be required to carry this flag.

The final modification to the structure is to require that the clear text buffers in the AVO, SDG and Object Descriptor processors and in the Composite-and-Render block be secured. This is to prevent a pirate from stealing content in these buffers. As a practical matter, this may be difficult, since tampering with these structures may well destroy synchronization of the streams. However, a higher state of security would come from placing these buffers into a protected processing environment.

CMPS 2302 *governs* the functioning of Player 2301, consistent with the following:

- Communication mechanism between CMPS 2302 and the MPEG-4 player (via CMPOs)
- A content release and decryption subsystem
- Version authentication subsystem
- Sufficient performance so as not to interfere with the stream processing in the MPEG-4 components

CMPS 2302 may have a bi-directional side-channel that is external to the MPEG-4 player that may also be used for the exchange of CMP information. Furthermore, the CMPS designer may choose to provide a user interface API that provides the user with the ability to communicate with the content and rights management side of the stream management (e.g., through Line 2316).

Encrypted content is decrypted and released by CMPS 2302 as a function of the rules associated with the protected content and the results of user interaction with CMPS 2302. Unencrypted content is passed through CMPS 2302 and is governed by associated rules and user interaction with CMPS 2302. As a consequence of these rules and user interaction, CMPS 2302 may need to transact with the SDG and AVO coding modules (e.g., 2310, 2311) to change scene structure and/or the QoS grade.

Ultimately, the CMPS designer may choose to have CMPS 2302 generate audit trail information that may be sent to a clearinghouse authority via CMPS Side Channel Port 2318 or as encrypted content that is packaged in the MPEG-4 bit stream.

The MPEG-4 v1 Systems CD uses the term "object" loosely. In this document, "object" is used to specifically mean a data structure that flows from one or more of the data paths in Figure 23.

Using multiple SD-graph update streams, each with its own CMPO, allows an author to apply arbitrarily specific controls to the SD-graph. For example, each node in the SD-graph can be created or modified by a separate SD-graph update stream. Each of these streams will have a distinct CMPO and ID. Thus, the CMPS can release and decrypt the creation and modification of each node and receive feedback information for each node individually. The practical implications for controlling release and implementing consequences should be comparable to having a CMPO on each node of the SD-graph, without the costs of having a CMPO on each SD-graph node.

Principles consistent with the present invention may be illustrated using the following examples:

In the first example, there is a bilingual video with either an English or French soundtrack. The user can choose during playback to hear either the English or French. The

basic presentation costs $1. If the French soundtrack is presented there is a $0.50 surcharge. If the user switches back and forth between French and English, during a single viewing of the presentation, the $0.50 surcharge will occur only once.

In this example, there will be four elementary streams:

The Scene Description Graph Update stream will have a CMPO. The CMPO will imply a $1.00 fee associated with the use of the content. The scene description graph displays the video, English audio and puts up a button that allows the user to switch to French. If the user clicks that button, the English stops, the French picks up from that point and the button changes to a switch-to-English button. (Optionally, there may be a little dialog at the beginning to allow the user to select the initial language. This is all easy to do in the SD graph.)

The Video Stream with the CMPO will say that it can only be released if the scene description graph update stream above is released.

The English Audio Stream will be similar to the Video stream.

The French Audio Stream will be similar to the Video stream but there is a $.50 charge it if is seen in the feedback channel. (The CMPS must to not count twice if the user switches between the two in a single play of the presentation.)

An important requirement is that the ID for the SD-graph update stream appears in the feedback path (e.g., Feedback Path 2313). This is so CMPS 2302 knows when the presentation stops and ends so that CMPS 2302 can correctly bill for the French audio.

The rules governing the release of the video and audio streams may include some variations. The rules for these streams, for example, may state something like "if you don't see the id for the scene description graph update stream X in the feedback channel, halt release of this stream." If the main presentation is not on the display, then the video should not be. This ties the video to this one presentation. Using the video in some other presentation would require access to the original video, not just this protected version of it.

In a second example, an author wants to have a presentation with a free attract sequence or "trailer". If the user clicks the correct button the system moves into the for-fee presentation, which is organized as a set of "acts".

Multiple SD-graph update streams may update a scene description graph. Multiple SD-graph update streams may be open in parallel. The time stamps on the ALUs in the streams are used to synchronize and coordinate.

The trailer and each act are represented by a separate SD-graph update stream with a separate CMPO. There is likely an additional SD-graph update stream that creates a simple

root node that is invisible and silent. This node brings in the other components of the presentation as needed.

The foregoing description of implementations of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practicing of the invention. For example, the described implementation includes software but the present invention may be implemented as a combination of hardware and software or in hardware alone. The invention may be implemented with both object-oriented and non-object-oriented programming systems. The scope of the invention is defined by the claims and their equivalents.

We claim:

1. A streaming media player providing content protection and digital rights management, including:

a port configured to receive a digital bit stream, the digital bit stream including:

content which is encrypted at least in part, and

a secure container including control information for controlling use of the content, including at least one key suitable for decryption of at least a portion of the content; and

a control arrangement including:

means for opening secure containers and extracting cryptographic keys, and

means for decrypting the encrypted portion of the content.

2. The player of Claim 1 in which the digital bit stream includes at least two sub-streams which have been muxed together, at least one of the sub-streams including compressed information, and

wherein the player further includes:

a demux designed to separate and route the sub-streams;

a decompression unit configured to decompress at least one of the sub-streams, the decompression unit and the demux being connected by a pathway for the transmission of information; and

a rendering unit designed to process decompressed content information for rendering.

3. The player of Claim 2, further including:

a stream controller operatively connected to the decompression unit, the stream controller including decryption functionality configured to decrypt at least a portion of a sub-stream and pass the decrypted sub-stream to the decompression unit.

4. The player of Claim 3, further including:

a path between the control arrangement and the stream controller to enable the control arrangement to pass at least one key to the stream controller for use with the stream controller's decryption functionality.

5. The player of Claim 4, further including:

a feedback path from the rendering unit to the control arrangement to allow the control arrangement to receive information from the rendering unit regarding the identification of objects which are to be rendered or have been rendered.

6. The player of Claim 1, wherein the digital bit stream is encoded in MPEG-4 format.

7. The player of Claim 1, wherein the digital bit stream is encoded in MP3 format.

8. The player of Claim 4, wherein the control arrangement contains a rule or rule set associated with governance of at least one sub-stream or object.

9. The player of Claim 8, wherein the rule or rule set is delivered from an external source.

10. The player of Claim 9, wherein the rule or rule set is delivered as part of the digital bit stream.

11. The player of Claim 8, wherein the rule or rule set specifies conditions under which the governed sub-stream or object may be decrypted.

12. The player of Claim 8, wherein the rule or rule set governs at least one aspect of access to or use of the governed sub-stream or object.

13. The player of Claim 12, wherein the governed aspect includes making copies of the governed sub-stream or object.

14. The player of Claim 12, wherein the governed aspect includes transmitting the governed sub-stream or object through a digital output port.

15. The player of Claim 14, wherein the rule or rule set specifies that the governed sub-stream or object can be transferred to a second device, but rendering of the governed sub-stream or object must be disabled in the first device prior to or during the transfer.

16. The player of Claim 15, wherein the second device includes rendering capability, lacks at least one feature present in the streaming media player, and is at least somewhat more portable than the streaming media player.

17. The player of Claim 11, wherein the control arrangement contains at least two rules governing access to or use of the same governed sub-stream or object.

18. The player of Claim 17, wherein a first of the two rules was supplied by a first entity, and the second of the two rules was supplied by a second entity.

19. The player of Claim 18, wherein the first rule controls at least one aspect of operation of the second rule.

20. The player of Claim 12, wherein the governed aspect includes use of at least one budget.

21. The player of Claim 12, wherein the governed aspect includes a requirement that audit information be provided.

22. The player of Claim 1, wherein the control arrangement includes tamper resistance.

23. A digital bit stream including:

content information that is compressed and at least in part encrypted; and

a secure container including

governance information for the governance of at least one aspect of access to or use of at least a portion of the content information; and

a key for decryption of at least a portion of the encrypted content information.

24. The digital bit stream of Claim 23, wherein the content information is encoded in MPEG-4 format.

25. The digital bit stream of Claim 23, wherein the content information is encoded in MP3 format.

26. A method of rendering a protected digital bit stream including:

receiving the protected digital bit stream,

passing the protected digital bit stream to a media player,

the media player reading first header information identifying a plugin used to process the protected digital bit stream, the first header information indicating that a first plugin is required;

the media player calling the first plugin;

the media player passing the protected digital bit stream to the first plugin;

the first plugin decrypting at least a portion of the protected digital bit stream;

the first plugin reading second header information identifying a second plugin necessary in order to render the decrypted digital bit stream;

the first plugin calling the second plugin;

the first plugin passing the decrypted digital bit stream to the second plugin;

the second plugin processing the decrypted digital bit stream, the processing including decompressing at least a portion of the decrypted digital bit stream;

the second plugin passing the decrypted and processed digital bit stream to the media player; and

the media player enabling rendering of the decrypted and processed digital bit stream,

whereby the first plugin may be used in an architecture not designed for multiple stages of plugin processing.

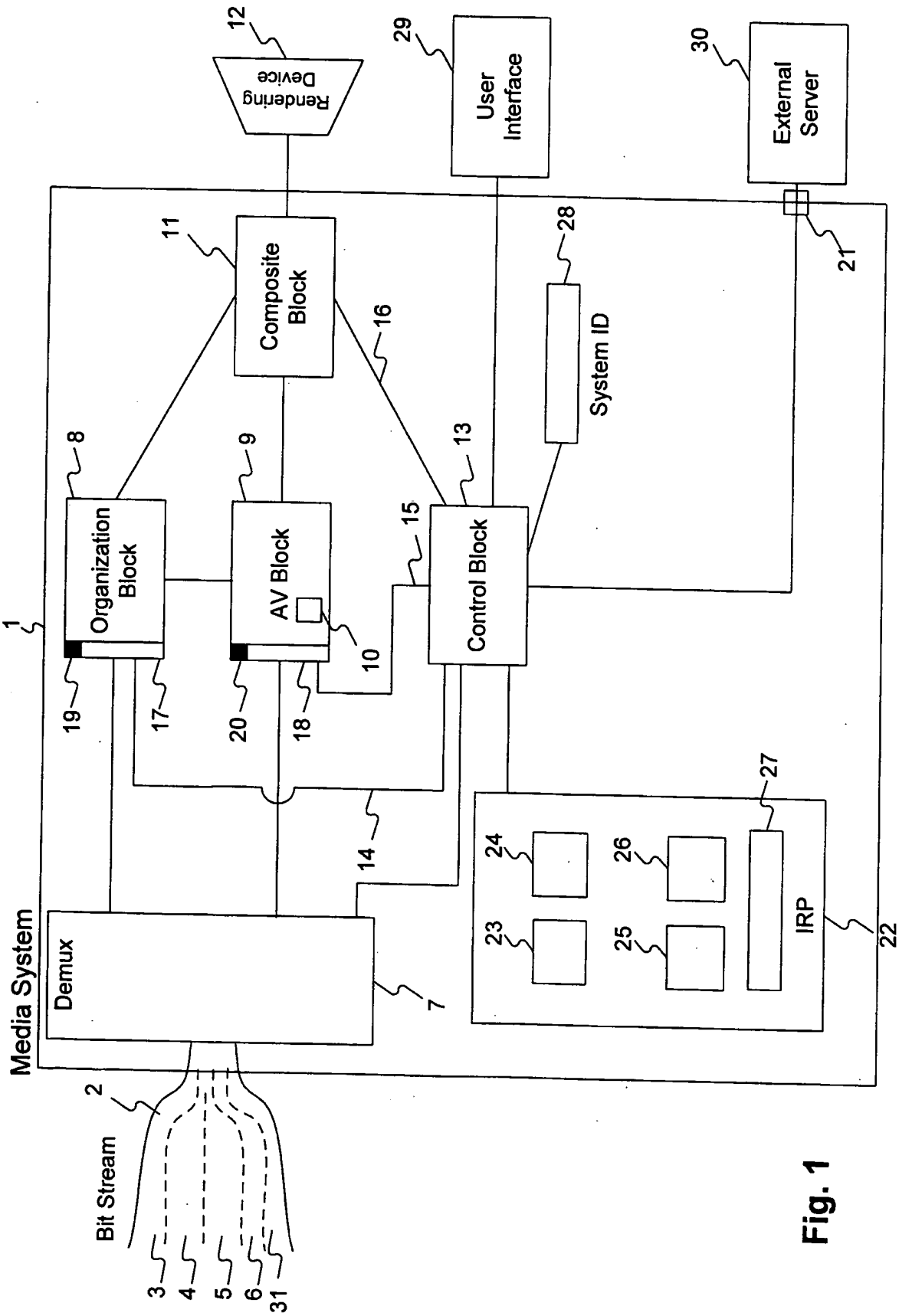

Rendering Device
12
User Interface
29
External Server
30
21
Composite Block
11
16
System ID
28
Organization Block
8
AV Block
9
10
Control Block
13
15
19
17
20
18
1
Media System
Demux
7
14
24
26
27
23
25
IRP
22
2
Bit Stream
3
4
5
6
31

Fig. 1

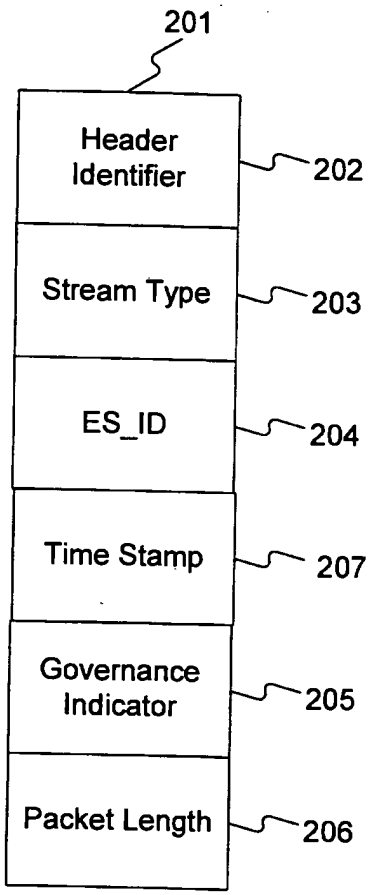

201
Header Identifier
202
Stream Type
203
ES_ID
204
Time Stamp
207
Governance Indicator
205
Packet Length
206

Fig. 2

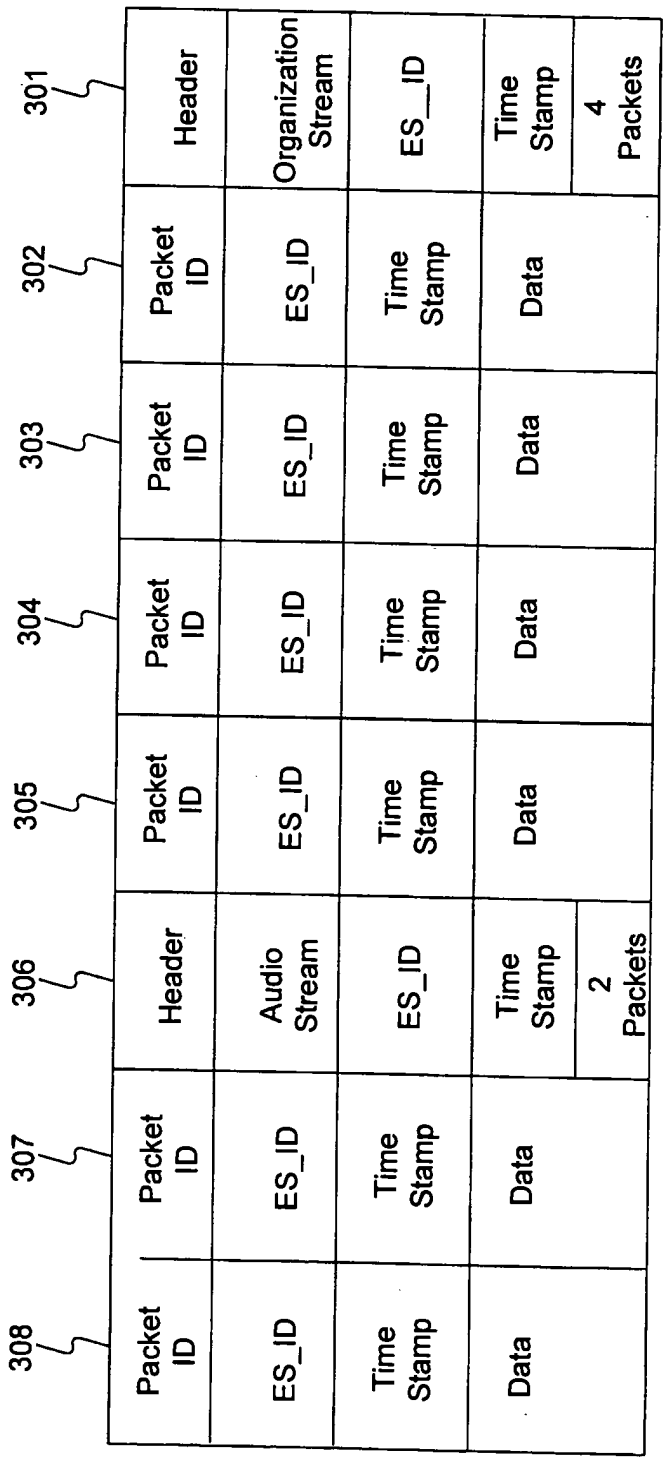

301
302
303
304
305
306
307
308
Header
Packet ID
Packet ID
Packet ID
Packet ID
Packet ID
Header
Packet ID
Organization Stream
ES_ID
Time Stamp
Data
ES_ID
Time Stamp
Data
ES_ID
Time Stamp
Data
ES_ID
Time Stamp
Data
Audio Stream
ES_ID
Time Stamp
2 Packets
ES_ID
Time Stamp
Data
Time Stamp
4 Packets
ES_ID
Time Stamp
Data

Fig. 3

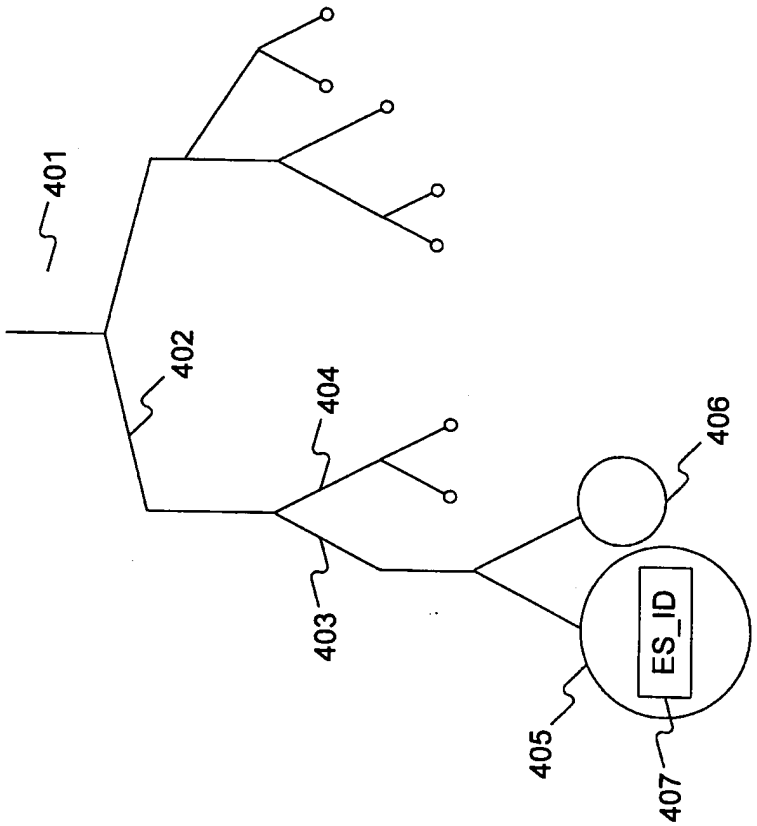


Fig. 4

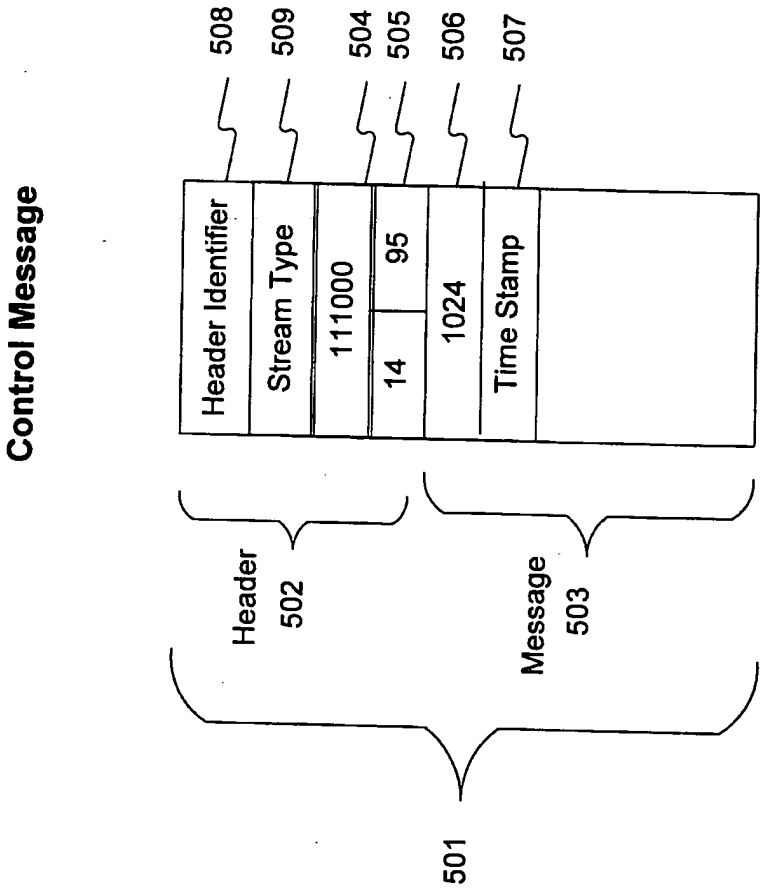

Control Message
508
509
504
505
506
507
Header Identifier
Stream Type
111000
14
95
1024
Time Stamp
Header
502
Message
503
501

Fig. 5

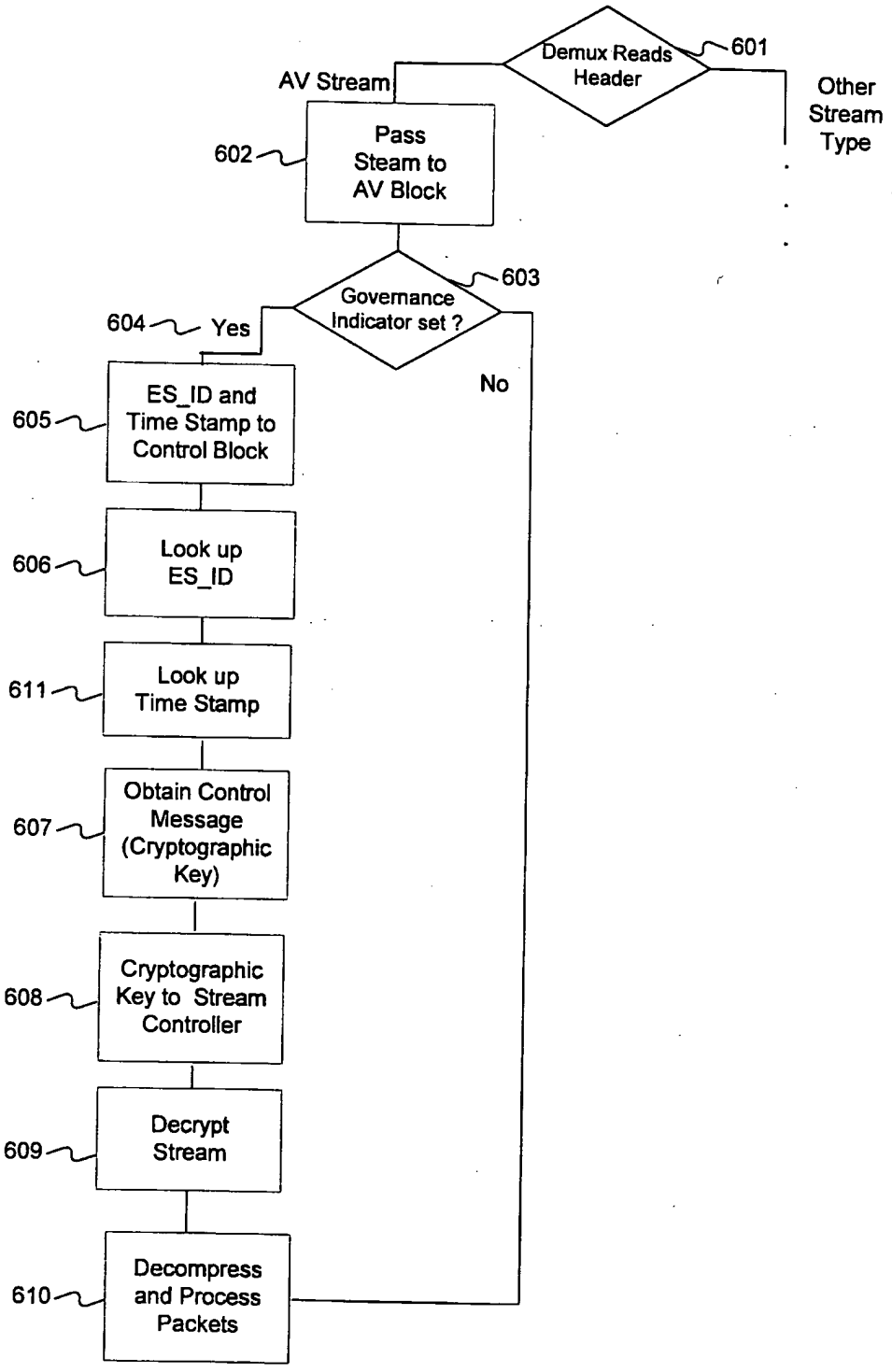

Demux Reads Header
601
AV Stream
Other Stream Type
602
Pass Steam to AV Block
603
Governance Indicator set ?
604
Yes
No
605
ES_ID and Time Stamp to Control Block
606
Look up ES_ID
611
Look up Time Stamp
607
Obtain Control Message (Cryptographic Key)
608
Cryptographic Key to Stream Controller
609
Decrypt Stream
610
Decompress and Process Packets

Fig. 6

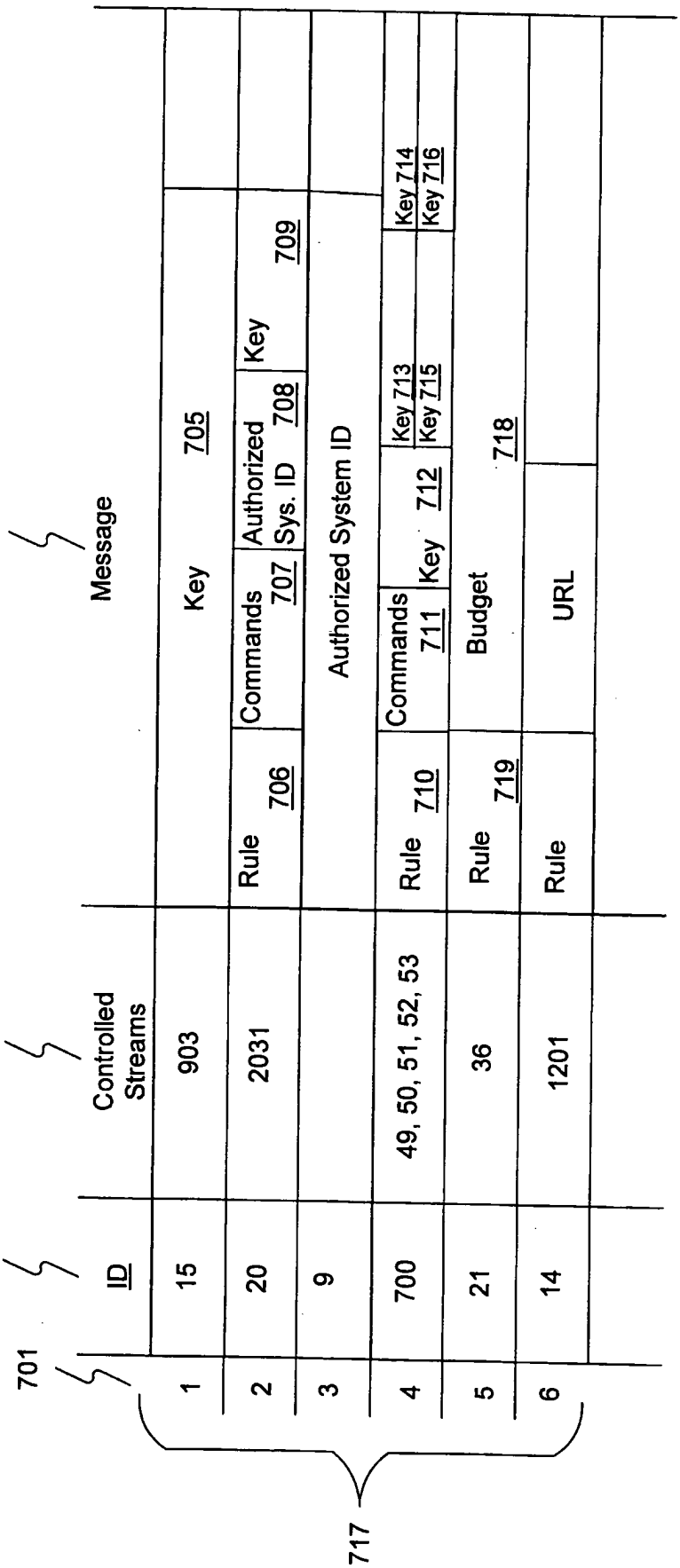

701
702
703
704
717
ID
Controlled Streams
Message
1
15
903
2
20
2031
3
9
4
700
49, 50, 51, 52, 53
5
21
36
6
14
1201
Key 705
Rule 706
Commands 707
Authorized Sys. ID 708
Key 709
Authorized System ID
Rule 710
Commands 711
Key 712
Key 713
Key 715
Key 714
Key 716
Rule 719
Budget 718
Rule
URL

Fig. 7

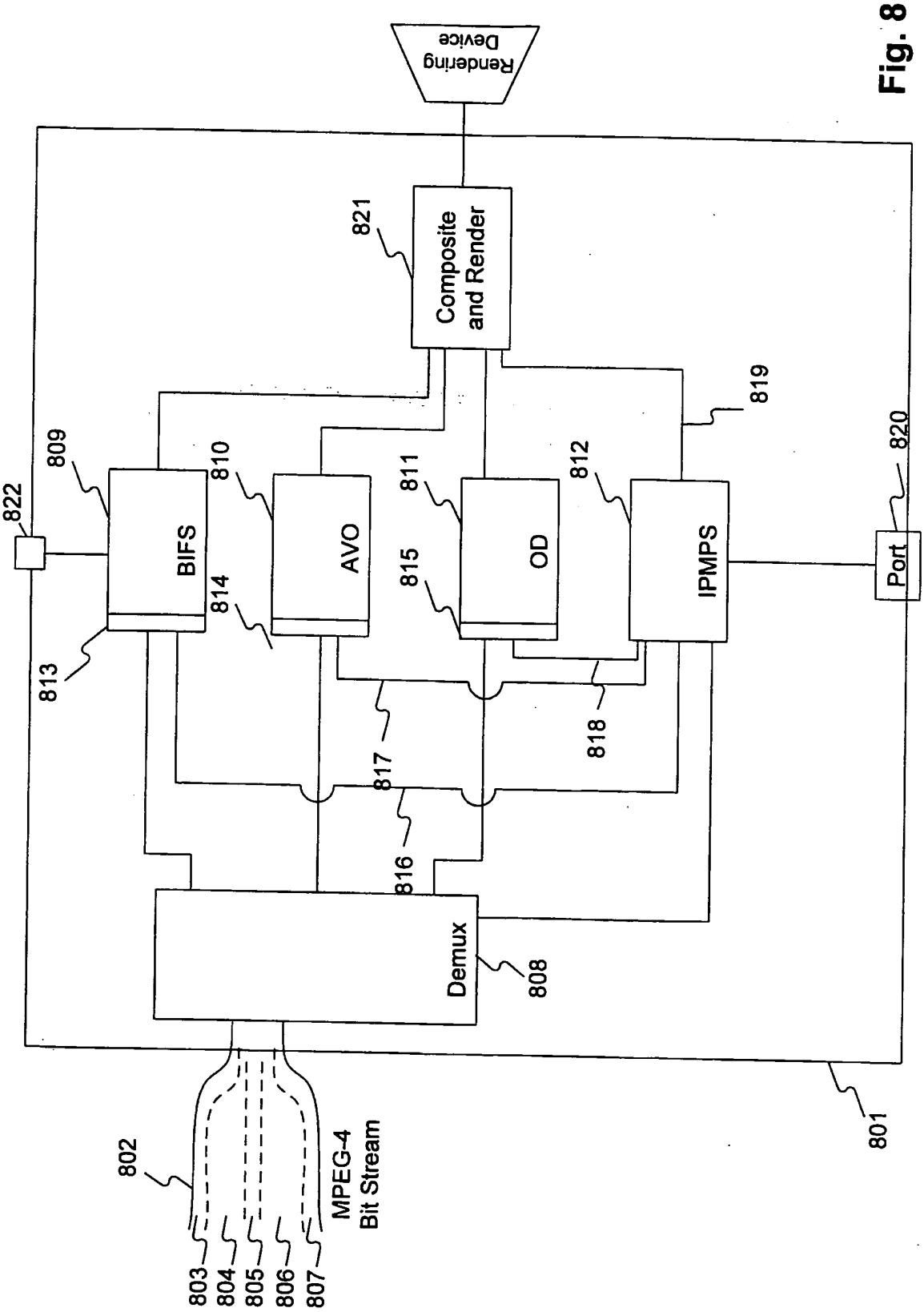

Rendering Device
Composite and Render
821
819
820
809
810
811
812
822
BIFS
AVO
OD
IPMS
Port
814
815
813
817
818
816
Demux
808
801
802
MPEG-4 Bit Stream
803
804
805
806
807

Fig. 8

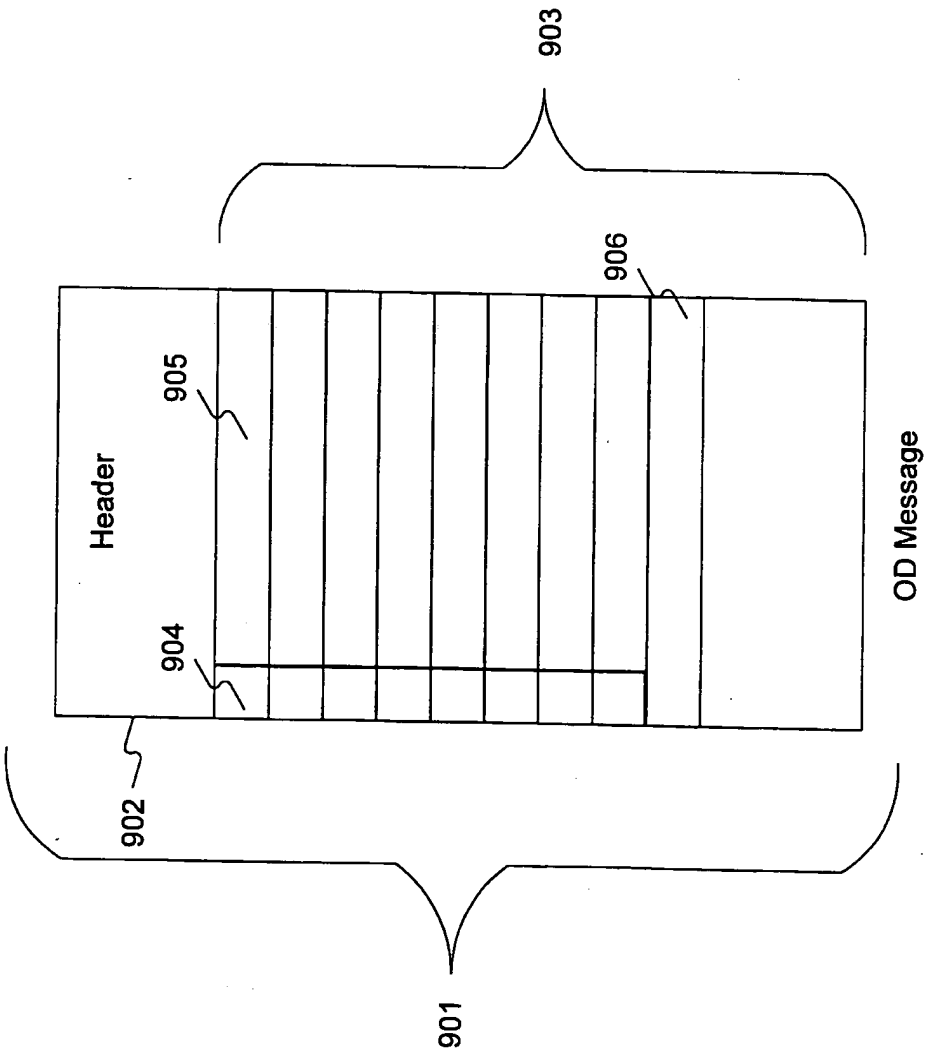


Fig. 9

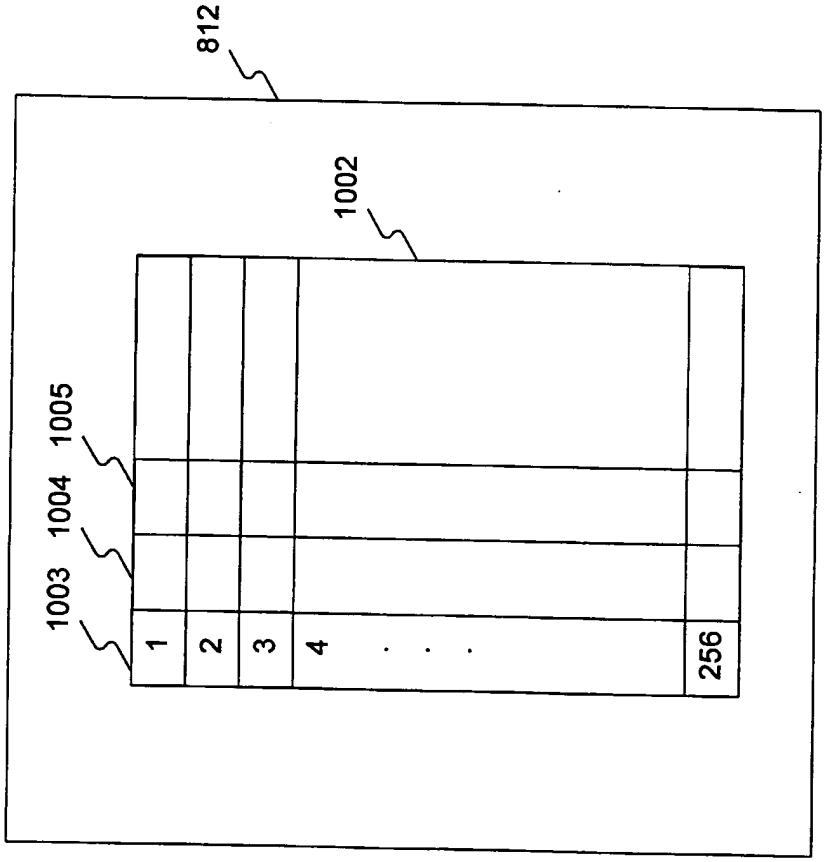


Fig. 10

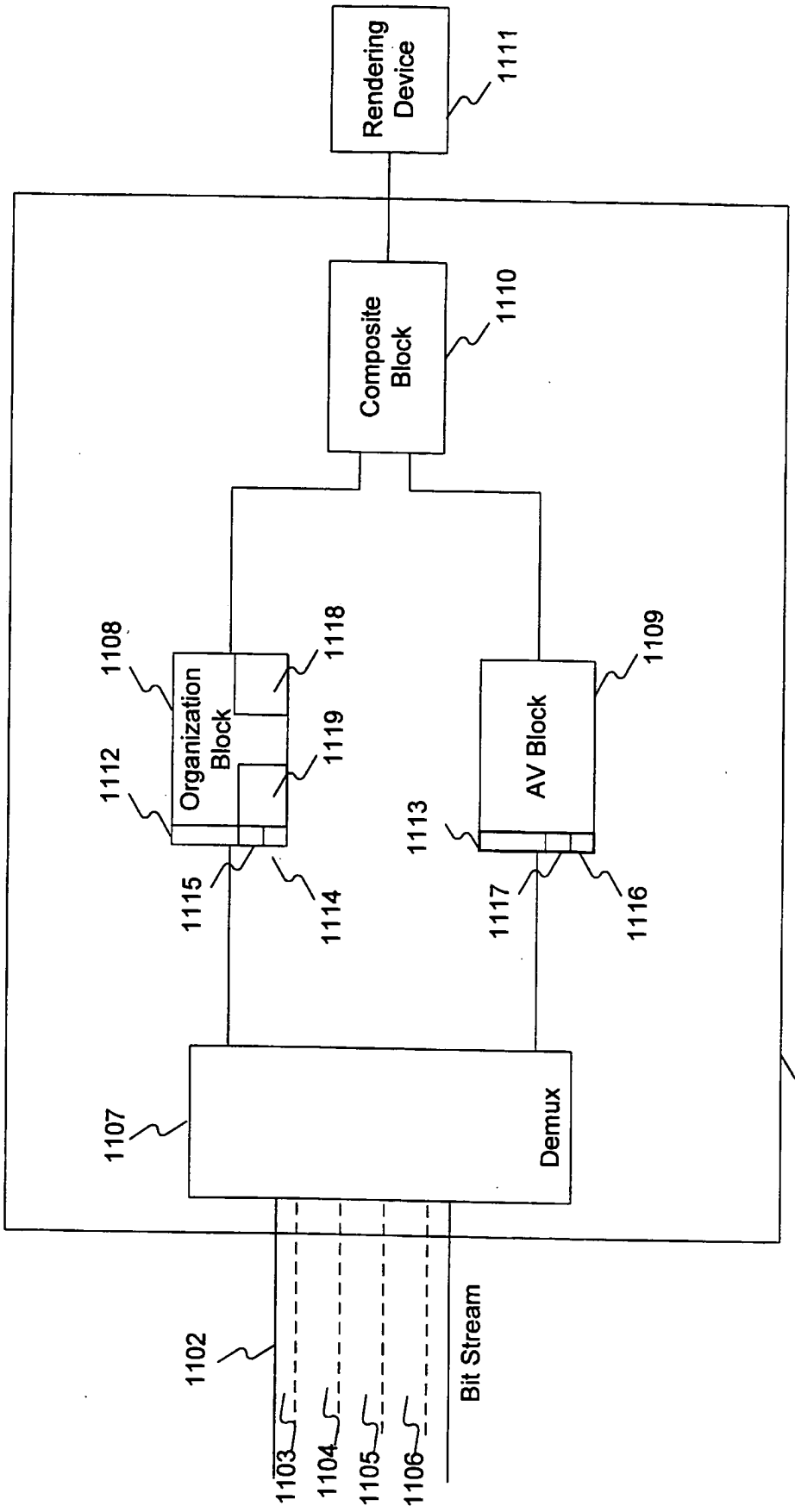

Rendering Device
1111
Composite Block
1110
Organization Block
1108
1118
1119
1112
1115
1114
AV Block
1109
1113
1117
1116
Demux
1107
1101
1102
Bit Stream
1103
1104
1105
1106

Fig. 11

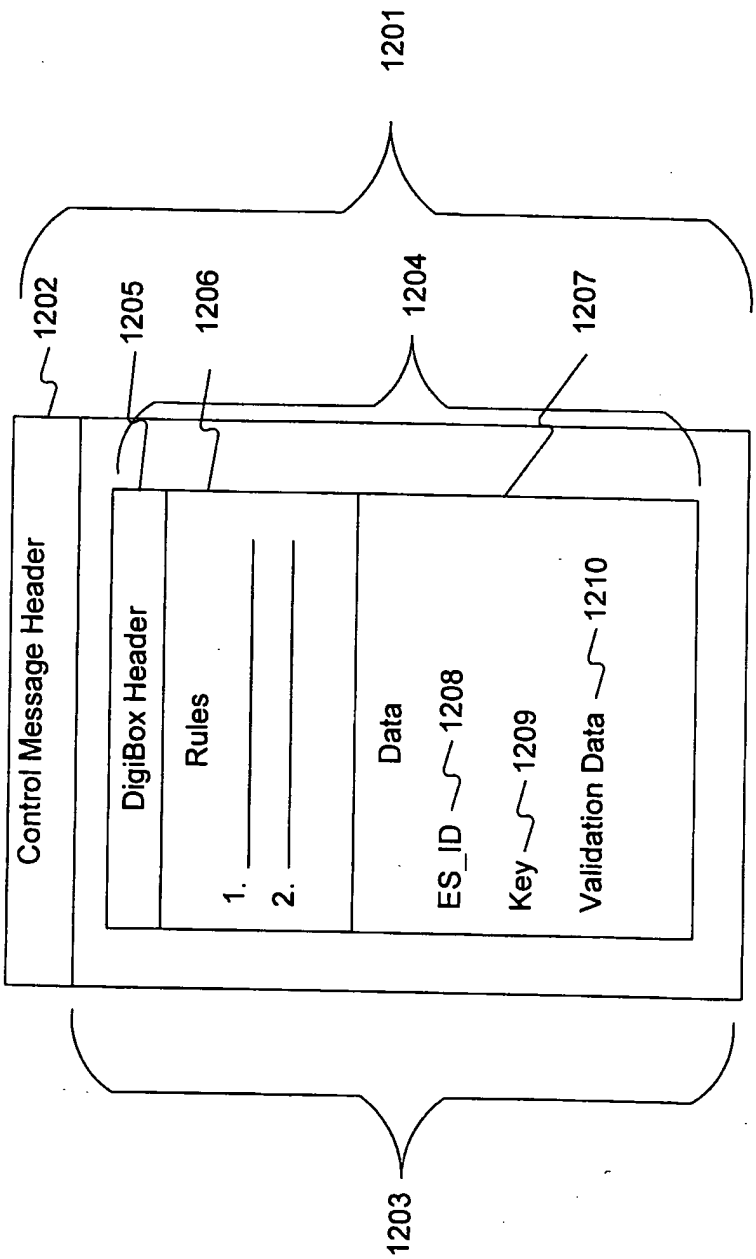

1201
1202
1205
1206
1204
1207
Control Message Header
DigiBox Header
Rules
1.
2.
Data
ES_ID
1208
Key
1209
Validation Data
1210
1203

Fig. 12

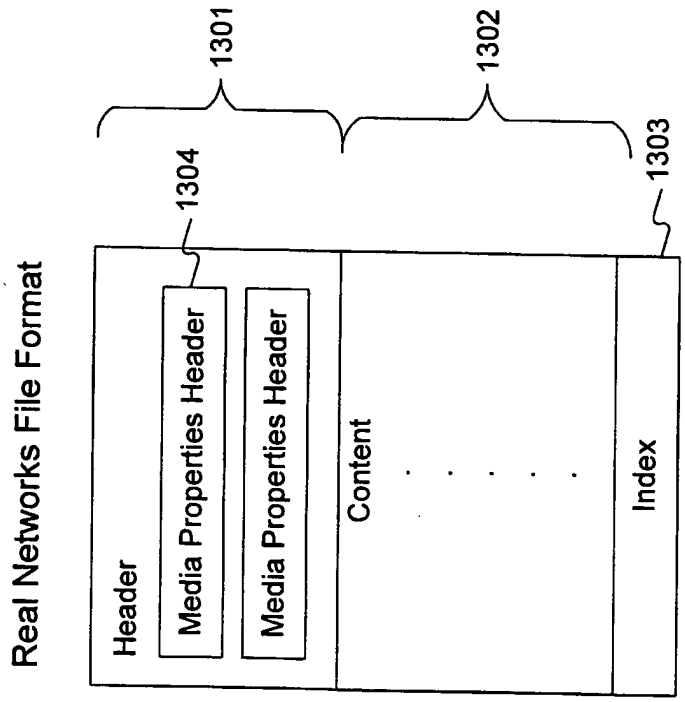

Real Networks File Format
Header
Media Properties Header
Media Properties Header
Content
Index
1301
1302
1303
1304

Fig. 13

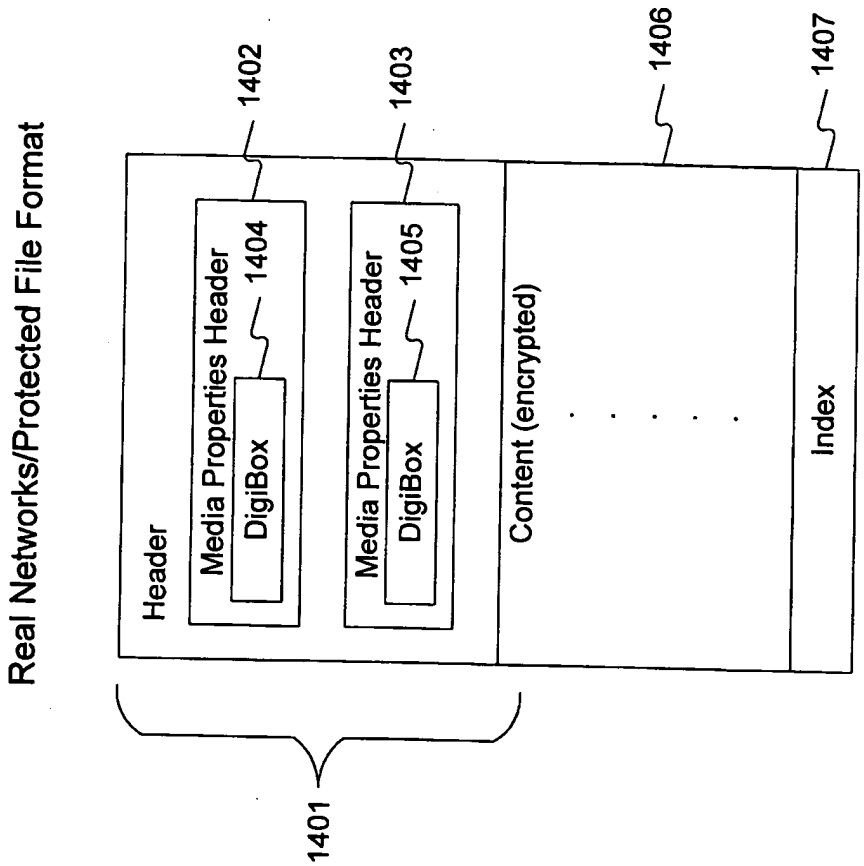

Real Networks/Protected File Format
1402
1403
1406
1407
Header
Media Properties Header
1404
DigiBox
Media Properties Header
1405
DigiBox
Content (encrypted)
Index
1401

Fig. 14

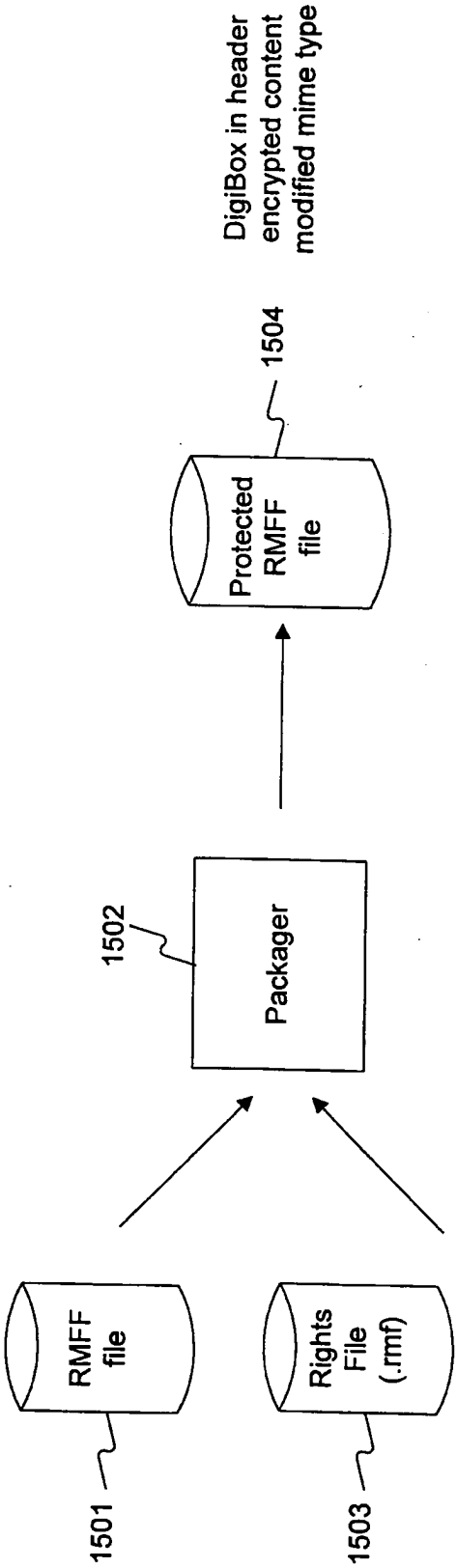

DigiBox in header
encrypted content
modified mime type
1504
Protected RMFF file
1502
Packager
RMFF file
Rights File (.rmf)
1501
1503

Fig. 15

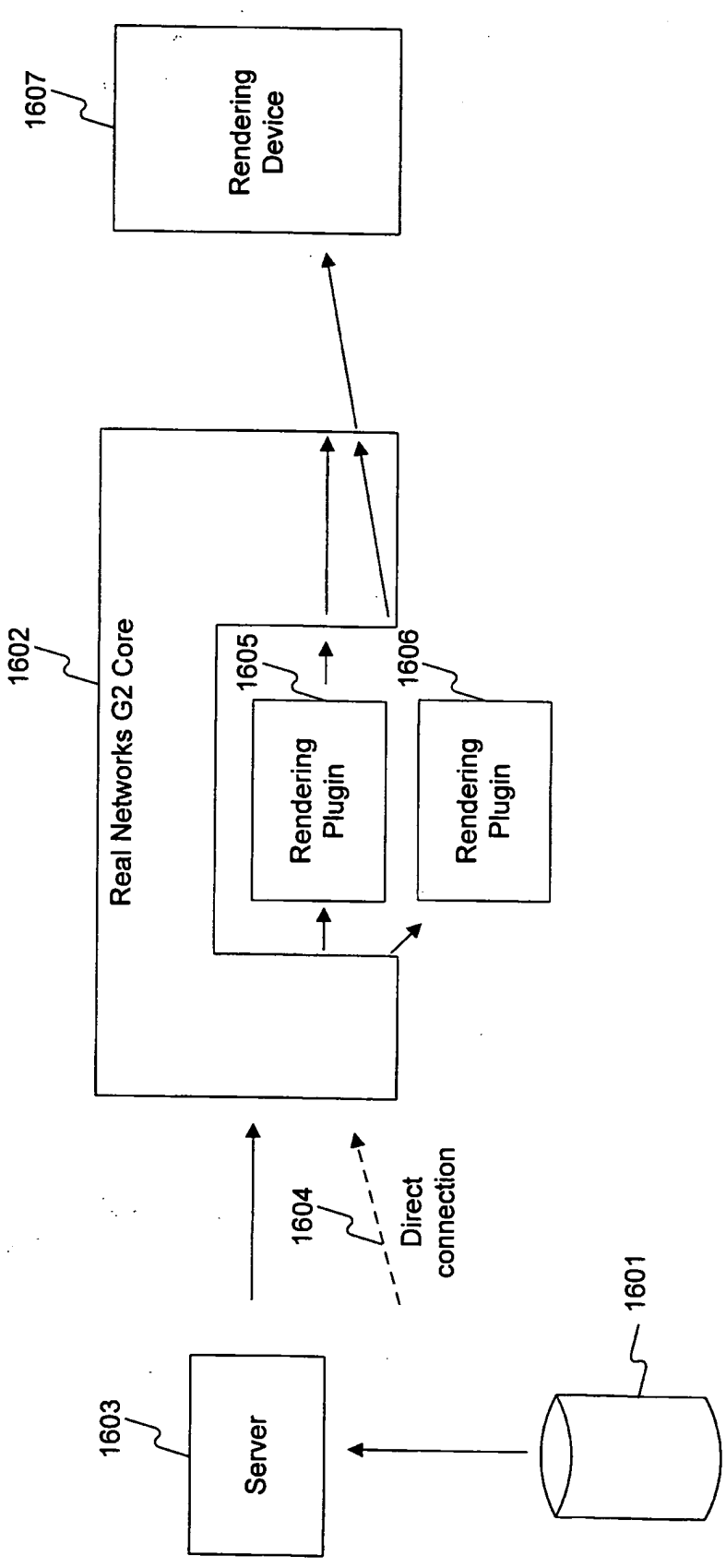

1607
Rendering Device
1602
Real Networks G2 Core
1605
1606
Rendering Plugin
Rendering Plugin
1604
Direct connection
1601
1603
Server

Fig. 16

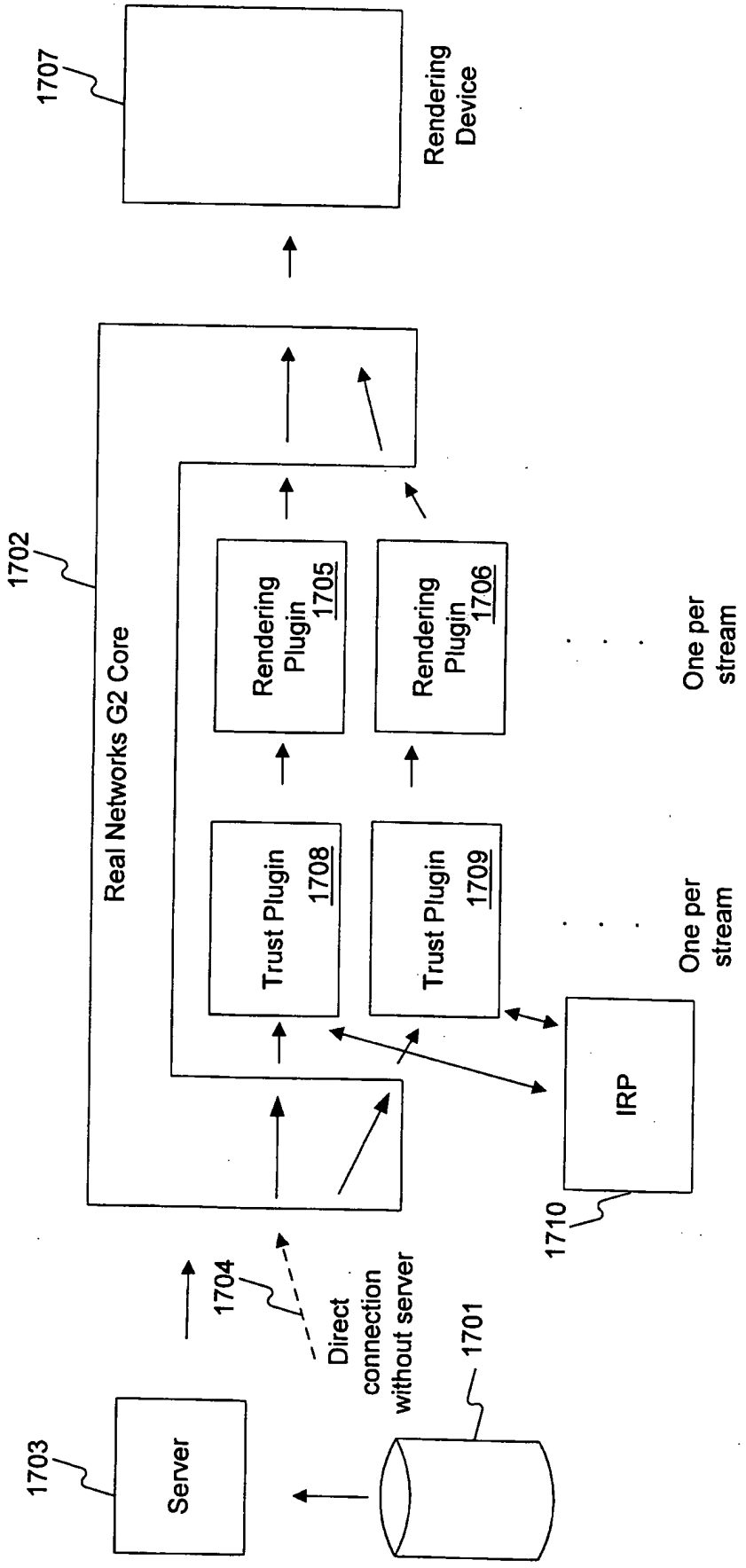

1707
Rendering Device
1702
Real Networks G2 Core
Rendering Plugin 1705
Rendering Plugin 1706
One per stream
Trust Plugin 1708
Trust Plugin 1709
One per stream
IRP
1710
1704
Direct connection without server
1701
1703
Server

Fig. 17

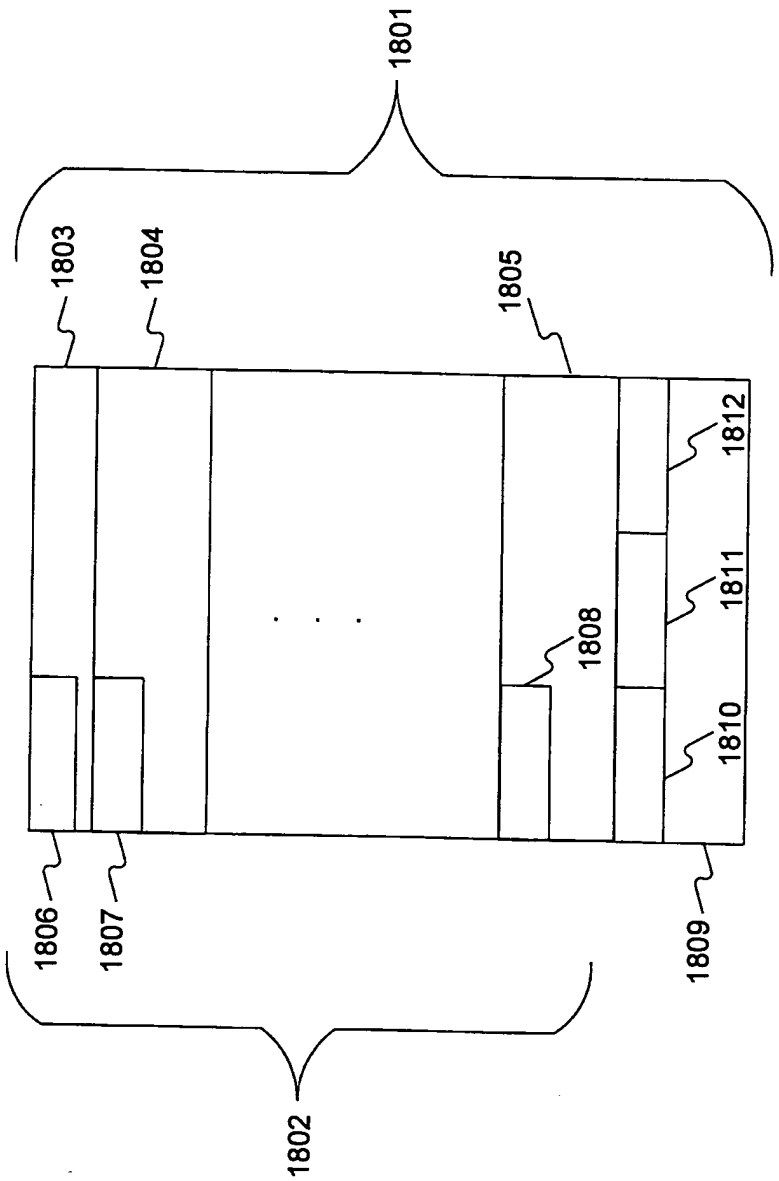
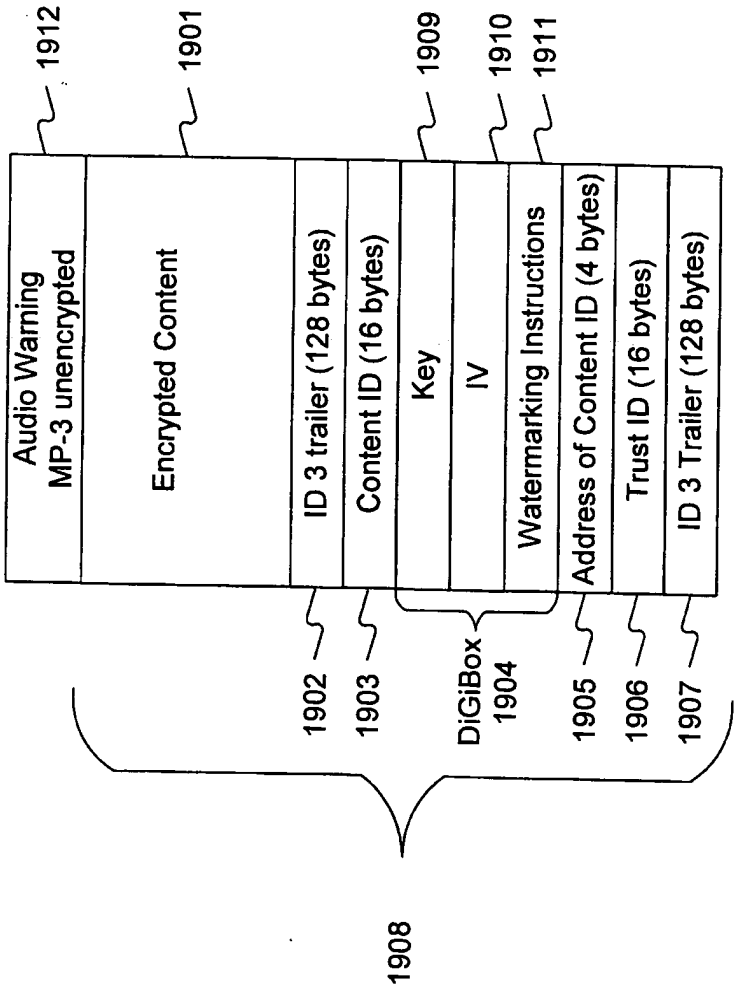


Fig. 18



Protected MP3 Format

Fig. 19

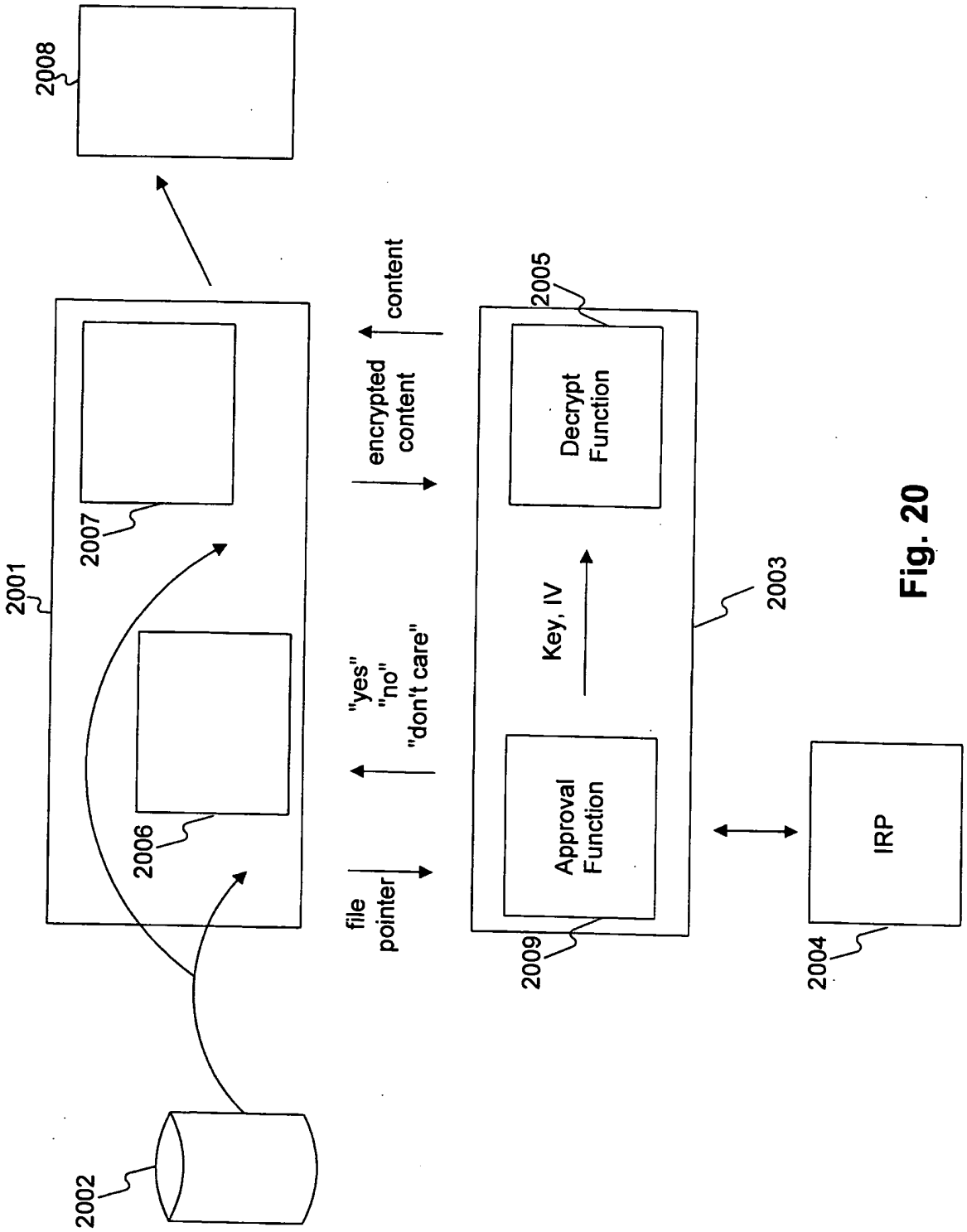

2008
2001
2007
2006
2002
content
encrypted content
"yes"
"no"
"don't care"
file pointer
2005
Decrypt Function
Key, IV
Approval Function
2009
2003
IRP
2004

Fig. 20

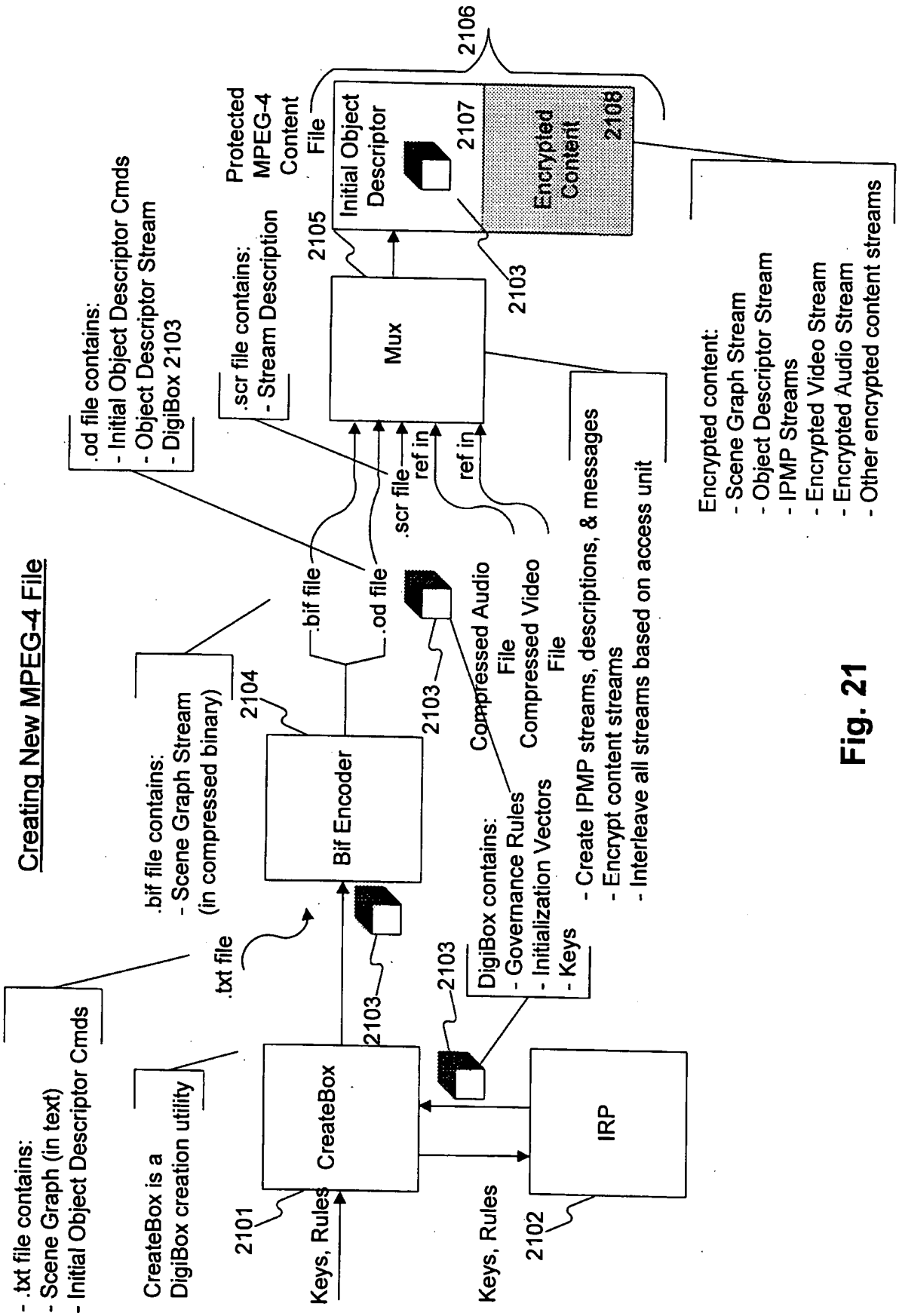

Creating New MPEG-4 File
- .txt file contains:
- Scene Graph (in text)
- Initial Object Descriptor Cmds
CreateBox is a DigiBox creation utility
CreateBox
2101
Keys, Rules
.txt file
.bif file contains:
- Scene Graph Stream
(in compressed binary)
2104
Bif Encoder
2103
2103
DigiBox contains:
- Governance Rules
- Initialization Vectors
- Keys
Keys, Rules
IRP
2102
.bif file
.od file
.od file contains:
- Initial Object Descriptor Cmds
- Object Descriptor Stream
- DigiBox 2103
.scr file
ref in
ref in
.scr file contains:
- Stream Description
Mux
2103
2105
Initial Object Descriptor
2107
Encrypted Content
2108
2106
Protected MPEG-4 Content File
Compressed Audio File
Compressed Video File
- Create IPMP streams, descriptions, & messages
- Encrypt content streams
- Interleave all streams based on access unit
Encrypted content:
- Scene Graph Stream
- Object Descriptor Stream
- IPMP Streams
- Encrypted Video Stream
- Encrypted Audio Stream
- Other encrypted content streams

Fig. 21

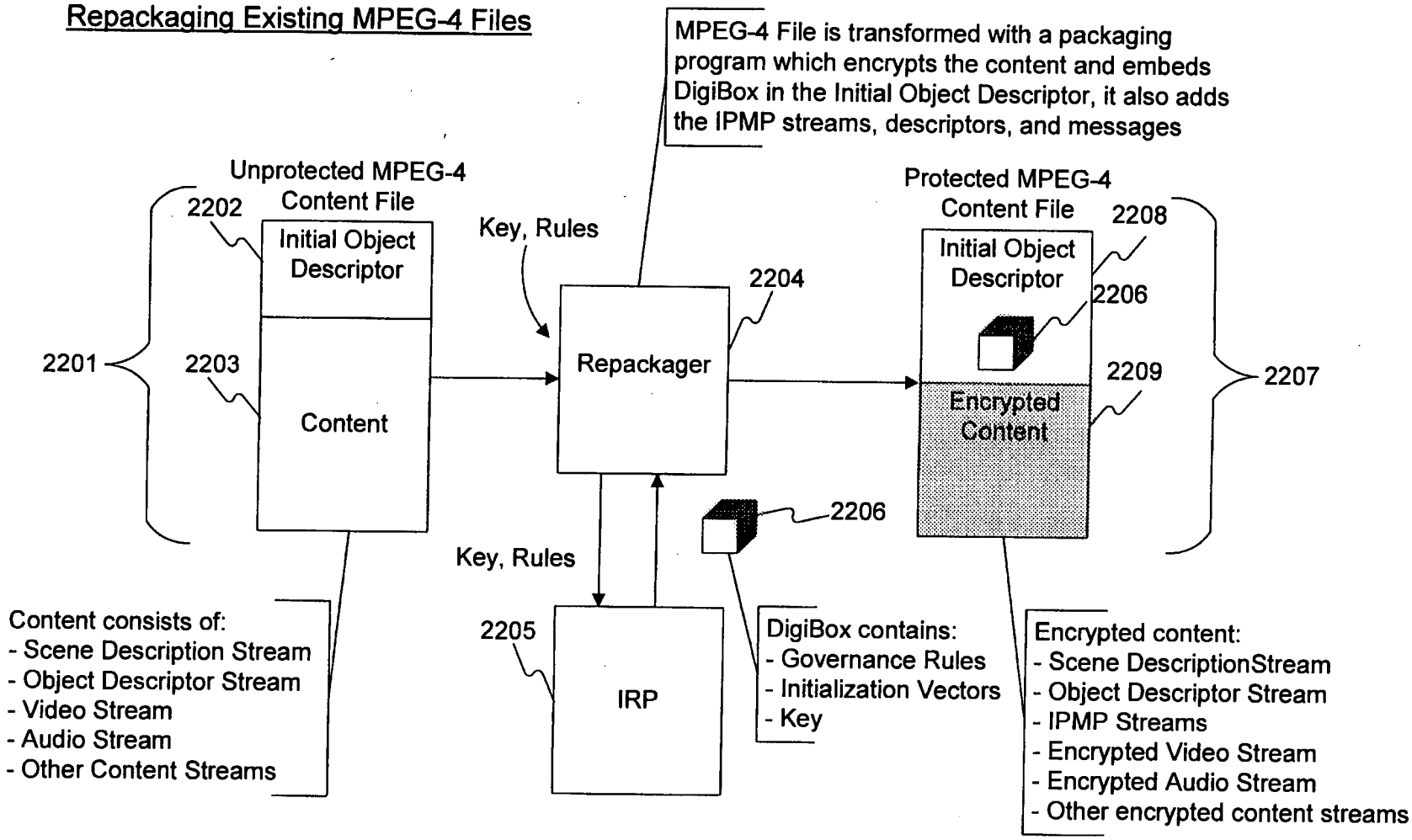


Fig. 22

Petitioner Apple Inc. - Exhibit 1004, p. 5159

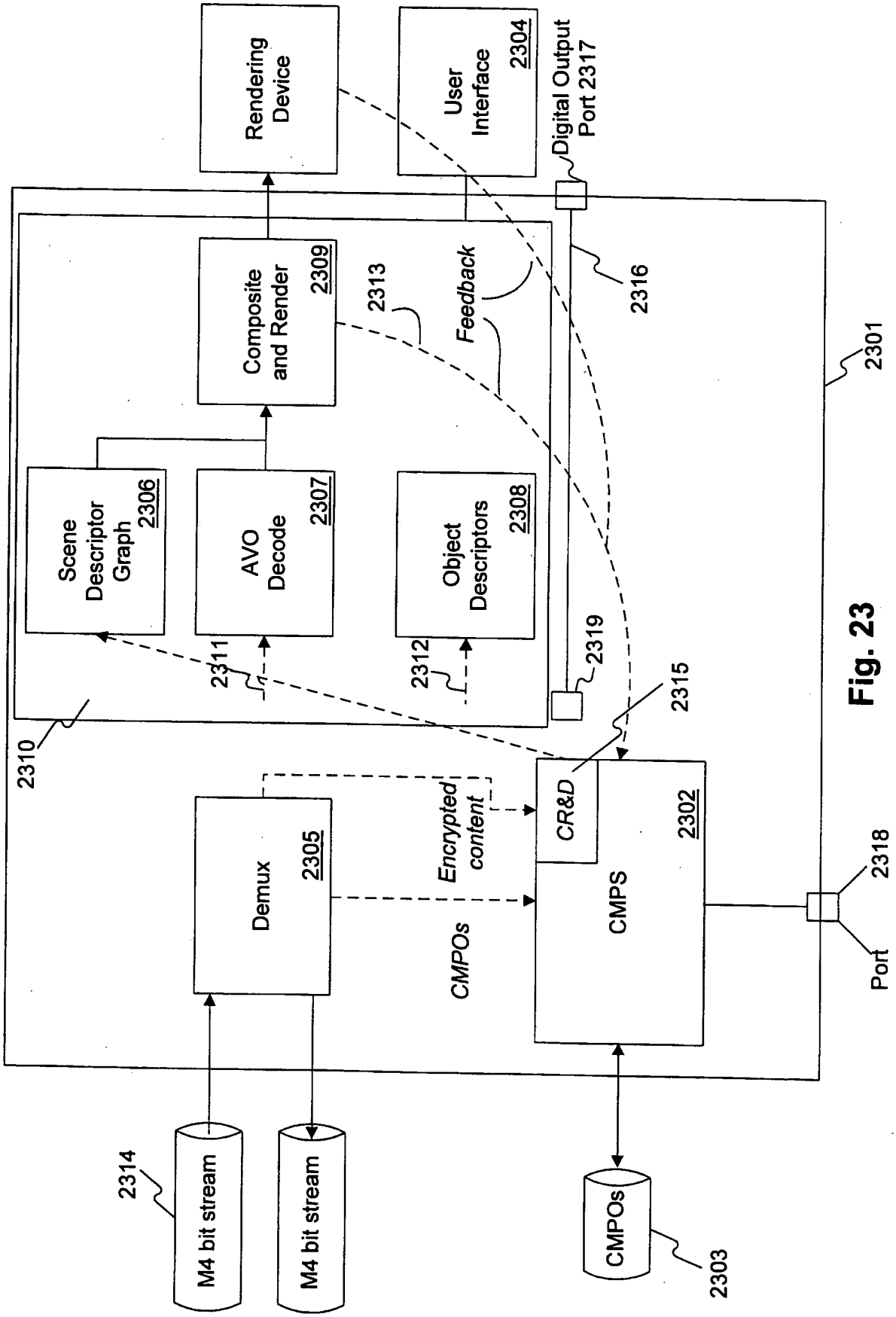

Rendering Device
User Interface 2304
Digital Output Port 2317
Composite and Render 2309
2313
Feedback
2316
2301
Scene Descriptor Graph 2306
AVO Decode 2307
Object Descriptors 2308
2311
2312
2319
2315
2310
CR&D
2302
Encrypted content
Demux 2305
CMPS
2318
Port
CMPOs
2314
M4 bit stream
M4 bit stream
CMPOs
2303

Fig. 23

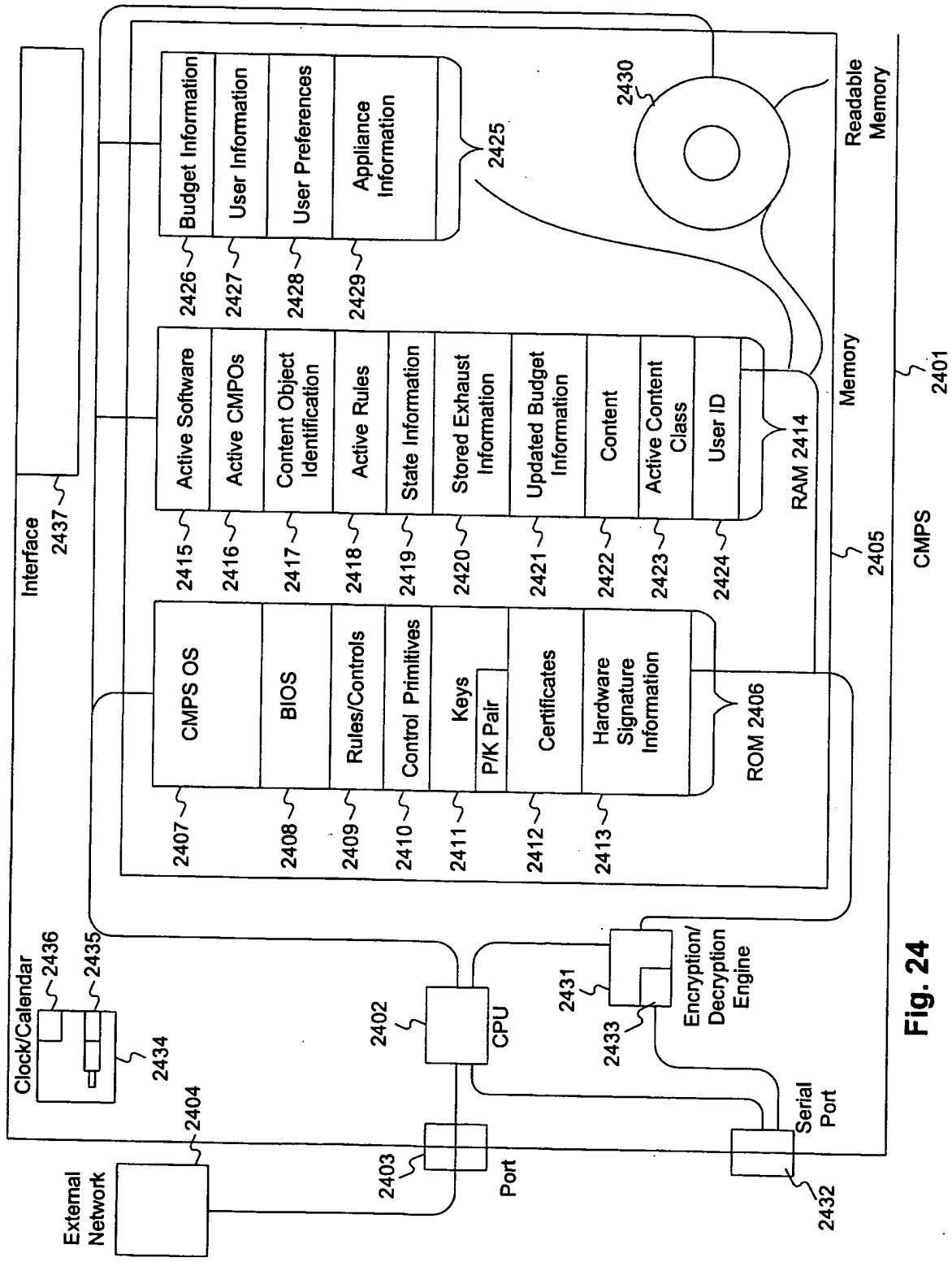

Budget Information
User Information
User Preferences
Appliance Information
2425
2426
2427
2428
2429
2430
Readable Memory
Active Software
Active CMPOs
Content Object Identification
Active Rules
State Information
Stored Exhaust Information
Updated Budget Information
Content
Active Content Class
User ID
RAM 2414
Memory
2401
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2405
CMPS
Interface
2437
CMPS OS
BIOS
Rules/Controls
Control Primitives
Keys
P/K Pair
Certificates
Hardware Signature Information
ROM 2406
2407
2408
2409
2410
2411
2412
2413
Clock/Calendar
2436
2435
2434
2404
External Network
2403
Port
2402
CPU
2431
2433
Encryption/ Decryption Engine
Serial Port
2432

Fig. 24

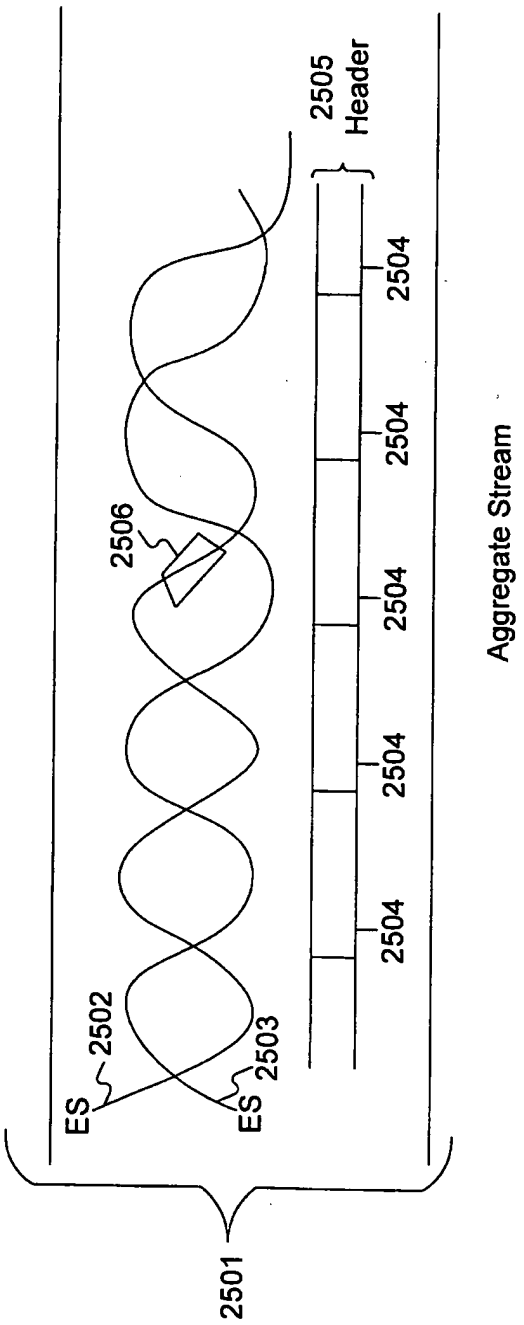

2501
ES
2502
ES
2503
2506
2505
Header
2504
2504
2504
2504
2504
Aggregate Stream

Fig. 25

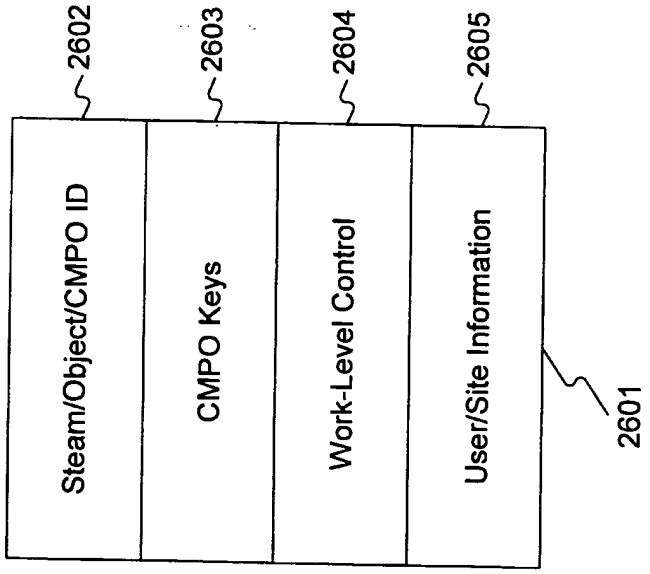

2602
2603
2604
2605
Steam/Object/CMPO ID
CMPO Keys
Work-Level Control
User/Site Information
2601

Fig. 26

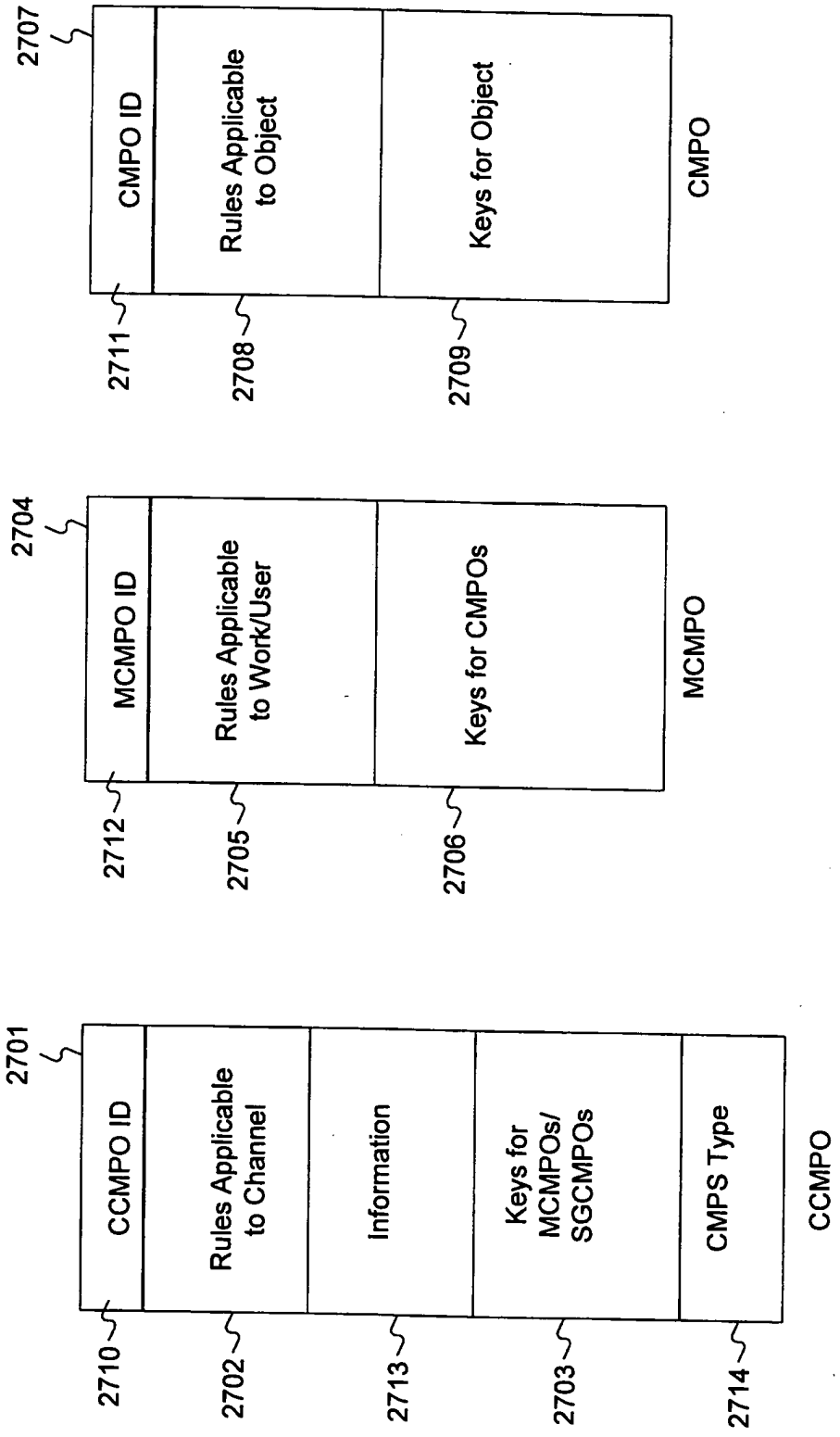

2707
CMPO ID
Rules Applicable to Object
Keys for Object
CMPO
2711
2708
2709
2704
MCMPO ID
Rules Applicable to Work/User
Keys for CMPOs
MCMPO
2712
2705
2706
2701
CCMPO ID
Rules Applicable to Channel
Information
Keys for MCMPOs/ SGCMPOs
CMPS Type
CCMPO
2710
2702
2713
2703
2714

Fig. 27

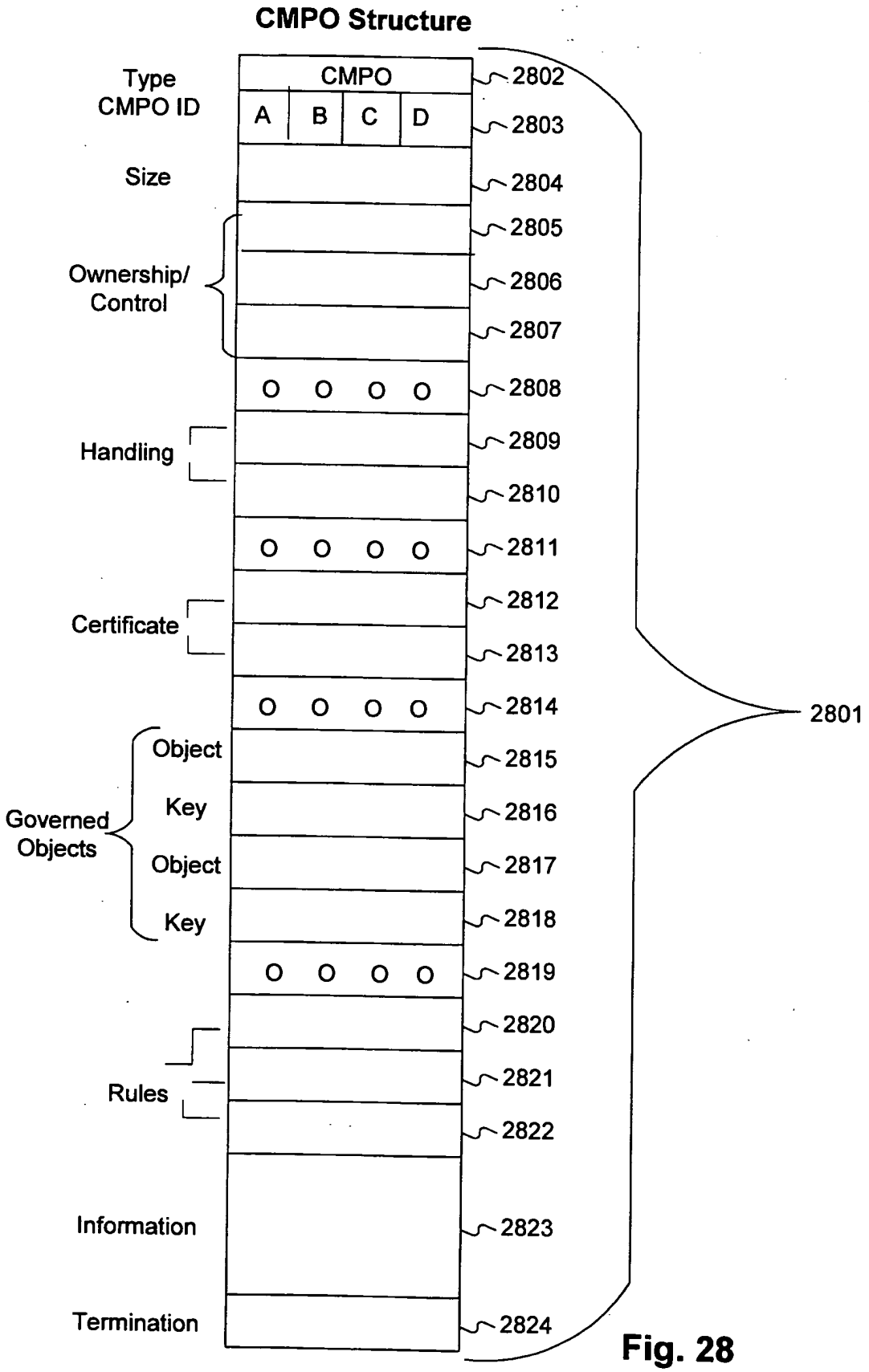

CMPO Structure
Type
CMPO ID
CMPO
A
B
C
D
Size
Ownership/
Control
O O O O
Handling
O O O O
Certificate
O O O O
Governed
Objects
Object
Key
Object
Key
O O O O
Rules
Information
Termination
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2801

Fig. 28

INTERNATIONAL SEARCH REPORT

Inter ɔnal Application No
PCT/US 99/05734

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04N7/167 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H04N G06F G11B

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 763 936 A (LG ELECTRONICS INC) 19 March 1997	1-4, 6-14, 17-20, 22-25
A	see abstract see column 6, line 27 - column 8, line 4 see column 9, line 6 - column 11, line 43 see column 16, line 47 - column 18, line 38 see figures 4,6A,6B,7 see figures 10,16 --- -/--	5,15,16, 21,26

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
1 July 1999	09/07/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Hampson, F

1

Form PCT/ISA/210 (second sheet) (July 1992)

Petitioner Apple Inc. - Exhibit 1004, p. 5166

INTERNATIONAL SEARCH REPORT

Inte: ›nal Application No
PCT/US 99/05734

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 714 204 A (LG ELECTRONICS INC) 29 May 1996	1-4, 6-14, 22-26
A	see abstract see page 6, line 14 - page 8, line 45 see figures 7,19A,B,20	5,15-21
A	EP 0 715 246 A (XEROX CORP) 5 June 1996 see abstract see page 3, line 46 - page 8, line 27 see figures 1-3,4A,4B	1-26
A	WO 97 25816 A (SONY CORP ;INOUE HAJIME (US); LEE CHUEN CHIEN (US); SONY ELECTRONI) 17 July 1997 see abstract see page 7, line 10 - page 10, line 7 see figures 2,3	1-26
A	EP 0 800 312 A (MATSUSHITA ELECTRIC IND CO LTD) 8 October 1997 see abstract see column 50, line 20 - column 51, line 53	1,6,7, 23-25

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

Petitioner Apple Inc. - Exhibit 1004, p. 5167

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter ional Application No
PCT/US 99/05734

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0763936 A	19-03-1997	CN 1150738 A	28-05-1997
		JP 9093561 A	04-04-1997
		US 5799081 A	25-08-1998
EP 0714204 A	29-05-1996	CN 1137723 A	11-12-1996
		JP 8242438 A	17-09-1996
		US 5757909 A	26-05-1998
EP 0715246 A	05-06-1996	US 5638443 A	10-06-1997
		JP 8263439 A	11-10-1996
WO 9725816 A	17-07-1997	AU 1344097 A	01-08-1997
		CN 1209247 A	24-02-1999
		EP 0882357 A	09-12-1998
		US 5889919 A	30-03-1999
EP 0800312 A	08-10-1997	WO 9714249 A	17-04-1997
		CN 1168054 A	17-12-1997
		EP 0789361 A	13-08-1997
		JP 10079174 A	24-03-1998

Petitioner Apple Inc. - Exhibit 1004, p. 5168

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 067 556**
B1

(12) # EUROPEAN PATENT SPECIFICATION

(45) Date of publication of patent specification: **18.04.90**

(51) Int. Cl.[5]: **G 06 F 9/46**

(21) Application number: **82302596.0**

(22) Date of filing: **21.05.82**

(60) **Divisional applications 88200917, 88200916, 88200921 filed on 09.05.88.**

(54) **Digital data processing system.**

(30) Priority: **22.05.81 US 266413**
22.05.81 US 266539 **22.05.81 US 266414**
22.05.81 US 266521 **22.05.81 US 266532**
22.05.81 US 266415 **22.05.81 US 266403**
22.05.81 US 266409 **22.05.81 US 266408**
22.05.81 US 266424 **22.05.81 US 266401**
22.05.81 US 266421 **22.05.81 US 266524**
22.05.81 US 266404

(43) Date of publication of application:
22.12.82 Bulletin 82/51

(45) Publication of the grant of the patent:
18.04.90 Bulletin 90/16

(84) Designated Contracting States:
AT BE CH DE FR GB IT LI LU NL SE

(56) References cited:
Hasselmeier, Spruth: "Rechnerstrukturen", 1974, pp 75-103

Klar, Wichmann: "Mikroprogrammierung", June 1975, pp 159-163, 176-179, 185-187, 195-205, 214-215

(73) Proprietor: **DATA GENERAL CORPORATION**
Route 9
Westboro Massachusetts 01581 (US)

(72) Inventor: **Ahlstrom, John K.**
1309 San Domar
Mountain View California 94043 (US)
Inventor: **Bachman, Brett L.**
214 W. Canton Street Suite 4
Boston Massachusetts 02116 (US)
Inventor: **Belgard, Richard A.**
21250 Glenmont Drive
Saratoga California 95070 (US)
Inventor: **Bernstein, David H.**
41 Bay Colony Drive
Ashland Massachusetts 01721 (US)
Inventor: **Bratt, Richard Glenn**
9 Brook Trail Road
Wayland Massachusetts 01778 (US)
Inventor: **Clancy, Gerald F.**
13069 Jaccaranda Center
Saratoga California 95070 (US)
Inventor: **Farber, David A.**
1700 Lakewood Avenue
Durham North Carolina 27707 (US)
Inventor: **Gavrin, Edward S.**
Beaver Pond Road RFD 4
Lincoln Massachusetts 01773 (US)

Note: Within nine months from the publication of the mention of the grant of the European patent, any person may give notice to the European Patent Office of opposition to the European patent granted. Notice of opposition shall be filed in a written reasoned statement. It shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European patent convention).

Courier Press, Leamington Spa, England.

EP 0 067 556 B1

(56) References cited:

IBM TECHNICAL DISCLOSURE BULLETIN, vol. 22, no. 3, august 1979, pages 1286-1289, New York, US D.B. LOMET: "Regions for controlling the propagation of addressability in capability systems"

IBM TECHNICAL DISCLOSURE BULLETIN, vol. 15, no. 9, February 1973, pages 2721-2722 W.L. DUNNE: "Common Compiler/Interpreter for a programming language"

ADVANCES IN COMPUTERS, vol, 14, 1976, Academic Press, New York, US pages 231-272 D.K. HSIAO et al.:"Information Secure Systems"

IEEE,DIGEST OF PAPERS FROM COMPCON FALL MEETING, September 10-12, 1974, Washington, Micros and Minis, pages 15-17 Long Beach, US C.J. NEUHAUSER et al.: "Description of an emulation laboratory"

7th Annual Symposium on COMPUTER ARCHITECTURE, May 6-8, 1980, Conference Proceedings, pages 245-252 New York, US V. BERSTIS: "Security and protection of data in the IBM System/38

VERY LARGE DATA BASES, vol.9, no. 2C, October 1977, Third International Conference Proceedings, pages 507-514 Tokyo, JP D. DOWNS et al.: "A kernel design for a secure data base management system"

Inventor: Gruner, Ronald Hans
112 Dublin Wood Drive
Cary North Carolina 27514 (US)
Inventor: Houseman, David L.
1213 Selwyn Lane
Cary North Carolina 27511 (US)
Inventor: Jones, Thomas M. Jones
300 Reade Road
Chapel Hill North Carolina 27514 (US)
Inventor: Katz, Lawrence H.
10943 S. Forest Ridge Road
Oregon City Oregon 97045 (US)
Inventor: Mundie, Craig James
136 Castlewood Drive
Cary North Carolina (US)
Inventor: Pilat, John F.
1308 Ravenhurst Drive
Raleigh North Carolina 27609 (US)
Inventor: Richmond, Michael S.
Fearrington Post Box 51
Pittsboro North Carolina 27312 (US)
Inventor: Schleimer, Stephen I.
1208 Ellen Place
Chapel Hill North Carolina 27514 (US)
Inventor: Wallach, Steven J.
12436 Green Meadow Lane
Saratoga California 95070 (US)
Inventor: Wallach, Walter A., Jr.
1336 Medfield Road
Raleigh North Carolina 27607 (US)
Inventor: Wells, Douglas M.
106 Robin Road
Chapel Hill North Carolina 27514 (US)

(74) Representative: Pears, David Ashley et al
REDDIE & GROSE 16 Theobalds Road
London WC1X 8PL (GB)

Petitioner Apple Inc. - Exhibit 1004, p. 5170

Description

The present invention relates to a digital data processing system and, more particularly, to a multiprocessor digital data processing system suitable for use in a data processing network and having a simplified, flexible user interface and flexible, multileveled internal mechanism.

A general trend in the development of data processing systems has been towards systems suitable for use in interconnected data processing networks. Another trend has been towards data processing systems wherein the internal structure of the system is flexible, protected from users, and effectively invisible to the user and wherein the user is presented with a flexible and simplified interface to the system.

Certain problems and shortcomings affecting the realization of such a data processing system have appeared repeatedly in the prior art and must be overcome to create a data processing system having the above attributes. These prior art problems and limitations include the following topics.

First, the data processing systems of the prior art have not provided a system wide addressing system suitable for use in common by a large number of data processing systems interconnected into a network. Addressing systems of the prior art have not provided sufficiently large address spaces and have not allowed information to be permanently and uniquely identified. Prior addressing systems have not made provisions for information to be located and identified as to type or format, and have not provided sufficient granularity. In addition, prior addressing systems have reflected the physical structure of particular data processing systems. That is, the addressing systems have been dependent upon whether a particular computer was, for example, an 8, 16, 32, 64 or 128 bit machine. Since prior data processing systems have incorporated addressing mechanisms wherein the actual physical structure of the processing system is apparent to the user, the operations a user could perform have been limited by the addressing mechanisms. In addition, prior processor systems have operated as fixed word length machines, further limiting user operations.

Prior data processing systems have not provided effective protection mechanisms preventing one user from effecting another user's data and programs without permission. Such protection mechanisms have not allowed unique, positive identification of users requesting access to information, or of information, nor have such mechanisms been sufficiently flexible in operation. In addition, access rights have pertained to the users rather than to the information, so that control of access rights has been difficult. Finally, prior art protection mechanisms have allowed the use of "Trojan Horse arguments". That is, users not having access rights to certain information have been able to gain access to that information through another user or procedure having such access rights.

Yet another problem of the prior art is that of providing a simple and flexible user's interface to a data processing system. The character of user's interface to a data processing system is determined, in part, by the means by which a user refers to and identifies operands and procedures of the user's programs and by the instruction structure of the system. Operands and procedures are customarily referred to and identified by some form of logical address having points of reference, and validity, only within a user's program. These addresses must be translated into logical and physical addresses within a data processing system each time a program is executed, and must then be frequently retranslated or generated during execution of a program. In addition, a user must provide specific instructions as to data format and handling. As such reference to operands or procedures typically comprise a major portion of the instruction stream of the user's program and requires numerous machine translations and operations to implement. A user's interface to a conventional system is thereby complicated, and the speed of execution of programs reduced, because of the complexity of the program references to operands and procedures.

A data processing system's instruction structure includes both the instructions for controlling system operations and the means by which these instructions are executed. Conventional data processing systems are designed to efficiently execute instructions in one or two user languages, for example, FORTRAN or COBOL. Programs written in any other language are not efficiently executable. In addition, a user is often faced with difficult programming problems when using any high level language other than the particular one or two languages that a particular conventional system is designed to utilize.

Yet another problem in conventional data processing systems is that of protecting the system's internal mechanisms, for example, stack mechanisms and internal control mechanisms, from accidental or malicious interference by a user.

Finally, the internal structure and operation of prior art data processing systems have not been flexible, or adaptive, in structure and operation. That is, the internal structure and operation of prior systems have not allowed the systems to be easily modified or adapted to meet particular data processing requirements. Such modifications may include changes in internal memory capacity, such as the addition or deletion of special purpose subsystems, for example, floating point or array processors. In addition, such modifications have significantly effected the users interface with the system. Ideally, the actual physical structure and operation of the data processing system should not be apparent at the user interface.

It has already been proposed (IBM Technical Disclosure Bulletin Vol. 22 No. 3 Aug. 1979 pp 1286—1289) to maintain such a large address space that every object which is ever created can have a unique identifier. This requires a very large identifier field, e.g. 40 to 50 bits.

The object of the present invention is to implement such a concept so that it may be applied across many computers geographically distributed and without requiring all computers to use the same

Petitioner Apple Inc. - Exhibit 1004, p. 5171

programming language.

The system according to the invention is defined in the appended claims.

It is known in virtual address machines to use name tables to provide logical addresses for translation to physical addresses (Klar, Wichmann, "Mikroprogrammierung" June 1975, especially pp. 159—163, 176—179, 185—187, 195—205, 214—215) and this reference also discusses the emulation in software of different target machines.

More specifically, the embodiment of the invention described in detail below provides a data processing system suitable for use in interconnected data processing networks, which internal structure is flexible, protected from users, effectively invisible to users, and provides a flexible and simplified interface to users. The data processing system provides an addressing mechanism allowing permanent and unique identification of all information generated for use in or by operation of the system, and an extremely large address space which is accessible to and common to all such data processing systems. The addressing mechanism provides addresses which are independent of the physical configuration of the system and allow information to be completely identified, with a single address, to the bit granular level and with regard to information type or format. The present invention further provides a protection mechanism wherein variable access rights are associated with individual bodies of information. Information, and users requesting access to information, are uniquely identified through the system addressing mechanism. The protection mechanism also prevents use of Trojan Horse arguments. And, the present invention provides an instruction structure wherein high level user language instructions are transformed into dialect coded, uniform, intermediate level instructions to provide equal facility of execution for a plurality of user languages. Another feature is the provision of an operand reference mechanism wherein operands are referred to in user's programs by uniform format names which are transformed, by an internal mechanism transparent to the user, into addresses. The present invention additionally provides multilevel control and stack mechanisms protecting the system's internal mechanism from interference by users. Yet another feature is a data processing system having a flexible internal structure capable of performing multiple, concurrent operations and comprised of a plurality of separate, independent processors. Each such independent processor has a separate microinstruction control and at least one separate and independent port to a central communications and memory node. The communications and memory node is also an independent processor having separate and independent microinstruction control. The memory processor is internally comprised of a plurality of independently operating, microinstruction controlled processors capable of performing multiple, concurrent memory and communications operations. The present invention also provides further data processing system structural and operational features for implementing the above features.

It is thus advantageous to incorporate the present invention into a data processing system because the present invention provides addressing mechanisms suitable for use in large interconnected data processing networks. Additionally, the present invention is advantageous in that it provides an information protection mechanism suitable for use in large, interconnected data processing networks. The present invention is further advantageous in that it provides a simplified, flexible, and more efficient interface to a data processing system. The present invention is yet further advantageous in that it provides a data processing system which is equally efficient with any user level language by providing a mechanism for referring to operands in user programs by uniform format names and instruction structure incorporating dialect coded, uniform format intermediate level instructions. Additionally, the present invention protects data processing system internal mechanisms from user interference by providing multilevel control and stack mechanisms. The present invention is yet further advantageous in providing a flexible internal system structure capable of performing multiple, concurrent operations, comprising a plurality of separate, independent processors, each having a separate microinstruction control and at least one separate and independent port to a central, independent communications and memory processor comprised of a plurality of independent processors capable of performing multiple, concurrent memory and communications operations.

Other advantages and features of the present invention will be understood by those of ordinary skill in the art, after referring to the following detailed description of the preferred embodiments and drawings wherein.

BRIEF DESCRIPTION OF DRAWINGS

Fig. 1 is a partial block diagram of a computer system incorporating the present invention;

Fig. 2 is a diagram illustrating computer system addressing structure of the present invention;

Fig. 3 is a diagram illustrating the computer system instruction stream of the present invention;

Fig. 4 is a diagram illustrating the control structure of a conventional computer system;

Fig. 4A is a diagram illustrating the control structure of a computer system incorporating the present invention;

Fig. 5 — Fig. A1 inclusive are diagrams all relating to the present invention;

Fig. 5 is a diagram illustrating a stack mechanism;

Fig. 6 is a diagram illustrating procedures, procedure objects, processes, and virtual processors;

Fig. 7 is a diagram illustrating operating levels and mechanisms of the present computer;

Fig. 8 is a diagram illustrating a physical implementation of processes and virtual processors;

Petitioner Apple Inc. - Exhibit 1004, p. 5172

Fig. 9 is a diagram illustrating a process and process stack objects;
Fig. 10 is a diagram illustrating operation of macrostacks and secure stacks;
Fig. 11 is a diagram illustrating detailed structure of a stack;
Fig. 12 is a diagram illustrating a physical descriptor;
Fig. 13 is a diagram illustrating the relationship between logical pages and frames in a memory storage space;
Fig. 14 is a diagram illustrating access control to objects;
Fig. 15 is a diagram illustrating virtual processors and virtual processor swapping;
Fig. 16 is a partial block diagram of an I/0 system of the present computer system;
Fig. 17 is a diagram illustrating operation of a ring grant generator;
Fig. 18 is a partial block diagram of a memory system;
Fig. 19 is a partial block diagram of a fetch unit of the present computer system;
Fig. 20 is a partial block diagram of an execute unit of the present computer system;
Fig. 101 is a more detailed partial block diagram of the present computer system;
Fig. 102 is a diagram illustrating certain information structures and mechanisms of the present computer system;
Fig. 103 is a diagram illustrating process structures;
Fig. 104 is a diagram illustrating a macrostack structure;
Fig. 105 is a diagram illustrating a secure stack structure;
Figs. 106 A, B, and C are diagrams illustrating the addressing structure of the present computer system;
Fig. 107 is a diagram illustrating addressing mechanisms of the present computer system;
Fig. 108 is a diagram illustrating a name table entry;
Fig. 109 is a diagram illustrating protection mechanisms of the present computer system;
Fig. 110 is a diagram illustrating instruction and microinstruction mechanism of the present computer system;
Fig. 201 is a detailed block diagram of a memory system;
Fig. 202 is a detailed block diagram of a fetch unit;
Fig. 203 is a detailed block diagram of an execute unit;
Fig. 204 is a detailed block diagram of an I/O system;
Fig. 205 is a partial block diagram of a diagnostic processor system;
Fig. 206 is a diagram illustrating assembly of Figs. 201—205 to form a detailed block diagram of the present computer system;
Fig. 207 is a detailed block diagram of a memory interface controller;
Fig. 209 is a diagram of a memory to I/O system port interface;
Fig. 210 is a diagram of a memory operand port interface;
Fig. 211 is a diagram of a memory instruction port interface;
Fig. 230 is a detailed block diagram of memory field interface unit logic;
Fig. 231 is a diagram illustrating memory format manipulation operations;
Fig. 238 is a detailed block diagram of fetch unit offset multiplexer;
Fig. 239 is a detailed block diagram of fetch unit bias logic;
Fig. 240 is a detailed block diagram of a generalized four way, set associative cache representing name cache, protection cache, and address translation unit;
Fig. 241 is a detailed block diagram of portions of computer system instruction and microinstruction control logic;
Fig. 242 is a detailed block diagram of portions of computer system microinstruction control logic;
Fig. 243 is a detailed block diagram of further portions of computer system microinstruction control logic;
Fig. 244 is a diagram illustrating computer system states of operation;
Fig. 245 is a diagram illustrating computer system states of operation for a trace trap request;
Fig. 246 is a diagram illustrating computer system states of operation for a memory repeat interrupt;
Fig. 247 is a diagram illustrating priority level and masking of computer system events;
Fig. 248 is a detailed block diagram of event logic.
Fig. 249 is a detailed block diagram of microinstruction control store logic;
Fig. 251 is a diagram illustrating a return control word stack word;
Fig. 252 is a diagram illustrating machine control words;
Fig. 253 is a detailed block diagram of a register address generator;
Fig. 254 is a block diagram of interval and egg timers;
Fig. 255 is a detailed block diagram of execute unit control logic;
Fig. 257 is a detailed block diagram of execute unit multiplier data paths and memory;
Fig. 260 is a diagram illustrating operation of an execute unit command queue load and interface to a fetch unit;
Fig. 261 is a diagram illustrating operation of an execute unit operand buffer load and interface to a fetch unit;
Fig. 262 is a diagram illustrating operation of an execute unit storeback or transfer of results and

Petitioner Apple Inc. - Exhibit 1004, p. 5173

interface to a fetch unit;

Fig. 263 is a diagram illustrating operation of an execute unit check test condition and interface to a fetch unit;

Fig. 264 is a diagram illustrating operation of an execute unit exception test and interface to a fetch unit;

Fig. 265 is a block diagram of an execute unit arithmetic operation stack mechanism;

Fig. 266 is a diagram illustrating execute unit and fetch unit interrupt handshaking and interface;

Fig. 267 is a diagram illustrating execute unit and fetch unit interface and operation for nested interrupts;

Fig. 268 is a diagram illustrating execute unit and fetch unit interface and operation for loading an execute unit control store;

Fig. 269 is a detailed block diagram and illustration of operation of an I/O system ring grant generator;

Fig. 270 is a detailed block diagram of a fetch unit micromachine of the present computer system;

Fig. 271 is a diagram illustrating a logical descriptor;

Fig. 272 is a diagram illustrating use of fetch unit stack registers;

Fig. 273 is a diagram ilustrating structures controlling event invocations;

Fig. 301 is a diagram illustrating pointer formats;

Fig. 302 is a diegram illustrating an associated address table;

Fig. 303 is a diagram illustrating a namespace overview of a procedure object;

Fig. 304 is a diagram illustrating name table entries;

Fig. 305 is a diagram illustrating an example of name resolution;

Fig. 306 is a diagram illustrating name cache entries;

Fig. 307 is a diagram illustrating translation of S-interpreter universal identifiers to dialect numbers;

Fig. 401 is a diagram illustrating operating systems and system resources;

Fig. 402 is a diagram illustrating multiprocess operating systems;

Fig. 403 is a diagram illustrating an extended operating system and a kernel operating system;

Fig. 404 is a diagram illustrating an EOS view of objects;

Fig. 405 is a diagram illustrating pathnames to universal identifier translation;

Fig. 406 is a diagram illustrating universal identifier detail;

Fig. 407 is a diagram illustrating address translation with an address translation unit, a memory hash table, and a memory;

Fig. 408 is a diagram illustrating hashing in an active subject table;

Fig. 409 is a diagram illustrating logical allocation units and objects;

Fig. 410 is a diagram illustrating an active logical allocation unit table and active allocation units;

Fig. 411 is a diagram illustrating a conceptual logical allocation unit directory structure;

Fig. 412 is a diagram illustrating detail of a logical allocation unit directory entry;

Fig. 413 is a diagram illustrating universal identifiers and active object numbers;

Fig. 416 is a diagram illustrating subject templates, primitive access control list entries, and extended access control list entries;

Fig. 421 is a diagram illustrating an active primitive access matrix and an active primitive access matrix entry;

Fig. 422 is a diagram illustrating primitive data access checking;

Fig. 448 is a diagram illustrating event counters and await entries;

Fig. 449 is a diagram illustrating an await table overview;

Fig. 453 is a diagram illustrating an overview of a virtual processor;

Fig. 454 is a diagram illustrating virtual processor synchronization;

Fig. 467 is a diagram illustrating an overview of a macrostack object;

Fig. 468 is a diagram illustrating details of a macrostack object base;

Fig. 469 is a diagram illustrating details of a macrostack frame;

Fig. 470 is a diagram illustrating an overview of a secure stack;

Fig. 471 is a diagram illustrating details of a secure stack frame; and,

Fig. 472 is a diagram illustrating an overview of procedure object.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following description presents the structure and operation of a computer system incorporating a presently preferred embodiment of the present invention. As indicated in the following Table of Contents, certain features of computer system structure and operation will first be described in an Introductory Overview. Next, these and other features will be described in further detail in a more detailed Introduction to the detailed descriptions of the computer system. Following the Introduction, the structure and operation of the computer system will be described in detail. The detailed descriptions will present descriptions of the structure and operation of each of the major subsystems, or elements, of the computer system, of the interfaces between these major subsystems, and of overall computer system operation. Next, certain features of the operation of the individual subsystems will be presented in further detail.

Certain conventions are used throughout the following descriptions to enhance clarity of presentation. First, and with exception of the Introductory Overview, each figure referred to in the following descriptions

Petitioner Apple Inc. - Exhibit 1004, p. 5174

will be referred to by a three digit number. The most significant digit represents the number of the chapter in the following descriptions in which a particular figure is first referred to. The two least significant digits represent the sequential number of appearance of a figure in a particular chapter. For example, Figure 319 would be the nineteenth figure appearing in the third chapter. Figures appearing in the Introductory Overview are referred to by a one or two digit number representing the order in which they are referred to in the Introductory Overview. It should be noted that certain figure numbers, for example, Figure 208, do not appear in the following figures and descriptions; the subject matter of these figures has been incorporated into other figures and these figures deleted, during drafting of the following descriptions, to enhance clarity of presentation.

Second, reference numerals comprise a two digit number (00—99) preceded by the number of the figure in which the corresponding elements first appear. For example, reference numerals 31901 to 31999 would refer to elements 1 through 99 appearing in Fig. 319.

Finally, interconnections between related circuitry is represented in two ways. First, to enhance clarity of presentation, interconnections between circuitry may be represented by common signal names or references, rather than by drawn representations of wires or buses. Second, where related circuitry is shown in two or more figures, the figures may share a common figure number and will be distinguished by a letter designation, for example, Figs. 319, 319A, and 319B. Common electrical points between such circuitry may be indicated by a bracket enclosing a lead to such a point and a designation of the form "A—b". "A" indicates other figures having the same common point for example, 319A, and "b" designates the particular common electrical point. In cases of related circuitry shown in this manner in two or more figures, reference numerals to elements will be assigned in sequence through the group of figures; the figure number portion of such reference numerals will be that of the first figure of the group of figures.

INTRODUCTORY OVERVIEW

A. Hardware Overview (Fig. 1)
B. Individual Operating Features (Figs. 2, 3, 4, 5, 6)
    1. Addressing (Fig. 2)
    2. S-Language Instructions and Namespace Addressing (Fig. 3)
    3. Architectural Base Pointer Addressing
    4. Stack Mechanisms (Figs. 4-5)
C. Procedure Processes and Virtual Processors (Fig. 6)
D. CS 101 Overall Structure and Operation (Figs. 7, 8, 9, 10, 11, 12, 13, 14, 15)
    1. Introduction (Fig. 7)
    2. Compilers 702 (Fig. 7)
    3. Binder 703 (Fig. 7)
    4. EOS 704 (Fig. 7)
    5. KOS and Architectural Interface 708 (Fig. 7)
    6. Processes 610 and Virtual Processors 612 (Fig. 8)
    7. Processes 610 and Stacks (Fig. 9)
    8. Processes 610 and Calls (Figs. 10, 11)
    9. Memory References and the Virtual Memory Management System (Fig. 12, 13)
    10. Access Control (Fig. 14)
    11. Virtual Processors and Virtual Processor Swapping (Fig. 15)
E. CS 101 Structural Implementation (Figs. 16, 17, 18, 19, 20)
    1. (IOS) 116 (Figs. 16, 17)
    2. Memory (MEM) 112 (Fig. 18)
    3. Fetch Unit (FU) 120 (Fig. 19)
    4. Execute Unit (EU) 122 (Fig. 20)

1. Introduction (Figs. 101—110)
A. General Structure and Operation (Fig. 101)
    a. General Structure
    b. General Operation
    c. Definition of Certain Terms
    d. Multi-program Operation
    e. Multi-Language Operation
    f. Addressing Structure
    g. Protection Mechanism
B. Computer System 10110 Information Structure and Mechanisms (Figs. 102, 103, 104, 105)
    a. Introduction (Fig. 102)
    b. Process Structures 10210 (Figs. 103, 104, 105)
        1. Procedure Objects (Fig. 103)
        2. Stack Mechanisms (Figs. 104, 105)
        3. FURSM 10214 (Fig. 103)
C. Virtual Processor State Blocks and Virtual Process Creation (Fig. 102)

Petitioner Apple Inc. - Exhibit 1004, p. 5175

D. Addressing Structures 10220 (Figs. 103, 106, 107, 108)
1. Objects, UID's, AON's, Names, and Physical Addresses (Fig. 106)
2. Addressing Mechanisms 10220 (Fig. 107)
3. Name Resolution (Figs. 103, 108)
4. Evaluation of AON Addresses to Physical Addresses (Fig. 107)
E. CS 10110 Protection Mechanisms (Fig. 109)
F. CS 10110 Micro-Instruction Mechanisms (Fig. 110)
G. Summary of Certain CS 10110 Features and Alternate Embodiments.

2. Detailed Description of CS 10110 Major Subsystems Figs. 201—206, 207—274
A. MEM 10110 (Figs. 201, 206, 207-237)
a. Terminology
b. MEM 10112 physical Structure (Fig. 201)
c. MEM 10112 General Operation
d. MEM 10112 Port Structure
1. IO Port Characteristics
2. JO Port Characteristics
3. JI Port Characteristics
e. MEM 10112 Control Structure and Operation (Fig. 207)
1. MEM 10112 Control Structure
2. MEM 10112 Control Operation
f. MEM 10112 Operations
g. MEM 10112 Interfaces to JP 10114 and IOS 10116 (Figs. 209, 210, 211, 204)
1. IO Port 20910 Operating Characteristics (Figs 209, 204)
2. JO Port 21010 Operating Characteristics (Fig. 210)
3. JI Port 21110 Operating Characteristics (Fig. 211)
h. FIU 20120 (Figs. 201, 230, 231)
B. Fetch Unit 10120 (Figs. 202, 206, 101, 103, 104, 238)
1. Descriptor Processor 20210 (Figs. 202, 101, 103, 104, 238, 239).
a. Offset Processor 20218 Structure
b. AON Processor 20216 Structure
c. Length Processor 20220 Structure
d. Descriptor Processor 20218 Operation
a.a. Offset Selector 20238
b.b. Offset Multiplexer 20240 Detailed Structure (Fig. 238)
c.c. Offset Multiplexer 20240 Detailed Operation
aaa. Internal Operation
bbb. Operation Relative to DESP 20210
e. Length Processor 20220 (Fig. 239)
a.a. Length ALU 20252
b.b. BIAS 20246 (Fig. 239)
f. AON Processor 20216
a.a. AONGRF 20232
b.b. AON Selector 20248
2. Memory Interface 20212 (Figs. 106, 240)
a.a. Descriptor Trap 20256 and Data Trap 20258
b.b. Name Cache 10226, Address Translation Unit 10228, and Protection Cache 10234 (Fig. 106)
c.c. Structure and Operation of Generalized Cache and NC 10226 (Fig. 240)
d.d. ATU 10228 and PC 10234
3. Fetch Unit Control Logic 20214 (Fig. 202)
a.a. Fetch Unit Control Logic 20214 Overall Structure
b.b. Fetch Unit Control Logic 20214 Operation
a.a.a. Prefetcher 20264, Instruction Buffer 20262, Parser 20264, Operation Code Register 20268, CPC 20270, IPC 20272, and EPC 20274 (Fig. 241)
b.b.b. Fetch Unit Dispatch Table 11010, Execute Unit Dispatch Table 20266, and Operation Code Register 20268 (Fig. 242)
c.c.c. Next Address Generator 24310 (Fig. 243)
cc. FUCTL 20214 Circuitry for CS 10110 Internal Mechanisms (Figs. 244-250)
a.a.a. State Logic 20294 (Figs. 244A—244Z)
b.b.b. Event Logic 20284 (Figs. 245, 246, 247, 248)
c.c.c. Fetch Unit S-Interpreter Table 11012 (Fig. 249)
d.d. CS 10110 Internal Mechanism Control
a.a.a. Return Control Word Stack 10358 (Fig. 251)
b.b.b. Machine Control Block (Fig. 252)
c.c.c. Register Address Generator 20288 (Fig. 253)

Petitioner Apple Inc. - Exhibit 1004, p. 5176

d.d.d. Timers 20296 (Fig. 254)
e.e.e. Fetch Unit 10120 Interface to Execute Unit 10122
C. Execute Unit 10122 (Figs. 203, 255-268)
a. General Structure of Execute Unit 10122
1. Execute Unit I/O 20312
2. Execute Unit Control Logic 20310
3. Multiplier Logic 20314
4. Exponent Logic 20316
5. Multiplier Control 20318
6. Test and Interface Logic 20320
b. Execute Unit 10122 Operation (Fig. 255)
1. Execute Unit Control Logic 20310 (Fig. 255)
a.a. Command Queue 20342
b.b. Command Queue Event Control Store 25514 and Command Queue Event Address Control Store 25516
c.c. Execute Unit S-Interpreter Table 20344
d.d. Microcode Control Decode Register 20346
e.e. Next Address Generator 20340
2. Operand Buffer 20322 (Fig. 256)
3. Multiplier 20314 (Figs. 257, 258)
a.a. Multiplier 20314 I/0 Data Paths and Memory (Fig. 257)
a.a.a. Container Size Check
b.b.b. Final Result Output Multiplexer 20324
4. Test and Interface Logic 20320 (Figs. 260-268)
a.a. FU 10120/EU 10122 Interface
a.a.a. Loading of Command Queue 20342 (Fig. 260)
b.b.b. Loading of Operand Buffer 20320 (Fig. 261)
c.c.c. Storeback (Fig. 262)
d.d.d. Test Conditions (Fig. 263)
e.e.e. Exception Checking (Fig. 264)
f.f.f. Idle Routine
g.g.g. EU 10122 Stack Mechanisms (Figs. 265, 266, 267)
h.h.h. Loading of Execute Unit S-Interpreter Table 20344 (Fig. 268)
D. I/O System 10116 (Figs. 204, 206, 269)
a. I/O System 10116 Structure (Fig. 204)
b. I/O System 10116 Operation (Fig. 269)
1. Data Channel Devices
2. I/O Control Processor 20412
3. Data Mover 20410 (Fig. 269)
a.a. Input Data Buffer 20440 and Output Data Buffer 20442
b.b. Priority Resolution and Control 20444 (Fig. 269)
E. Diagnostic Processor 10118 (Fig. 101, 205)
F. CS 10110 Micromachine Structure and Operation (Figs. 270—274)
a. Introduction
b. Overview of Devices Comprising FU Micromachine (Fig. 270)
1. Devices Used By Most Microcode
a.a. MOD Bus 10144, JPD Bus 10142, and DB Bus 27021
b.b. Microcode Addressing
c.c. Descriptor Processor 20218 (Fig. 271)
d.d. EU 10122 Interface
2. Specialized Micromachine Devices
a.a. Instruction Stream Reader 27001
b.b. SOP Decoder 27003
c.c. Name Translation Unit 27015
d.d. Memory Reference Unit 27017
e.e. Protection Unit 27019
f.f. KOS Micromachine Devices
c. Micromachine Stacks and Microroutine Calls and Returns (Figs. 272, 273)
1. Micromachine Stacks (Fig. 272)
2. Micromachine Invocations and Returns
3. Means of Invoking Microroutines
4. Occurrence of Event Invocations (Fig. 273)
d. Virtual Micromachines and Monitor Micromachine
1. Virtual Mode
2. Monitor Micromachine

Petitioner Apple Inc. - Exhibit 1004, p. 5177

e. Interrupt and Fault Handling
1. General Principles
2. Hardware Interrupt and Fault Handling in CS 10110
3. Monitor Mode: Differential Masking and Hardware Interrupt Handling
g. FU Micromachine and CS 10110 Subsystems

3. Namespace, S-Interpreters and Pointers (Figs. 301—307, 274)
A. Pointers and Pointer Resolution (Figs. 301, 302)
a. Pointer Formats (Fig. 301)
b. Pointers in FU 10120 (Fig. 302)
c. Descriptor to Pointer Conversion
B. Namespace and the S-Interpreters (Figs. 303—307)
a. Procedure Object 606 Overview (Fig. 303)
b. Namespace
1. Name Resolution and Evaluation
2. The Name Table (Fig. 304)
3. Architectural Base Pointers (Figs. 305, 306)
a.a. Resolving and Evaluating Names (Fig. 305)
b.b. Implementation of Name Evaluation and Name Resolve in CS 10110
c.c. Name Cache 10226 Entries (Fig. 306)
d.d. Name Cache 10226 Hits
e.e. Name Cache 10226 Misses
f.f. Flushing Name Cache 10226
g.g. Fetching the Instruction Stream
h.h. Parsing the Instruction Stream
c. The S-Interpreters (Fig. 307)
1. Translating SIP Into a Dialect Number (Fig. 307)
2. Dispatching

4. The Kernel Operation System
A. Introduction
a. Operating Systems (Fig. 401)
1. Resources Controlled by Operating Systems (Fig. 402)
b. The Operating System in CS 10110
c. Extended Operating System and the Kernel Operating System (Fig. 403)
B. Objects and Object Management (Fig. 404)
a. Objects and User Programs (Fig. 405)
b. UIDs 40401 (Fig. 406)
c. Object Attributes
d. Attributes and Access Control
e. Implementation of Objects
1. Introduction (Figs 407, 408)
2. Objects in Secondary Storage 10124 (Figs. 409, 410).
a.a. Representation of an Object's Contents on Secondary Storage 10124
b.b. LAUD 40903 (Figs. 411, 412)
3. Active Objects (Fig. 413)
a.a. UID 40401 to AON 41304 Translation
C. The Access Control System
a. Subjects
b. Domains
c. Access Control Lists
1. Subject Templates (Fig. 416)
2. Primitive Access Control Lists (PACLs)
3. APAM 10918 and Protection Cache 10234 (Fig. 421)
4. Protection Cache 10234 and Protection Checking (Fig. 422)
D. Processes
1. Synchronization of Processes 610 and Virtual Processors 612
a. Event Counters 44801, Await Entries 44804, and Await Tables (Fig. 448, 449)
b. Synchronization with Event Counters 44801 and Await Entries 44804
E. Virtual processors 612 (Fig. 453)
a. Virtual Processor Management (Fig. 453)
b. Virtual Processors 612 and Synchronization (Fig. 454)
F. Process 610 Stack Manipulation
1. Introduction to Call and Return
2. Macrostacks (MAS) 502 (Fig. 467)

Petitioner Apple Inc. - Exhibit 1004, p. 5178

a.a. MAS Base 10410 (Fig. 468)
b.b. Per-domain Data Area 46853 (Fig. 468)
c.c. MAS Frame 46709 Detail (Fig. 469)
3. SS 504 (Fig. 470)
a.a. SS Base 47001 (Fig. 471)
b.b. SS Frames 47003 (Fig. 471)
a.a.a. Ordinary SS Frame Headers 10514 (Fig. 471)
b.b.b. Detailed Structure of Macrostate 10516 (Fig. 471)
c.c.c. Cross-domain SS Frames 47039 (Fig. 471)
4. Portions of Procedure Object 608 Relevant to Call and Return (Fig. 472)
5. Execution of Mediated Calls
a.a. Mediated Call SINs
b.b. Simple Mediated Calls (Figs. 270, 468, 469, 470, 471, 472)
c.c. Invocations of Procedures 602 Requiring SEBs 46864 (Figs. 270, 468, 469, 470, 471, 472)
d.d. Cross-Procedure Object Calls (Figs. 270, 468, 469, 470, 471, 472)
e.e. Cross-Domain Calls (Figs. 270, 408, 418, 468, 469, 470, 471, 472)
f.f. Failed Cross-Domain Calls (Figs. 270, 468, 469, 470, 471, 472)
6. Neighborhood Calls (Figs. 468, 469, 472)

INTRODUCTORY OVERVIEW

The following overview will first briefly describe the overall physical structure and operation of a presently preferred embodiment of a digital computer system incorporating the present invention. Then certain operating features of that computer system will be individually described. Next, overall operation of the computer system will be described in terms of those individual features.

A. Hardware Overview (Fig. 1)

Referring to Fig. 1, a block diagram of Computer System (CS) 101 incorporating the present invention is shown. Major elements of CS 101 are I/O System (IOS) 116, Memory (MEM) 112, and Job Processor (JP) 114. JP 114 is comprised of a Fetch Unit (FU) 120 and an Execute Unit (EU) 122. CS 101 may also include a Diagnostic Processor (DP), not shown or described in the instant description.

Referring first to IOS 116, a primary function of IOS 116 is control of transfer of information between MEM 112 and the outside world. Information is transferred from MEM 112 to IOS 116 through IOM Bus 130, and from IOS 116 to MEM 112 through MIO Bus 129. IOMC Bus 131 is comprised of bi-directional control signals coordinating operation of MEM 112 and IOS 116. IOS 116 also has an interface to FU 120 through IOJP Bus 132. IOJP Bus 132 is a bi-directional control bus comprised essentially of two interrupt lines. These interrupt lines allow FU 120 to indicate to IOS 116 that a request for information by FU 120 has been placed in MEM 112, and allows IOS 116 to inform FU 120 that information requested by FU 120 has been transferred into a location in MEM 112. MEM 112 is CS 101's main memory and serves as the path for information transfer between the outside world and JP 114. MEM 112 provides instructions and data to FU 120 and EU 122 through Memory Output Data (MOD) Bus 140 and receives information from FU 120 and EU 122 through Job Processor Data (JPD) Bus 142. FU 120 submits read and write requests to MEM 112 through Physical Descriptor (PD) Bus 146.

JP 114 is CS 101's CPU and, as described above, is comprised of FU 120 and EU 122. A primary function of FU 120 is executing operations of user's programs. As part of this function, FU 120 controls transfer of instructions and data from MEM 112 and transfer of results of JP 114 operations back to MEM 112. FU 120 also performs operating system type functions, and is capable of operating as a complete, general purpose CPU. EU 122 is primarily an arithmetic and logic unit provided to relieve FU 120 of certain arithmetic operations. FU 120, however, is capable of performing EU 122 operations. In alternate embodiments of CS 101, EU 122 may be provided only as an option for users having particular arithmetic requirements. Coordination of FU 120 and EU 122 operations is accomplished through FU/EU (FUEU) Bus 148, which includes bi-directional control signals and mutual interrupt lines. As described further below, both FU 120 and EU 122 contain register file arrays referred to respectively as CRF and ERF, in addition to registers associated with, for example, ALUs.

A primary feature of CS 101 is that IOS 116, MEM 112, FU 120 and EU 122 each contain separate and independent microinstruction control, so that IOS 116, MEM 112, and EU 122 operate asynchronously under the general control of FU 120. EU 122, for example, may execute a complex arithmetic operation upon receipt of data and a single, initial command from FU 120.

Having briefly described the overall structure and operation of CS 101, certain features of CS 101 will be individually further described next below.

B. Individual Operating Features (Figs. 2, 3, 4, 5, 6)

1. Addressing (Fig. 2)

Referring to Fig. 2, a diagramic representation of portions of CS 101's addressing structure is shown. CS 101's addressing structure is based upon the concept of Objects. An Object may be regarded as a

Petitioner Apple Inc. - Exhibit 1004, p. 5179

container for holding a particular type of information. For example, one type of Object may contain data while another type of Object may contain instructions or procedures, such as a user program. Still another type of Object may contain microcode. In general, a particular Object may contain only one type or class of information. An Object may, for example, contain up to 232 bits of information, but the actual size of a particular Object is flexible. That is, the actual size of a particular Object will increase as information is written into that Object and will decrease as information is taken from that Object. In general, information in Objects is stored sequentially, that is without gaps.

Each Object which can ever exist in any CS 101 system is uniquely identified by a serial number referred to as a Unique Identifier (UID). A UID is a 128 bit value comprised of a serial number dependent upon, for example, the particular CS 101 system and user, and a time code indicating time of creation of that Object. UIDs are permanently assigned to Objects, no two Objects may have the same UID, and UIDs may not be reused. UIDs provide an addressing base common to all CS 101 systems which may ever exist, through which any Object ever created may be permanently and uniquely identified.

As described above, UIDs are 128 bit values and are thus larger than may be conveniently handled in present embodiments of CS 101. In each CS 101, therefore, those Objects which are active (currently being used) in that system are assigned 14 bit Active Object Numbers (AONs). Each Object active in that system will have a unique AON. Unlike UIDs, AONs are only temporarily assigned to particular Objects. AONs are valid only within a particular CS 101 and are not unique between systems. An Object need not physically reside in a system to be assigned an AON, but can be active in that system only if it has been assigned an AON.

A particular bit within a particular Object may be identified by means of a UID address or an AON address. In CS 101, AONs and AON addresses are valid only within JP 114 while UIDs and UID addresses are used in MEM 112 and elsewhere. UID and AON addresses are formed by appending a 32 bit Offset (O) field to that Object's UID or AON. O fields indicate offset, or location, of a particular bit relative to the start of a particular Object.

Segments of information (sequences of information bits) within particular Objects may be identified by means of descriptors. A UID descriptor is formed by appending a 32 bit Length (L) field of a UID address. An AON, or logical descriptor is formed by appending a 32 bit L field to an AON address. L fields identify length of a segment of information bits within an Object, starting from the information bit identified by the UID or AON address. In addition to length information, UID and logical descriptors also contain Type fields containing information regarding certain characteristics of the information in the information segment. Again, AON based descriptors are used within JP 114, while UID based descriptors are used in MEM 112.

Referring to Figs. 1 and 2 together, translation between UID addresses and descriptors and AON addresses and descriptors is performed at the interface between MEM 112 and JP 114. That is, addresses and descriptors within JP 114 are in AON form while addresses and descriptors in MEM 112, IOS 116, and the external world are in UID form. In other embodiments of CS 101 using AONs, transformation from UID to AON addressing may occur at other interfaces, for example at the IOS 116 to MEM 112 interface, or at the IOS 116 to external world interface. Other embodiments of CS 101 may use UIDs throughout, that is not use AONs even in JP 114.

Finally, information within MEM 112 is located through MEM 112 Physical Addresses identifying particular physical locations within MEM 112's memory space. Both IOS 116 and JP 114 address information within MEM 112 by providing physical addresses to MEM 112. In the case of physical addresses provided by JP 114, these addresses are referred to as Physical Descriptors (PDs). As described below, JP 114 contains circuitry to translate logical descriptors into physical descriptors.

2. S-Language Instructions and Namespace Addressing (Fig. 3)

CS 101 is both an S-Language machine and a Namespace machine. That is, operations to be executed by CS 101 are expressed as S-Language Operations (SOPs) while operands are identified by Names. SOPs are of a lower, more detailed, level than user language instructions, for example FORTRAN and COBOL, but of a higher level than conventional machine language instructions. SOPs are specific to particular user languages rather than a particular embodiment of CS 101, while conventional machine language instructions are specific to particular machines. SOPs are in turn interpreted and executed by microcode. There will be an S-Language Dialect, a set of SOPs, for each user languages. CS 101, for example, may have SOP Dialects for COBOL, FORTRAN, and SPL. A particular distinction of CS 101 is that all SOPs are of a uniform, fixed length, for example 16 bits. CS 101 may generally contain one or more sets of microcode for each S-Language Dialect. These microcode Dialect Sets may be completely distinct, or may overlap where more than one SOP utilizes the same microcode.

As stated above, in CS 101 all operands are identified by Names, which are 8, 12, or 16 bit numbers. CS 101 includes one or more "Name Tables" containing an Entry for each operand Name appearing in programs currently being executed. Each Name Table Entry contains information describing the operand referred to by a particular Name, and the directions necessary for CS 101 to translate that information into a corresponding logical descriptor. As previously described, logical descriptors may then be transformed into physical descriptors to read and write operands from or to MEM 112. As described above, UIDs are unique for all CS 101 systems and AONs are unique within individual CS 101 systems. Names, however, are unique only within the context of a user's program. That is, a particular Name may appear in two different

Petitioner Apple Inc. - Exhibit 1004, p. 5180

user's programs and, within each program, will have different Name Table Entries and will refer to different operands.

CS 101 may thereby be considered as utilizing two sets of instructions. A first set is comprised of SOPs, that is instructions selecting algorithms to be executed. The second set of instructions are comprised of Names, which may be regarded as entry points into tables of instructions for making references regarding operands.

Referring to Fig. 3, a diagramic representation of CS 101 instruction stream is shown. A typical SIN is comprised of an SOP and may include one or more Names referring to operands. SOPs and Names allow user's programs to be expressed in very compact code. Fewer SOPs than machine language instructions are required to express a user's program. Also, use of SOPs allows easier and simpler construction of compilers, and facilitates adaption of CS 101 systems to new user languages. In addition, use of Names to refer to operands means that SOPs are independent of the form of the operands upon which they operate. This in turn allows for more compact code in expressing user programs in that SOPs specifying operations dependent upon operand form are not required.

3. Architectural Base Pointer Addressing

As will be described further below, a user's program residing in CS 101 will include one or more Objects. First, a Procedure Object contains at least the SINs of the user's programs and a Name Table containing entries for operand Names of the program. The SINs may include references, or calls, to other Procedure Objects containing, for example, procedures available in common to many users. Second, a Static Data Area may contain static data, that is data having an existence for at least a single execution of the program. And third, a Macro-stack, described below, may contain local data, that is data generated during execution of a program. Each Procedure Object, the Static Data Area and the Macro-stack are individual Objects identified by UIDs and AONs and addressable through UID and AON addresses an descriptors.

Locations of information within a user's Procedure Objects, Static Data Area, and Macro-stack are expressed as offsets from one of three values, or base addresses, referred to as Architectural Base Pointers (ABPs). For example, location information in Name Tables is expressed as offsets from one of the ABPs. ABPs may be expressed as previously described.

The three ABPs are the Frame Pointer (FP), the Procedure Base Pointer (PBP), and the Static Data Pointer (SDP). Locations of data local to a procedure, for example in the procedure's Macrostack, are described as offsets from FP. Locations of non-local data, that is Static Data, are described as offsets from SDP. Locations of SINs in Procedure Objects are expressed as offsets from PBP; these offsets are determined as a Program Counter (PC) value. Values of the ABPs vary during program execution and are therefore not provided by the compiler converting a user's high level language program into a program to be executed in a CS 101 system. When the program is executed, CS 101 provides the proper values for the ABPs. When a program is actually being executed, the ABP's values are stored in FU 120's GRF.

Other pointers are used, for example, to identify the top frame of CS 101's Secure Stack (a microcode level stack described below) or to identify the microcode Dialect currently being used in execute the SINs of a procedure. These pointers are similar to FP, SDP, and PBP.

4. Stack Mechanisms (Fig. 4—5)

Referring to Fig. 4 and 4A, diagramic representations of various control levels and stack mechanisms of, respectively, conventional machines and CS 101, are shown. Referring first to Fig. 4, top level of control is provided by User Language Instructions 402, for example in FORTRAN or COBOL. User Language Instructions 402 are converted into a greater number of more detailed Machine Language Instructions 404, used within a machine to execute user's programs. Within the machine, Machine Language Instructions 404 are interpreted and executed by Microcode Instructions 406, that is sequences of microinstructions which in turn directly control Machine Hardware 408. Some conventional machines may include a Stack Mechanism 410 used to save current machine state, that is current microinstruction and contents of various machine registers, if a current Machine Language Instruction 404 cannot be executed or is interrupted. In general, machine state on the microcode and hardware level is not saved. Execution of a current Machine Language Instruction 404 is later resumed at start of the microinstruction sequence for executing that Machine Language Instruction 404.

Referring to Fig. 4A, top level control in CS 101 is by User Language Instructions 412 as in a conventional machine. In CS 101, however, User Language Instructions 412 are translated into SOPs 414 which are of a higher level than conventional machine language instructions. In general, a single User Language Instruction 412 is transformed into at most two or three SOPs 414, as opposed to an entire sequence of conventional Machine Language Instructions 404. SOPs 414 are interpreted and executed by Microcode Instructions 416 (sequences of microinstructions) which directly control CS 101 Hardware 418. CS 101 includes a Macro-stack Mechanism (MAS) 420, at SOPs 414 level, which is comparable to but different in construction and operation from a conventional Machine Language Stack Mechanism 410. CS 101 also includes Micro-code Stack Mechanisms 422 operating at Microcode 416 level, so that execution of an interrupted microinstruction of a microinstruction sequence may be later resumed with the particular microinstruction which was active at the time of the interrupt. CS 101 is therefore more efficient in handling

Petitioner Apple Inc. - Exhibit 1004, p. 5181

Interrupts in that execution of microinstruction sequences is resumed from the particular point that a microinstruction sequence was interrupted, rather from the beginning of that sequence. As will be described further below, CS 101's Micro-code Stack Mechanisms 422 on microcode level is effectively comprised of two stack mechanisms. The first stack is Micro-instruction Stack (MIS) 424 while the second stack is referred to as Monitor Stack (MOS) 426. CS 101 SIN Microcode 428 and MIS 424 are primarily concerned with execution of SOPs of user's programs. Monitor Microcode 430 and MOS 426 are concerned with operation of certain CS 101 internal functions.

Division of CS 101's microcode stacks into an MIS 424 and a MOS 426 illustrates a further feature of CS 101. In conventional machines, monitor functions may be performed by a separate CPU operating in conjunction with the machine's primary CPU. In CS 101, a single hardware CPU is used to perform both functions with actual execution of both functions performed by separate groups of microcode. Monitor microcode operations may be initiated either by certain SINs 414 or by control signals generated directly by CS 101's Hardware 418. Invocation of Monitor Microcode 430 by Hardware 418 generated signals insures that CS 101's monitor functions may always be invoked.

Referring to Fig. 5, a diagramic representation of CS 101,s stack mechanisms for *a single user's program, or procedure*, is shown. Basically, and with exception of MOS 426, CS 101's stacks reside in MEM 112 with certain portions of those stacks accelerated into FU 120 and EU 122 to enhance speed of operation.

Certain areas of MEM 112 storage space are set aside to contain Macro-Stacks (MASs) 502, stack mechanisms operating on the SINs level, as described above. Other areas of MEM 112 are set aside to contain Secure Stack (SS) 504, operating on the microcode level, as described above and of which MIS 424 is a part.

As described further below, both FU 120 and EU 122 contain register file arrays, referred to respectively as GRF and ERF, in addition to registers associated with, for example, ALUs. Referring to FU 120, shown therein is FU 120's GRF 506. GRF 506 is horizontally divided into three areas. A first area, referred to as General Registers (GRs) 508 may in general be used in the same manner as registers in a conventional machine. A second area of GRF 506 is Micro-Stack (MIS) 424, and is set aside to contain a portion of a Process's SS 504. A third portion of GRF 506 is set aside to contain MOS 426. Also indicated in FU 120 is a block referred to as Microcode Control State (mCS) 510. mCS 510 represents registers and other FU 120 hardware containing current operating state of FU 120 on the microinstruction and hardware level. mCS 510 may include, for example, the current microinstruction controlling operation of FU 120.

Referring to EU 122, indicated therein is a first block referred to as Execute Unit State (EUS) 512 and a second block referred to as SOP Stack 514. EUS 512 is similar to mCS 510 in FU 120 and includes all registers and other EU 122 hardware containing information reflecting EU 122's current operating state. SOP Stack 518 is a portion of EU 122's ERF 516 which has been set aside as a stack mechanism to contain a portion of a process's SS 504 pertaining to EU 122 operations.

Considering first MASs 502, as stated above MASs 502 operate generally upon the SINs level. MASs 502 are used in general to store current state of a process's (defined below) execution of a user's program.

Referring next to MIS 424, in a present embodiment of CS 101 that portion of GRF 506 set aside to contain MIS 424 may have a capacity of eight stack frames. That is, up to 8 microinstruction level interrupts or calls pertaining to execution of a user's program may be stacked within MIS 424. Information stored in MIS 424 stack frames is generally information from GR 508 and MCS 510. MIS 424 stack frames are transferred between MIS 424 and SS 504 such that at least one frame, and no more than 8 frames, of SS 504 reside in GRF 506. This insures that at least the top-most frames of a process's SS 504 are present in FU 120, thereby enhancing speed of operation of FU 120 by providing rapid access to those top frames. SS 504, residing in MEM 112, may contain, for all practical purposes, an unlimited number of frames so that MIS 424 and SS 504 appear to a user to be effectively an infinitely deep stack.

MOS 426 resides entirely in FU 120 and, in a present embodiment of CS 101, may have a capacity of 8 stack frames. A feature of CS 101 operation is that CS 101 mechanisms for handling certain events or interrupts should not rely in its operation upon those portions of CS 101 whose operation has resulted in those faults or interrupts. Among events handled by CS 101 monitor microcode, for example, are MEM 112 page faults. An MEM 112 page fault occurs whenever FU 120 makes a reference to data in MEM 112 and that data is not in MEM 112. Due to this and similar operations, MOS 426 resides entirely in FU 120 and thus does not rely upon information in MEM 112.

As described above, GRs 508, MIS 424, and MOS 426 each reside in certain assigned portions of GRF 506. This allows flexibility in modifying the capacity of GRs 508, MIS 424, and MOS 426 as indicated by experience, or to modify an individual CS 101 for particular purposes.

Referring finally to EU 122, EUS 512 is functionally a part of a process's SS 504. Also as previously described, EU 122 performs arithmetic operations in response to SINs and may be interrupted by FU 120 to aid certain FU 120 operations. EUS 512 allows stacking of interrupts. For example, FU 120 may first interrupt an arithmetic SOP to request EU 122 to aid in evaluation of a Name Table Entry. Before that first interrupt is completed, FU 120 may interrupt again, and so on.

SOP Stack 514, is a single frame stack for storing current state of EU 122 when an interrupt interrupts execution of an arithmetic SOP. An interrupted SOP's state is transferred into SOP Stack 514 and the interrupt begins execution in EUS 512. Upon occurrence of a second interrupt (before the first interrupt is completed) EU's first interrupt state is transferred from EUS 512 to a stack frame in SS 504, and execution

Petitioner Apple Inc. - Exhibit 1004, p. 5182

of the second interrupt begins in EUS 512. If a third interrupt occurs before completion of second interrupt, EU's second interrupt state is transferred from EUS 512 to another stack frame in SS 504 and execution of the third interrupt is begun in EUS 512; and so on. EUS 512 and SS 504 thus provide an apparently infinitely deep microstack for EU 122. Assuming that the third interrupt is completed, state of second interrupt is transferred from SS 504 to EUS 512 and execution of second interrupt resumed. Upon completion of second interrupt, state of first interrupt is transferred from SS 504 to EUS 512 and completed. After completion of first interrupt, state of the original SOP is transferred from SOP Stack 514 to EUS 512 and execution of that SOP resumed.

C. Procedure Processes, and Virtual Processors (Fig. 6)

Referring to Fig. 6, a diagramic representation of procedures, processes, and virtual processes is shown. As described above, a user's program to be executed is compiled to result in a Procedure 602. A Procedure 602 includes a User's Procedure Object 604 containing the SOPs of the user's program and a Name Table containing Entries for operand Names of the user's program, and a Static Data Area 606. A Procedure 602 may also include other Procedure Objects 608, for example utility programs available in common to many users. In effect, a Procedure 602 contains the instructions (procedures) and data of a user's program.

A Process 610 includes, as described above, a Macro Stack (MAS) 502 storing state of execution of a user's Procedure 602 at the SOP level, and a Secure Stack (SS) 504 storing state of execution of a user's Procedure 602 at the microcode level. A Process 610 is associated with a user's Procedure 602 through the ABPs described above and which are stored in the MAS 502 of the Process 610. Similarly, the MAS 502 and SS 504 of a Process 610 are associated through non-architectural pointers, described above. A Process 602 is effectively a body of information linking the resources, hardware, microcode, and software, of CS 101 to a user's Procedure 602. In effect, a Process 610 makes the resources of CS 101 available to a user's Procedure 602 for executing of that Procedure 602. CS 101 is a multi-program machine capable of accommodating up to, for example, 128 processes 610 concurrently. The number of Processes 610 which may be executed concurrently is determined by the number of Virtual Processors 612 of CS 101. There may be, for example, up to 16 Virtual Processors 612.

As indicated in Fig. 6, a Virtual Processor 612 is comprised of a Virtual Processor State Block (VPSB) 614 associated with the SS 504 of a Process 612. A VPSB 614 is, in effect, a body of information accessible to CS 101's operating system and through which CS 101's operating system is informed of, and provided with access to, a Process 610 through that process 610's SS 504. A VPSB 614 is associated with a particular Process 610 by writing information regarding that Process 610 into that VPSB 614. CS 101's operating system may, by gaining access to a Process 610 through an associated PSB 614, read information, such as ABP's, from that Process 610 to FU 120, thereby swapping that Process 610 onto FU 120 for execution. It is said that a Virtual Processor 612 thereby executes a process 610; a Virtual Processor 612 may be regarded therefor, as a processor having "Virtual", or potential, existence which becomes "real" when its associated Process 610 is swapped into FU 120. In CS 101, as indicated in Fig. 6, only one Virtual Processor 612 may execute on FU 120 at a time and the operating system selects which Virtual Processor 612 will excecute on FU 120 at any given time. In addition, CS 101's operating system selects which Processes 610 will be associated with the available Virtual Processors 612.

Having briefly described certain individual structural and operating features of CS 101, the overall operation of CS 101 will be described in further detail next below in terms of these individual features.

D. CS 101 Overall Structure and Operation (Figs. 7, 8, 9, 10, 11, 12, 13, 14, 15)

1. Introduction (Fig. 7)

As indicated in Fig. 7, CS 101 is a multiple level system wherein operations in one level are generally transparent to higher levels. User 701 does not see the S-Language, addressing, and protection mechanisms defined at Architectural Level 708. Instead, he sees User Interface 709, which is defined by Compilers 702, Binder 703, and Extended (high level) Operating System (EOS) 704. Compilers 702 translate high-level language code into SINs and Binder 703 translates symbolic Names in programs into UID-offset addresses.

As Fig. 7 shows, Architectural Level 708 is not defined by FU 120 Interface 711. Instead, the architectural resources level are created by S-Language Interpreted SINs when a program is executed; Name Interpreter 715 operates under control of S-Language Interpreters 705 and translates Names into logical descriptors. In CS 101, both S-Language Interpreters 705 and Name Interpreter 715 are implemented as microcode which executes on FU 120. S-Language Interpreters 705 may also use EU 122 to perform calculations. A Kernel Operating System (KOS) provides CS 101 with UID-offset addressing, objects, access checking, processes, and virtual processors, described further below. KOS has three kinds of components: KOS Microcode 710, KOS Software 706, and KOS Tables in MEM 112. KOS 710 components are microcode routines which assist FU 120 in performing certain required operations. Like other high-level language routines, KOS 706 components contain SINs which are interpreted by S-Interpreter Microcode 705. Many KOS High-Level Language Routines 706 are executed by special KOS processes; others may be executed by any process. Both KOS High-Level Language Routines 706 and KOS Microcode 710 manipulate KOS Tables in MEM 112.

Petitioner Apple Inc. - Exhibit 1004, p. 5183

FU 120 Interface 711 is visible only to KOS and to S-Interpreter Microcode 705. For the purposes of this discussion, FU 120 may be seen as a processor which contains the following main elements:

A Control Mechanism 725 which executes microcode stored in Writable Control Store 713 and manipulates FU 120 devices as directed by this microcode.

A GRF 506 containing registers in which data may be stored.

A Processing Unit 715.

All microcode which executes on FU 120 uses these devices; there is in addition a group of devices for performing special functions; these devices are used only by microcode connected with those functions. The microcode, the specialized devices, and sometimes tables in MEM 112 make up logical machines for performing certain functions. These machines will be described in detail below.

In the following, each of the levels illustrated in Fig. 7 will be discussed in turn. First, the components at User Interface 709 will be examined to see how they translate user programs and requests into forms usable by CS 101. Then the components below the User Interface 709 will be examined to see how they create logical machines for performing CS 101 operations.

2. Compilers 702 (Fig. 7)

Compilers 702 translate files containing the highlevel language code written by User 701 into Procedure Objects 608. Two components of a Procedure Object 608 are code (SOPs) and Names, previously described. SOPs represent operations, and the Names represent data. A single SIN thus specifies an operation to be performed on the data represented by the Names.

3. Binder 703 (Fig. 7)

In some cases, Compiler 702 cannot define locations as offsets from an ABP. For example, if a procedure calls a procedure contained in another procedure object, the location to which the call transfers control cannot be defined as an offset from the PBP used by the calling procedure. In these cases, the compiler uses symbolic Names to define the locations. Binder 703 is a utility which translates symbolic Names into UID-offset addresses. It does so in two ways: by combining separate Procedure Objects 608 into a single large Procedure Object 608, and then redefining symbolic Names as offsets from that Procedure Object 608's ABPs, or by translating symbolic Names when the program is executed. In the second case, Binder 703 requires assistance from EOS 704.

4. EOS 704 (Fig. 7)

EOS 704 manages the resources that User 701 requires to execute his programs. From User 701's point of view, the most important of these resources are files and processes. EOS 704 creates files by requesting KOS to create an object and then mapping the file onto the object. When a User 701 performs an operation on a file, EOS 704 translates the file operation into an operation on an object. KOS creates them at EOS 704's request and makes them available to EOS 704, which in turn makes them available to User 701. EOS 704 causes a process to execute by associating it a Virtual Processor 612. In logical terms, a Virtual Processor 612 is the means which KOS provides EOS 704 for executing processes 610. As many Processes 610 may apparently execute simultaneously in CS 101 as there are Virtual Processors 612. The illusion of simultaneous execution is created by multiplexing JP 114 among the Virtual processors; the manner in which Processes 610 and Virtual Processors 610 are implemented will be explained in detail below.

5. KOS and Architectural Interface 708 (Fig. 7)

S-Interpreter Microcode 710 and Name Interpreter Microcode 715 require an environment provided by KOS Microcode 710 and KOS Software 706 to execute SINs. For example, as previously explained, Names and program locations are defined in terms of ABPs whose values vary during execution of the program. The KOS environment provides values for the ABPs, and therefore makes it possible to interpret Names and program locations as locations in MEM 112. Similarly, KOS help is required to transform logical descriptors into references to MEM 112 and to perform protection checks.

The environment provided by KOS has the following elements:

A Process 610 which contains the state of an execution of the program for a given User 701.

A Virtual Processor 612 which gives the Process 610 access to JP 114.

An Object Management System which translates UIDs into values that are usable inside JP 114.

A Protection System which checks whether a Process 610 has the right to perform an operation on an Object.

A Virtual Memory Management System which moves those portions of Objects which a Process 610 actually references from the outside world into MEM 112 and translates logical descriptors into physical descriptors.

In the following, the logical properties of this environment and the manner in which a program is executed in it will be explained.

6. Processes 610 and Virtual Processors 612 (Fig. 8)

Processes 610 and Virtual Processors 612 have already been described in logical terms; Fig. 8 gives a high-level view of their physical implementation.

Petitioner Apple Inc. - Exhibit 1004, p. 5184

Fig. 8 illustrates the relationship between Processes 610, Virtual Processors 612, and JP 114. In physical terms, a Process 610 is an area of MEM 112 which contains the current state of a user's execution of a program. One example of such state is the current values of the ABPs and a program Counter (PC). Given the current value of the PBP and the PC, the next SOP in the program can be executed; similarly, given the current values of SDP and FP, the program's Names can be correctly resolved. Since the Process 610 contains the current state of a program's execution, the program's physical execution can be stopped and resumed at any point. It is thus possible to control program execution by means of the Process 610.

As already mentioned, a process 610's execution proceeds only when KOS has bound it to a Virtual Processor 612, that is, an area of MEM 112 containing the state required to execute microinstructions on JP 114 hardware. The operation of binding is simply a transfer of Process 610 state from the Process 610's area of MEM 112 to a Virtual Processor 612's area of MEM 112. Since binding and unbinding may take place at any time, EOS 704 may multiplex Processes 610 among Virtual Processors 612. In Fig. 8, there are more Processes 610 than there are Virtual Processors 612. The physical execution of a Process 610 on JP 114 takes place only while the Process 610's Virtual Processor 612 is bound to JP 114, i.e., when state is transferred from Virtual Processor 612's area of MEM 112 to JP 114's registers. Just as EOS 704 multiplexes Virtual Processors 612 among Processes 610, KOS multiplexes JP 114 among Virtual Processors 612. In Fig. 8, only one Process 610 is being physically executed. The means by which JP 114 is multiplexed among Virtual Processors 612 will be described in further detail below.

7. Processes 610 and Stacks (Fig. 9)

In CS 101 systems, a Process 610 is made up of six Objects: one Process Object 901 and Five Stack Objects 902 to 906. Fig. 9 illustrates a Process 610. Process Object 901 contains the information which EOS 704 requires to manage the Process 610. EOS 704 has no direct access to Process Object 901, but instead obtains the information it needs by means of functions provided to it by KOS 706, 710. Included in the information are the UIDs of Stack Objects 902 through 906. Stack Objects 902 to 906 contain the Process 610's state.

Stack Objects 902 through 905, are required by CS 101's domain protection method and comprise Process 610's MAS 502. Briefly, a domain is determined in part by operations performed when a system is operating in that domain. For example, the system is in EOS 704 domain when executing EOS 704 operations and in KOS 706, 710 domain when executing KOS 706, 710 operations. A Process 610 must have one stack for each domain it enters. In the present embodiment, the number of domains is fixed at four, but alternate embodiments may allow any number of domains, and correspondingly, any number of Stack Objects. Stack Object 906 comprises Process 610's Secure Stack 504 and is required to store state which may be manipulated only by KOS 706, 710.

Each invocation made by a Process 610 results in the addition of frames to Secure Stack 504 and to Macro-Stack 502. The state stored in the Secure Stack 504 frame includes the macrostate for the invocation, the state required to bind Process 610 to a Virtual Processor 612. The frame added to Macro-Stack 502 is placed in one of Stack Objects 902 through 905. Which Stack Objects 902 to 905 gets the frame is determined by the invoked procedure's domain of execution.

Fig. 9 shows the condition of a Process 610's MAS 502 and Secure Stack 504 after the Process 610 has executed four invocations. Secure Stack 504 has one frame for each invocation; the frames of Process 610's MAS 502 are found in Stack Objects 902, 904, and 905. As revealed by their locations, Frame 1 is for an invocation of a routine with KOS 706, 710 domain of execution, Frame 2 for an invocation of a routine with the EOS 704 domain of execution, and Frames 3 and 4 for invocations of routines with the User domain of execution. Process 610 has not yet invoked a routine with the Data Base Management System (DBMS) domain of execution. The frames in Stack Objects 902 through 905 are linked together, and a frame is added to or removed from Secure Stack 504 every time a frame is added to Stack Objects 902 through 905. MAS 502 and Secure Stack 504 thereby function as a single logical stack even though logically contained in five separate Objects.

8. Processes 610 and Calls (Figs. 10, 11)

In the CS 101, calls and returns are executed by KOS 706, 710. When KOS 706, 710 performs a call for a process, it does the following:

It saves the calling invocation's macrostate in the top frame of Secure Stack 504 (Fig. 9).

It locates the procedure whose Name is contained in the call. The location of the first SIN in the procedure becomes the new PBP.

Using information contained in the called procedure, KOS 706, 710 creates a new MAS 502 frame in the proper Stack Object 902 through 905 and a new Secure Stack 504 frame in Secure Stack 504. FP is updated to point to the new MAS 502. If necessary, SDP is also updated.

Once the values of the ABPs have been updated, the PC is defined, Names can be resolved, and execution of the invoked routine can commence. On a return from the invocation to the invoking routine, the stack frames are deleted and the ABPs are set to the values saved in the invoking routine's macrostate. The invoking routine then continues execution at the point following the invocation.

A Process 610 may be illustrated in detail by putting the FORTRAN statement A + B into a FORTRAN routine called EXAMPLE and invoking it from another FORTRAN routine named CALLER. To simplify the

Petitioner Apple Inc. - Exhibit 1004, p. 5185

example, it is assumed that CALLER and EXAMPLE both have the same domain of execution. The parts of EXAMPLE which are of interest look like this:

```
SUBROUTINE EXAMPLE (C)
INTEGER X,C
INTEGER A,B
...
A = B
...
RETURN
END
```

The new elements are a formal argument, C, and a new local variable, X. A formal argument is a data item which receives its value from adata item used in the invoking routine. The formal argument's value thus varies from invocation to invocation. The portions of INVOKER which are of interest look like this:

```
SUBROUTINE INVOKER
INTEGER Z
...
    CALL EXAMPLE (Z)
...
END
```

The CALL statement in INVOKER specifies the Name of the subroutine being invoked and the actual arguments for the subroutine's formal arguments. During the invocation, the subroutine's formal arguments take on the values of the actual arguments. Thus, during the invocation specified by this CALL statement, the formal argument C will have the value represented by the variable Z in INVOKER.

When INVOKER is compiled, the compiler produces a CALL SIN corresponding to the CALL statement. The CALL SIN contains a Name representing a pointer to the beginning of the called routine's location in a procedure object and a list of Names representing the call's actual arguments. When CALL is executed, the Names are interpreted to resolve the SIN's Names as previously described, and KOS 710 microcode to perform MAS 502 and Secure Stack 504 operations.

Fig. 10 illustrates the manner in which the KOS 710 call microcode manipulates MAS 502 and Secure Stack 504.

Fig. 10 includes the following elements:

Call Microcode 1001, contained in FU 120 Writable Control Store 1014.

PC Device 1002, which contains part of macrostate belonging to the invocation of INVOKER which is executing the CALL statement.

Registers in FU Registers 1014. Registers 1004 contents include the remainder of macrostate and the descriptors corresponding to Names for EXAMPLE's location and the actual argument Z.

Procedure Object 1006 contains the entries for INVOKER and EXAMPLE, their Name Tables, and their code.

Macro-Stack Object 1008 (MAS 502) and Secure Stack Object 1010 (Secure Stack 504) contain the stack frames for the invocations of INVOKER and EXAMPLE being discussed here. EXAMPLE's frame is in the same Macro-Stack object as INVOKER's frame because both routines are contained in the same Procedure Object 1006, and therefore have the same domain of execution.

KOS Call Microcode 1001 first saves the macrostate of INVOKER's invocation on Secure Stack 504. As will be discussed later, when the state is saved, KOS 706 Call Microcode 1001 uses other KOS 706 microcode to translate the location information contained in the macrostate into the kind of pointers used in MEM 112. Then Microcode 1001 uses the descriptor for the routine Name to locate the pointer to EXAMPLE's entry in Procedure Object 1006. From the entry, it locates pointers to EXAMPLE's Name Table and the beginning of EXAMPLE's code. Microcode 1001 takes these pointers, uses other KOS 706 microcode to translate them into descriptors, and places the descriptors in the locations in Registers 1004 reserved for the values of the PBP and NTP. It then updates the values contained in PC Device 1002 so that

Petitioner Apple Inc. - Exhibit 1004, p. 5186

when the call is finished, the next SIN to be executed will be the first SIN in EXAMPLE.

CALL Microcode 1001 next constructs the frames for EXAMPLE on Secure Stack 504 and Macro-Stack 502. This discussion concerns itself only with Frame 1102 on Macro-Stack 502. Fig. 11 illustrates EXAMPLE's Frame 1102. The size of Frame 1102 is determined by EXAMPLE's local variables (X, A, and B) and formal arguments (C). At the bottom of Frame 1102 is Header 1104. Header 1104 contains information used by KOS 706, 710 to manage the stack. Next comes Pointer 1106 to the location which contains the value represented by the argument C. In the invocation, the actual for C is the local variable Z in INVOKER. As is the case with all local variables, the storage represented by Z is contained in the stack frame belonging to INVOKER's invocation. When a name interpreter resolved C's name, it placed the descriptor in a register. Call Microcode 1001 takes this descriptor, converts it to a pointer, and stores the pointer above Header 1104.

Since the FP ABP points to the location following the last pointer to an actual argument, Call Microcode 1001 can now calculate that location, convert it into a descriptor, and place it in a FU Register 1004 reserved for FP. The next step is providing storage for EXAMPLE's local variables. EXAMPLE's procedure Object 1006 contains the size of the storage required for the local variables, so Call Microcode 1001 obtains this information from procedure Object 1006 and adds that much storage to Frame 1102. Using the new value of FP and the information contained in the Name Table Entries for the local data, Name Interpreter 715 can now construct descriptors for the local data. For example, A's entry in Name Table specified that it was offset 32 bits from FP, and was 32 bits long. Thus, its storage falls between the storage for X and B in Figure 11.

9. Memory References and the Virtual Memory Managment System (Fig. 12, 13)

As already explained, a logical descriptor contains an AON field, an offset field, and a length field. Fig. 12 illustrates a Physical Descriptor. Physical Descriptor 1202 contains a Frame Number (FN) field, a Displacement (D) field, and a Length (L) field. Together, the Frame Number field and the Displacement field specify the location in MEM 112 containing the data, and the Length field specifies the length of the data.

As is clear from the above, the virtual memory management system must translate the AON-offset location contained in a logical descriptor 1204 into a Frame Number-Displacement location. It does so by associating logical pages with MEM 112 frames. (N.B: MEM 112 frames are not to be confused with stack frames). Fig. 13, illustrates how Macrostack 502 Object 1302 is divided into Logical Pages 1304 in secondary memory and how Logical Pages 1304 are moved onto Frames 1306 in MEM 112. A Frame 1306 is a fixed-size, contiguous area of MEM 112. When the virtual memory management system brings data into MEM 112, it does so in frame-sized chunks called Logical Pages 1308. Thus, from the virtual memory system's point of view, each object is divided into Logical Pages 1308 and the address of data on a page consists of the AON of the data's Object, the number of pages in the object, and its displacement on the page. In Fig. 13, the location of the local variable B of EXAMPLE is shown as it is defined by the virtual memory system. B's location is a UID and an offset, or, inside JP 114, an AON and an offset. As defined by the virtual memory system, B's location is the AON, the page number 1308, and a displacement within the page. When a process references the variable B, the virtual memory management system moves all of Logical Page 1308 into a MEM 112 Frame 1306. B's displacement remains the same, and the virtual memory system translates its Logical Page Number 1308 into the number of Frame 1306 in MEM 112 which contains the page.

The virtual memory management system must therefore perform two kinds of translations: (1) AON-offset addresses into AON-page number-displacement addresses, and (2) AON-page number into a frame number.

10. Access Control (Fig. 14)

Each time a reference is made to an Object, KOS 706, 710 checks whether the reference is legal. The following discusson will first present the logical structure of access control in CS 101, and then discuss the microcode and devices which implement it.

CS 101 defines access in terms of subjects, modes of access, and Object size. A process may reference a data item located in an Object if three conditions hold:

1) If the process's subject has access to the Object.

2) If the modes of access specified for the subject include those required to perform the intended operation.

3) If the data item is completely contained in the Object, i.e., if the data item's length added to the data item's offset do not exceed the number of bits in the Object.

The subjects which have access to an Object and the kinds of access they have to the Object are specified by a data structure associated with the Object called the Access Control List (ACL). An Object's size is one of its attributes. Neither an Object's size nor its ACL is contained in the Object. Both are contained in system tables, and are accessible by means of the Object's UID.

Fig. 14 shows the logical structure of access control in CS 101. Subject 1408 has four components: Principal 1404, Process 1405, Domain 1406, and Tag 1407. Tag 1407 is not implemented in a present embodiment of CS 101, so the following description will deal only with principal 1404, Process 1405, and Domain 1406.

Petitioner Apple Inc. - Exhibit 1004, p. 5187

Principal 1404 specifies a user for which the process which is making the reference was created;
Process 1405 specifies the process which is making the reference; and,
Domain 1406 specifies the domain of execution of the procedure which the process is executing when it makes the reference.

Each component of the Subject 1408 is represented by a UID. If the UID is a null UID, that component of the subject does not affect access checking. Non-null UIDs are the UIDs of Objects that contain information about the subject components. Principal Object 1404 contains identification and accounting information regarding system users, Process Object 1405 contains process management information, and Domain Object 1406 contains information about per-domain error handlers.

There may be three modes of accessing an Object 1410: read, write, and execute. Read and write are self-explanatory; execute is access which allows a subject to execute instructions contained in the Object.

Access Control Lists (ACLs) 1412 are made up of Entries 1414. Each entry two components: Subject Template 1416 and Mode Specifier 1418. Subject Template 1416 specifies a group of subjects that may reference the Object and Mode Specifier 1418 specifies the kinds of access these subjects may have to the Object. Logically speaking, ACL 1412 is checked each time a process references an Object 1410. The reference may succeed only if the process's current Subject 1408 is one of those on Object 1410's ACL 1412 and if the modes in the ACL Entry 1414 for the Subject 1408 allow the kind of access the process wishes to make.

11. Virtual Processors and Virtual Processor Swapping (Fig. 15)

As previously mentioned, the execution of a program by a Process 610 cannot take place unless EOS 704 has bound the Process 610 to a Virtual Processor 612. Physical execution of the Process 610 takes place only while the process's Virtual Processor 612 is bound to JP 114. The following discussion deals with the data bases belonging to a Virtual Processor 612 and the means by which a Virtual Processor 612 is bound to and removed from JP 114.

Fig. 15 illustrates the devices and tables which KOS 706, 710 uses to implement Virtual Processors 612. FU 120 WCS contains KOS Microcode 706 for binding Virtual Processors 612 to JP 114 and removing them from JP 114. Timers 1502 and Interrupt Line 1504 are hardware devices which produce signals that cause the invocation of KOS Microcode 706. Timers 1502 contains two timing devices: Interval Timer 1506, which may be set by KOS 706, 710 to signal when a certain time is reached, and Egg Timer 1508, which guarantees that there is a maximum time interval for which a Virtual processor 612 can be bound to JP 114 before it invokes KOS Microcode 706. Interrupt Line 1504 becomes active when JP 114 receives a message from IOS 116, for example when IOS 116 has finished loading a logical page into MEM 112.

FU 120 Registers 508 contain state belonging to the Virtual Processor 612 currently bound to JP 114. Here, this Virtual Processor 612 is called Virtual Processor A. In addition, Registers 508 contain registers reserved for the execution of VP Swapping Microcode 1510. ALU 1942 (part of FU 120) is used for the descriptor-to-pointer and pointer-to-descriptor transformations required when one Virtual Processor 612 is unbound from JP 114 and another bound to JP 114. MEM 112 contains data bases for Virtual Processors 612 and data bases used by KOS 706, 710 to manage Virtual Processors 612. KOS 706, 710 provides a fixed number of Virtual Processors 612 for CS 101. Each Virtual Processor 612 is represented by a Virtual Processor State Block (VPSB) 614. Each VPSB 614 contains information used by KOS 706, 710 to manage the Virtual Processor 612, and in addition contains information associating the Virtual Processor 612 with a process. Fig. 15 shows two VPSBs 614, one belonging to Virtual Processor 612A, and another belonging to Virtual Processor 612B, which will replace Virtual Processor 612A on JP 114. The VPSBs 614 are contained in VPSB Array 1512. The index of a VPSB 614 in VPSB Array 1512 is Virtual Processor Number 1514 belonging to the Virtual Processor 612 represented by a VPSB 614. Virtual Processor Lists 1516 are lists which KOS 706, 710 uses to manage Virtual Processors 612. If a Virtual Processor 612 is able to execute, its Virtual Processor Number 1514 is on a list called the Runnable List; Virtual Processors 612 which cannot run are on other lists, depending on the reason why they cannot run. It is assumed that Virtual Processor 612B's Virtual Processor Number 1514 is the first one on the Runnable List.

When a process is bound to a Virtual Procesor 612, the Virtual Processor Number 1514 is copied into the process's Process Object 901 and the AONs of the process's Process Object 901 and stacks are copied into the Virtual Processor 612's VPSB 614. (AONs are used because a process's stacks are wired active as long as the process is bound to a Virtual Processor 612). Binding is carried out by KOS 706, 710 at the request of EOS 704. In Fig. 15, two Secure Stack Objects 906 are shown, one belonging to the process to which Virtual Processor 612A is bound, and one belonging to that to which Virtual Processor 612B is bound.

Having described certain overall operating features of CS 101, a present implementation of CS 101's structure will be described further next below.

E. CS 101 Structural Implementation (Figs. 16, 17, 18, 19, 20)

1. (IOS) 116 (Figs. 16, 17)

Referring to Fig. 16, a partial block diagram of IOS 116 is shown. Major elements of IOS 116 include an ECLIPSE® Burst Multiplexer Channel (BMC) 1614 and a NOVA® Data Channel (NDC) 1616, an IO Controller (IOC) 1618 and a Data Mover (DM) 1610. IOS 116's data channel devices, for example BMC 1614 and NDC 1616, comprise IOS 116's interface to the outside world. Information and addresses are received from

Petitioner Apple Inc. - Exhibit 1004, p. 5188

external devices, such as disk drives, communications modes, or other computer systems, by IOS 116's data channel devices and are transferred to DM 1610 (described below) to be written into MEM 112. Similarly, information read from MEM 112 is provided through DM 1610 to IOS 116's data channel devices and thus to the above described external devices. These external devices are a part of CS 101's addressable memory space and may be addressed through UID addresses.

IOC 1618 is a general purpose CPU, for example an ECLIPSE® computer available from Data General Corporation. A primary function of IOC 1618 is control of data transfer through IOS116. In addition, IOC 1618 generates individual Maps for each data channel device for translating external device addresses into physical addresses within MEM 112. As indicated in Fig. 16, each data channel device contains an individual Address Translation Map (MAP) 1632 and 1636. This allows IOS 116 to assign individual areas of MEM 112's physical address space to each data channel device. This feature provides protection against one data channel device writing into or reading from information belonging to another data channel device. In addition, IOC 1618 may generate overlapping address translation Maps for two or more data channel devices to allow these data channel devices to share a common area of MEM 112 physical address space.

Data transfer between IOS 116's data channel devices and MEM 112 is through DM 1610, which includes a Buffer memory (BUF) 1641. BUF 1641 allows MEM 112 and IOS 116 to operate asychronously. DM 1610 also includes a Ring Grant Generator (RGG) 1644 which controls access of various data channel devices to MEM 112. RGG 1644 is designed to be flexible in apportioning access to MEM 112 among IOS 116's data channel devices as loads carried by various data channel devices varies. In addition, RGG 1644 insures that no one, or group, of data channel devices may monopolize access to MEM 112.

Referring to Fig. 17, a diagramic representation of RGG 1644's operation is shown. As described further in a following description, RGG 1644 may be regarded as a commutator scanning a number of ports which are assigned to various channel devices. For example, ports A, C, E, and G may be assigned to a BMC 1614, ports B and F to a NDC 1616, and ports D and H to another data channel device. RGG 1644 will scan each of these ports in turn and, if the data channel device associated with a particular port is requesting access to MEM 112, will grant access to MEM 112 to that data channel device. If no request is present at a given port, RGG 1644 will continue immediately to the next port. Each data channel device assigned one or more ports is thereby insured opportunity of access to MEM 112. Unused ports, for example indicating data channel devices which are not presently engaged in information transfer, are effectively skipped over so that access to MEM 112 is dynamically modified according to the information transfer loads of the various data channel devices. RGG 1644's ports may be reassigned among IOS 116's various data channel devices as required to suit the needs of a particular CS 101 system. If, for example, a particular CS 101 utilizes NDC 1616 more than a BMC 1614, that CS 101's NDC 1616 may be assigned more ports while that CS 101's BMC 1614 is assigned fewer ports.

2. Memory (MEM) 112 (Fig. 18)

Referring to Fig. 18, a partial block diagram of MEM 112 is shown. Major elements of MEM 112 are Main Store Bank (MSB) 1810, a Bank Controller (BC) 1814, a Memory Cache (MC) 1816, a Field Interface Unit (FIU) 1820, and Memory Interface Controller (MIC) 1822. Interconnections of these elements with input and output buses of MEM 112 to IOS 116 and JP 114 are indicated.

MEM 112 is an intelligent, prioritizing memory having a single port to IOS 116, comprised of IOM Bus 130, MIO Bus 129, and IOMC Bus 131, and dual ports to JP 114. A first JP 114 port is comprised of MOD Bus 140 and PD Bus 146, and a second port is comprised of JPD Bus 142 and PD Bus 146. In general, all data transfers from and to MEM 112 by IOS 116 and JP 114 are of single, 32 bit words; IOM Bus 130, MIO Bus 129, MOD Bus 140, and JPD Bus 142 are each 32 bits wide. CS 101, however, is a variable word length machine wherein the actual physical width of data buses are not apparent to a user. For example, a Name in a user's program may refer to an operand containing 97 bits of data. To the user, that 97 bit data item will appear to be read from MEM 112 to JP 114 in a single operation. In actuality, JP 114 will read that operand from MEM 112 in a series of read operations referred to as a string transfer. In this example, the string transfer will comprise three 32 bit read transfers and one single bit read transfer. The final single bit transfer, containing a single data bit, will be of a 32 bit word wherein one bit is data and 31 bits are fill. Write operations to MEM 112 may be performed in the same manner. If a single read or write request to MEM 112 specifies a data item of less than 32 bits of data, that transfer will be accomplished in the same manner as the final transfer described above. That is, a single 32 bit word will be transferred wherein non-data bits are fill bits.

Bulk data storage in MEM 112 is provided in MSB 1810, which is comprised of one or more Memory Array cards (MAs) 1812. The data path into and out of MA 1812 is through BC 1814, which performs all control and timing functions for MAs 1812. BC 1814's functions include addressing, transfer of data, controlling whether a read or write operation is performed, refresh, sniffing, and error correction code operations. All read and write operations from and to MAs 1812 through BC 1814 are in blocks of four 32 bit words.

The various MAs 1812 comprising MSB 1810 need not be of the same data storage capacity. For example, certain MAs 1812 may have a capacity of 256 kilobytes while other MAs 1812 may have a capacity of 512 kilobytes. Addressing of the MAs 1812 in MSB 1810 is automatically adapted to various MA 1812

Petitioner Apple Inc. - Exhibit 1004, p. 5189

configurations. As indicated in Fig. 18, each MA 1812 contains an address circuit (A) which receives an input from the next lower MA 1812 indicating the highest address in that next lower MA 1812. The A circuit on an MA 1812 also receives an input from that MA 1812 indicating the total address space of that MA 1812. The A circuit of that MA 1812 adds the highest address input from next lower MA 1812 to its own input representing its own capacity and generates an output to the next MA 1812 indicating its own highest address. All MAs 1812 of MSB 1810 are addressed in parallel by BC 1814. Each MA 1812 compares such addresses to its input from the next lower MA 1812, representing highest address of that next lower MA 1812, and its own output, representing its own highest address, to determine whether a particular address provided by BC 1814 lies within the range of addresses contained within that particular MA 1812. The particular MA 1812 whose address space includes that address will then respond by accepting the read or write request from BC 1814.

MC 1816 is the data path for transfer of data between BC 1814 and IOS 116 and JP 114. MC 1816 contains a high speed cache storing data from MSB 1810 which is currently being utilized by either IOS 116 or JP 114. MSB 1810 thereby provides MEM 112 with a large storage capacity while MC 1816 provides the appearance of a high speed memory. In addition to operating as a cache, MC 1816 includes a bypass write path which allows IOS 116 to write blocks of four 32 bit words directly into MSB 1810 through BC 1814. In addition, MC 1816 includes a cache write-back path which allows data to be transferred out of MC 1816's cache and stored while further data is transferred into MC 1816's cache. Displaced data from MC 1816's cache may then be written back into MSB 1810 at a later, more convenient time. This write-back path enhances speed of operation of MC 1816 by avoiding delays incurred by transferring data from MC 1816 to MSB 1810 before new data may be written into MC 1816.

MEM 112's FIU 1820 allows manipulation of data formats in writes to and reads from MEM 112 by both JP 114 and IOS 116. For example, FIU 1820 may convert unpacked decimal data to packed decimal data, and vice versa. In addition, FIU 1820 allows MEM 112 to operate as a bit addressable memory. For example, as described all data transfers to and from MEM 112 are of 32 bit words. If a data transfer of less than 32 bits is required, the 32 bit word containing those data bits may be read from MC 1816 to FIU 1820 and therein manipulated to extract the required data bits. FIU 1820 then generates a 32 bit word containing those required data bits, plus fill bits, and provides that new 32 bit word to JP 114 or IOS 116. When writing into MEM 112 from IOS 116 through FIU 1820, data is transferred onto IOM Bus 130, read into FIU 1820, operated upon, transferred onto MOD Bus 140, and transferred from MOD Bus 140 to MC 1816. In read operations from MEM 112 to IOS 116, data is transferred from MC 1816 to MOD Bus 140, written into FIU 1820 and operated upon, and transferred onto MIO Bus 129 to IOS 116. In a data read from MEM 112 to JP 114, data is transferred from MC 1816 onto MOD Bus 140, transferred into FIU 1820 and operated upon, and transferred again onto MOD Bus 140 to JP 114. In write operations from JP 114 to MEM 112, data on JPD Bus 142 is transferred into FIU 1820 and operated upon, and is then transferred onto MOD Bus 140 to MC 1816. MOD Bus 140 is thereby utilized as an MEM 112 internal bus for FIU 1820 operations.

Finally, MIC 1822 provides primary control of BC 1814, MC 1816, and FIU 1820. MIC 1822 receives control inputs from and provides control outputs to PD Bus 146 and IOMC Bus 131. MIC 1822 contains primary microcode control for MEM 112, but BC 1814, MC 1816, and FIU 1820 each include internal microcode control. Independent, internal microcode controls allow BC 1814, MC 1816, and FIU 1820 to operate independently of MIC 1822 after their operations have been initiated by MIC 1822. This allows BC 1814 and MSB 1810, MC 1816, and FIU 1820 to operate independently and asynchronously. Efficiency and speed of operation of MEM 112 are thereby enhanced by allowing pipelining of MEM 112 operations.

3. Fetch Unit (FU) 120 (Fig. 19)

A primary function of FU 120 is to execute SINs. In doing so, FU 120 fetches instructions and data (SOPs and Names) from MEM 112, returns results of operations to MEM 112, directs operation of EU 122, executes instructions of user's programs, and performs the various functions of CS 101's operating systems. As part of these functions, FU 120 generates and manipulates logical addresses and descriptors and is capable of operating as a general purpose CPU.

Referring to Fig. 19, a major element of FU 120 is the Descriptor Processor (DESP) 1910. DESP 1910 includes General Register File (GRF) 506. GRF 506 is a large register array divided vertically into three parts which are addressed in parallel. A first part, AONGRF 1932, stores AON fields of logical addresses and descriptors. A second part, OFFGRF 1934, stores offset fields of logical addresses and descriptors and is utilized as a 32 bit wide general register array. A third portion GRF 506, LENGRF 1936, is a 32 bit wide register array for storing length fields of logical descriptors and as a general register for storing data. Primary data path from MEM 112 to FU 120 is through MOD Bus 140, which provides inputs to OFFGRF 1934. As indicated in Fig. 19, data may be transferred from OFFGRF 1934 to inputs of AONGRF 1932 and LENGRF 1936 through various interconnections. Similarly, outputs from LENGRF 1936 and AONGRF 1932 may be transferred to inputs of AONGRF 1932, OFFGRF 1934, and LENGRF 1936.

Output of OFFGRF 1934 is connected to inputs of DESP 1910's Arithmetic and Logic Unit (ALU) 1942. ALU 1942 is a general purpose 32 bit ALU which may be used in generating and manipulating logical addresses and descriptors, as distinct from general purpose arithmetic and logic operands performed by MUX 1940. Output of ALU 1942 is connected to JPD Bus 142 to allow results of arithmetic and logic operations to be transferred to MEM 112 or EU 122.

Petitioner Apple Inc. - Exhibit 1004, p. 5190

Also connected from output of OFFGRF 1934 is Descriptor Multiplexer (MUX) 1940. An output of MUX 1940 is provided to an input of ALU 1942. MUX 1940 is a 32 bit ALU, including an accumulator, for data manipulation operations. MUX 1940, together with ALU 1942, allows DESP 1910 to perform 32 bit arithmetic and logic operations. MUX 1940 and ALU 1942 may allow arithmetic and logic operations upon operands of greater than 32 bits by performing successive operations upon successive 32 bit words of larger operands.

Logical descriptors or addresses generated or provided by DESP 1910, are provided to Logical Descriptor (LD) Bus 1902. LD Bus 1902 in turn is connected to an input of Address Translation Unit (ATU) 1928. ATU 1928 is a cache mechanism for converting logical descriptors to MEM 112 physical descriptors.

LD Bus 1902 is also connected to write input of Name Cache (NC) 1926. NC 1926 is a cache mechanism for storing logical descriptors corresponding to operand Names currently being used in user's programs. As previously described, Name Table Entries corresponding to operands currently being used in user's programs are stored in MEM 112. Certain Name Table Entries for operands of a user's program currently being executed are transferred from those Name Tables in MEM 112 to FU 120 and are therein evaluated to generate corresponding logical descriptors. These logical descriptors are then stored in NC 1926. As will be described further below, the instruction stream of a user's program is provided to FU 120's Instruction Buffer (IB) 1962 through MOD Bus 140. FU 120's Parser (P) 1964 separates out, or parses, Names from IB 1962 and provides those Names as address inputs to NC 1924. NC 1924 in turn provides logical descriptor outputs to LD Bus 1902, and thus to input of ATU 1928. NC 1926 input from LD Bus 1902 allows logical descriptors resulting from evaluation of Name Table Entries to be written into NC 1926. FU 120's Protections Cache (PC) 1934 is a cache mechanism having an input connected from LD Bus 1902 and providing information, as described further below, regarding protection aspects of references to data in MEM 112 by user's programs. NC 1926, ATU 1928, and PC 1934 are thereby acceleration mechanisms of, respectively, CS 101's Namespace addressing, logical to physical address structure, and protection mechanism.

Referring again to DESP 1910, DESP 1910 includes BIAS 1952, connected from output of LENGRF 1936. As previously described, operands containing more than 32 data bits are transferred beteen MEM 112 and JP 114 by means of string transfers. In order to perform string transfers, it is necessary for FU 120 to generate a corresponding succession of logical descriptors wherein length fields of those logical descriptors is no greater than 5 bits, that is, specify lengths of no greater than 32 data bits.

A logical descriptor describing a data item to be transferred by means of a string transfer will be stored in GRF 506. AON field of the logical descriptor will reside in AONGRF 1932, O field in OFFGRF 1934, and L field in LENGRF 1936. At each successive transfer of a 32 bit word in the string transfer, O field of that original logical descriptor will be incremented by the number of data bits transferred while L field will be accordingly decremented. The logical descriptor residing in GRF 506 will thereby describe, upon each successsive transfer of the string transfer, that portion of the data item yet to be transferred. O field in OFFGRF 1934 will indicate increasingly larger offsets into that data item, while L field will indicate successively shorter lengths. AON and O fields of the logical descriptor in GRF 506 may be utilized directly as AON and O fields of the successive logical descriptors of the string transfer. L field of the logical descriptor residing in LENGRF 1936, however, may not be so used as L fields of the successive string transfer logical descriptors as this L field indicates remaining length of data item yet to be transferred. Instead, BIAS 1952 generates the 5 bit L fields of successive string transfer logical descriptors while correspondingly decrementing L field of the logical descriptor in LENGRF 1936. During each transfer, BIAS 1952 generates L field of the *next* string transfer logical descriptor while concurrently providing L field of the *current* string transfer logical descriptor. By doing so, BIAS 1952 thereby increases speed of execution of string transfers by performing pipelined L field operations. BIAS 1952 thereby allows CS 101 to appear to the user to be a variable word length machine by automatically performing string transfers. This mechanism is used for transfer of any data item greater than 32 bits, for example double precision floating point numbers.

Finally, FU 120 includes microcode circuitry for controlling all FU 120 operations described above. In particular, FU 120 includes a microinstruction sequence control store (mC) 1920 storing sequences of microinstructions for controlling step by step execution of all FU 120 operations. In general, these FU 120 operations fall into two classes. A first class includes those microinstruction sequences directly concerned with executing the SOPs of user's programs. The second class includes microinstruction sequences concerned with CS 101's operating systems, including and certain automatic, internal FU 120 functions such as evaluation of Name Table Entries.

As previously described, CS 101 is a multiple S-Language machine. For example, mC 1920 may contain microinstruction sequences for executing user's SOPs in at least four different Dialects. mC 1920 is comprised of a writeable control store and sets of microinstruction sequences for various Dialects may be transferred into and out of mC 1920 as required for execution of various user's programs. By storing sets of microinstruction sequences for more than one Dialect in mC 1920, it is possible for user's programs to be written in a mixture of user languages. For example, a particular user's program may be written primarily in FORTRAN but may call certain COBOL routines. These COBOL routines will be correspondingly translated into COBOL dialect SOPs and executed by COBOL microinstruction sequences stored in mC 1920.

The instruction stream provided to FU 120 from MEM 112 has been previously described with

Petitioner Apple Inc. - Exhibit 1004, p. 5191

reference to Fig. 3. SOPs and Names of this instruction stream are transferred from MOD Bus 140 into IB 1962 as they are provided from MEM 112. IB 1962 includes two 32 bit (one word) registers. IB 1962 also includes prefetch circuitry for reading for SOPs and Names of the instruction stream from MEM 112 in such a manner that IB 1962 shall always contain at least one SOPs or Name. FU 120 includes (P) 1964 which reads and separates, or parses, SOPs and Names from IB 1962. As previously described, P 1964 provides those Names to NC 1926, which accordingly provides logical descriptors to ATU 1928 so as to read the corresponding operands from MEM 112.

SOPs parsed by P 1964 are provided as inputs to Fetch Unit Dispatch Table (FUDT) 1904 and Execute Unit Dispatch Table (EUDT) 1966. Referring first to FUDT 1904, FUDT 1904 is effectively a table for translating SOPs to starting addresses in mC 1912 of corresponding microinstruction sequences. This intermediate translation of SOPs to mC 1912 addresses allows efficient packing of microinstruction sequences within mC 1912. That is, certain microinstruction sequences may be common to two or more S-Language Dialects. Such microinstruction sequences may therefore be written into mC 1912 once andy may be referred to by different SOPs of different S-Language Dialects.

EUDT 1966 performs a similar function with respect to EU 122. As will be described below, EU 122 contains a mC, similar to mC 1912, which is addressed through EUDT 1966 by SOPs specifying EU 122 operations. In addition, FU 120 may provide such addresses mC 1912 to initiate EU 122 operations as required to assist certain FU 120 operations. Examples of such operations which may be requested by FU 120 include calculations required in evaluating Name Table Entries to provide logical descriptors to be loaded into NC 1926.

Associated with both FUDT 1904 and EUDT 1966 are Dialect (D) registers 1905 and 1967. D registers 1905 and 1967 store information indicating the particular S-Language Dialect currently being utilized in execution of a user's program. Outputs of D registers 1905 and 1967 are utilized as part of the address inputs to mC 1912 and EU 122's mC.

4. Execute Unit (EU) 122 (Fig. 20)

As previously described, EU 122 is an arithmetic and logic unit provided to relieve FU 120 of certain arithmetic operations. EU 122 is capable of performing addition, subtraction, multiplication, and division operations on integer, packed and unpacked decimal, and single and double precision floating operands. EU 122 is an independently operating microcode controlled machine including Microcode Control (mC) 2010 which, as described above, is addressed by EUDT 1966 to initiate EU 122 operations. mC 2010 also includes logic for handling mutual interrupts between FU 120 and EU 122. That is, FU 120 may interrupt current EU 122 operations to call upon EU 122 to assist an FU 120 operation. For example, FU 120 may interrupt an arithmetic operation currently being executed by EU 122 to call upon EU 122 to assist in generating a logical descriptor from a Name Table Entry.

Similarly, EU 122 may interrupt current FU 120 operations when EU 122 requires FU 120 assistance in executing a current arithmetic operation. For example, EU 122 may interrupt a current FU 120 operation if EU 122 receives an instruction and operands requiring EU 122 to perform a divide by zero.

Referring to Fig. 20, a partial block diagram of EU 122 is shown. EU 122 includes two arithmetic and logic units. A first arithmetic and logic unit (MULT) 2014 is utilized to perform addition, subtraction, multiplication, and division operations upon integer and decimal operands, and upon mantissa fields of single and double precision floating point operands. Second ALU (EXP) 2016 is utilized to perform operations upon single and double precision floating point operand exponent fields in parallel with operations performed upon floating point mantissa fields by MULT 2014. Both MULT 2014 and EXP 2016 include an arithmetic and logic unit, respectively MALU 2074 and EXPALU 2084. MULT 2014 and EXP 2016 also include register files, respectively MRF 2050 and ERF 2080, which operate and are addressed in parallel in a manner similar to AONGRF 1932, OFFGRF 1984 and LENGRF 1936.

Operands for EU 122 to operate upon are provided from MEM 112 through MOD Bus 140 and are transferred into Operand Buffer (OPB) 2022. In addition to serving as an input buffer, OPB 2022 performs certain data format manipulation operations to transform input operands into formats most efficiently operated with by EU 122. In particular, EU 122 and MULT 2014 may be designed to operate efficiently with packed decimal operands. OPB 2022 may transform unpacked decimal operands into packed decimal operands. Unpacked decimal operands are in the form of ASCII characters wherein four bits of each characters are binary codes specifying a decimal value between zero and nine. Other bits of each character are referred to as zone fields and in general contain information identifying particular ASCII characters. For example, zone field bits may specify whether a particular ASCII character is a number, a letter, or punctuation. Packed decimal operands are comprised of a series of four bit fields wherein each field contains a binary number specifying a decimal value of between zero and nine. OPB 2022 converts unpacked decimal to packed decimal operands by extracting zone field bits and packing the four numeric value bits of each character into the four bit fields of a packed decimal number.

EU 122 is also capable of transforming the results of arithmetic operands, for example in packed decimal format, into unpacked decimal format for transfer back to MEM 112 or FU 120. In this case, a packed decimal result appearing at output of MALU 2074 is written into MRF 2050 through a multiplexer, not shown in Fig. 20, which transforms the four bit numeric code fields of the packed decimal results into corresponding bits of unpacked decimal operand characters, and forces blanks into the zone field bits of

Petitioner Apple Inc. - Exhibit 1004, p. 5192

those unpacked decimal characters. The results of this operation are then read from MRF 2050 to MALU 2074 and zone field bits for those unpacked decimal characters are read from Constant Store (CST) 2060 to MALU 2074. These inputs from MRF 2050 and CST 2060 are added by MALU 2074 to generate final result outputs in unpacked decimal format. These final results may then be transferred onto JPD Bus 142 through Output Multiplexer (OM) 2024.

Considering first floating point operations, in addition or subtraction of floating point operands it is necessary to equalize the values of the floating point operand exponent fields. This is referred to as prealignment. In floating point operations, exponent fields of the two operands are transferred into EXPALU 2034 and compared to determine the difference between exponent fields. An output representing difference between exponent fields is provided from EXPALU 2034 to an input of floating point control (FPC) 2002. FPC 2002 in turn provides control outputs to MALU 2074, which has received the mantissa fields of the two operands. MALU 2074, operating under direction of FPC 2002, accordingly right or left shifts one operand's mantissa field to effectively align that operand's exponent field with the other operand's exponent field. Addition or subtraction of the operand's mantissa fields may then proceed.

EXPALU 2034 also performs addition or subtraction of floating point operand exponent fields in multiplication or division operations, while MALU 2074 performs multiplication and division of the operand mantissa fields. Multiplication and division of floating point operand mantissa fields by MALU 2074 is performed by successive shifting of one operand, corresponding generation of partial products of the other operand, and successive addition and subtraction of those partial products.

Finally, EU 122 performs normalization of the results of floating point operand operations by left shifting of a final result's mantissa field to eliminate zeros in the most significant characters of the final result mantissa field, and corresponding shifting of the final result exponent fields. Normalization of floating point operation results is controlled by FPC 2002. FPC 2002 examines an unnormalized floating point result output of MALU 2074 to detect which, if any, of the most significant characters of that results contain zeros. FPC 2002 then accordingly provides control outputs to EXPALU 2034 and MALU 2074 to correspondingly shift the exponent and mantissa fields of those results so as to eliminate leading character zeros from the mantissa field. Normalized mantissa and exponent fields of floating point results may then be transferred from MALU 2074 and EXPALU 2034 to JPD Bus 142 through OM 2024.

As described above, EU 122 also performs addition, subtraction, multiplication, and division operations on operands. In this respect, EU 122 uses a leading zero detector in FPC 2002 in efficiently performing multiplication and division operations. FPC 2002's leading zero detector examines the characters or bits of two operands to be multiplied or divided, starting from the highest, to determine which, if any, contain zeros so as not to require a multiplication or division operation. FPC 2002 accordingly left shifts the operands to effectively eliminate those characters or bits, thus reducing the number of operations to multiply or divide the operands and accordingly reducing the time required to operate upon the operands.

Finally, EU 122 utilizes a unique method, with associated hardware, for performing arithmetic operations on decimal operands by utilizing circuitry which is otherwise conventionally used only to perform operations upon floating point operands. As described above, MULT 2074 is designed to operate with packed decimal operands, that is operands in the form of consecutive blocks of four bits wherein each block of four bits contains a binary code representing numeric values of between zero and nine. Floating point operands are similarly in the form of consecutive blocks of four bits. Each block of four bits in a floating point operand, however, contains a binary number representing a hexadecimal value of between zero and fifteen. As an initial step in operating with packed decimal operands, those operands are loaded, one at a time, into MALU 2074 and, with each such operand, a number comprised of all hexadecimal sixes is loaded into MALU 2074 from CST 2060. This CST 2060 number is added to each packed decimal operand to effectively convert those packed decimal operands into hexadecimal operands wherein the four bit blocks contain numeric values in the range of six to fifteen, rather than in the original range of zero to nine. MULT 2014 then performs arithmetic operation upon those transformed operands, and in doing so detects and saves information regarding which four bit characters of those operands have resulted in generation of carries during the arithmetic operations. In a final step, the intermediate result resulting from completion of those arithmetic operations upon those transformed operands are reconverted to packed decimal format by subtraction of hexadecimal sixes from those characters for which carries have been generated. Effectively, EU 122 converts packed decimal operands into "Excess Six" operands, performs arithmetic operations upon those "Excess Six" operands, and reconverts "Excess Six" results of those operations back into packed decimal format.

Finally, as previously descibed FU 120 controls transfer of arithmetic results from EU 122 to MEM 112. In doing so, FU 120 generates a logical descriptor describing the size of MEM 112 address space, or "container", that result is to be transferred into. In certain arithmetic operations, for example integer operations, an arithmetic result may be larger than anticipated and may contain more bits than the MEM 112 "container". Container Size Check Circuit (CSC) 2052 compares actual size of arithmetic results and L fields of MEM 112 "container" logical descriptors. CSC 2052 generates an output indicating whether an MEM 112 "container" is smaller than an arithmetic result.

Having briefly described certain features of CS 101 structure and operation in the above overview, these and other features of CS 101 will be described in further detail next below in a more detailed

Petitioner Apple Inc. - Exhibit 1004, p. 5193

introduction of CS 101 structure and operation. Then, in further descriptions, these and other features of CS 101 structure and operation will be described in depth.

1. Introduction (Figs. 101—110)

A. General Structure and Operation (Fig 101)

a. General Structure

Referring to Fig. 101, a partial block diagram of Computer System (CS) 10110 is shown. Major elements of CS 10110 are Dual Port Memory (MEM) 10112, Job Processor (JP) 10114, Input/Output System (IOS) 10116, and Diagnostic Processor (DP) 10118. JP 10114 includes Fetch Unit (FU) 10120 and Execute Unit (EU) 10122.

Referring first to IOS 10116, IOS 10116 is interconnected with External Devices (ED) 10124 through Input/Output (I/O) Bus 10126. ED 10124 may include, for example, other computer systems, keyboard/display units, and disc drive memories. IOS 10116 is interconnected with Memory Input/Output (MIO) Port 10128 of MEM 10112 through Input/Output to Memory (IOM) Bus 10130 and Memory to Input/Output (MIO) Bus 10129, and with FU 10120 through I/O Job Processor (IOJP) Bus 10132.

DP 10118 is interconnected with, for example, external keyboard/CRT Display Unit (DU) 10134 through Diagnostic Processor Input/Output (DPIO) Bus 10136. DP 10118 is interconnected with IOS 10116, MEM 10112, FU 10120, and EU 10122 through Diagnostic Processor (DP) Bus 10138.

Memory to Job Processor (MJP) Port 10140 of Memory 10112 is interconnected with FU 10120 and EU 10122 through Job Processor Data (JPD) Bus 10142. An output of MJP 10140 is connected to inputs of FU 10120 and EU 10122 through Memory Output Data (MOD) Bus 10144. An output of FU 10120 is connected to an input of MJP 10140 through Physical Descriptor (PD) Bus 10146. FU 10120 and EU 10122 are interconnected through Fetch/Execute (F/E) Bus 10148.

b. General Operation

As will be discussed further below, IOS 10116 and MEM 10112 operate independently under general control of JP 10114 in executing multiple user's programs. In this regard, MEM 10112 is an intelligent, prioritizing memory having separate and independent ports MIO 10128 and MJP 10140 to IOS 10116 and JP 10114 respectively. MEM 10112 is the primary path for information transfer between External Devices 10124 (through IOS 10116) and JP 10114. MEM 10112 thus operates both as a buffer for receiving and storing various individual user's programs (e.g., data, instructions, and results of program execution) and as a main memory for JP 10114.

A primary function of IOS 10116 is as an input/output buffer between CS 10110 and ED 10124. Data and instructions are transferred from ED 10124 to IOS 10116 through I/O Bus 10126 in a manner and format compatible with ED 10124. IOS 10116 receives and stores this information, and manipulates the information into formats suitable for transfer into MEM 10112. IOS 10116 then indicates to MEM 10112 that new information is available for transfer into MEM 10112. Upon acknowledgement by MEM 10112, this information is transferred into MEM 10112 through IOM Bus 10130 and MIO Port 10128. MEM 10112 stores the information in selected portions of MEM 10112 physical address space. At this time, IOS 10116 notifies JP 10114 that new information is present in MEM 10112 by providing a "semaphore" signal to FU 10120 through IOJP Bus 10132. As will be described further below, CS 10110 manipulates the data and instructions stored in MEM 10112 into certain information structures used in executing user's programs. Among these structures are certain structures, discussed further below, which are used by CS 10110 in organizing and controlling flow and execution of user programs.

FU 10120 and EU 10122 are independently operating microcode controlled "machines" together comprising the CS 10110 micromachine for executing user's programs stored in MEM 10112. Among the principal functions of FU 10120 are: (1) fetching and interpreting instructions and data from MEM 10112 for use by FU 10120 and EU 10122; (2) organizing and controlling flow of user programs; (3) initiating EU 10122 operations; (4) performing arithmetic and logic operations on data; (5) controlling transfer of data from FU 10120 and EU 10122 to MEM 10112; and, (6) maintaining certain "stack" and "register" mechanisms, described below. FU 10120 "cache" mechanisms, also described below, are provided to enhance the speed of operation of JP 10114. These cache mechanisms are acceleration circuitry including, in part, high speed memories for storing copies of selected information stored in MEM 10112. The information stored in this acceleration circuitry is therefore more rapidly available to JP 10114. EU 10122 is an arithmetic unit capable of executing integer, decimal, or floating point arithmetic operations. The primary function of EU 10122 is to relieve FU 10120 from certain extensive arithmetic operations, thus enhancing the efficiency of CS 10110.

In general, operations in JP 10114 are executed on a memory to memory basis; data is read from MEM 10112, operated upon, and the results returned to MEM 10112. In this regard, certain stack and cache mechanisms in JP 10114 (described below) operate as extensions of MEM 10112 address space.

In operation, FU 10120 reads data and instructions from MEM 10112 by providing physical addresses to MEM 10112 by way of PA Bus 10146 and MJP Port 10140. The instructions and data are transferred to FU 10120 and EU 10122 by way of MJP Port 10140 and MOD Bus 10144. Instructions are interpreted by FU 10120 microcode circuitry, not shown in Fig. 101 but described below, and when necessary, microcode instructions are provided to EU 10122 from FU 10120's microcode control by way of F/E Bus 10148, or by way of JPD Bus 10142.

Petitioner Apple Inc. - Exhibit 1004, p. 5194

As stated above, FU 10120 and EU 10122 operate asynchronously with respect to each other's functions. A microinstruction from FU 10120 microcode circuitry to EU 10122 may initiate a selected operation of EU 10122. EU 10122 may then proceed to independently execute the selected operation. FU 10120 may proceed to concurrently execute other operations while EU 10122 is completing the selected arithmetic operation. At completion of the selected arithmetic operation, EU 10122 signals FU 10120 that the operation results are available by way of a "handshake" signal through F/E Bus 10148. FU 10120 may then receive the arithmetic operation results for further processing or, as discussed momentarily, may directly transfer the arithmetic operation results to MEM 10112. As described further below, an instruction buffer referred to as a "queue" between FU 10120 and EU 10122 allows FU 10120 to assign a sequence of arithmetic operations to be performed by EU 10122.

Information, such as results of executing an instruction, is written into MEM 10112 from FU 10120 or EU 10122 by way of JPD Bus 10142. FU 10120 provides a "physical write address" signal to MEM 10112 by way of PA Bus 10146 and MJP Port 10140. Concurrently, the information to be written into MEM 10112 is placed on JPD Bus 10142 and is subsequently written into MEM 10112 at the locations selected by the physical write address.

FU 10120 places a semaphore signal on IOJP Bus 10132 to signal to IOS 10116 that information, such as the results of executing a user's program, is available to be read out of CS 10110. IOS 10116 may then transfer the information from MEM 10112 to IOS 10116 by way of MIO Port 10128 and IOM Bus 10130. Information stored in IOS 10116 is then transferred to ED 10124 through I/O Bus 10126.

During execution of a user's program, certain information required by JP 10116 may not be available in MEM 10112. In such cases as further described in a following discussion, JP 10114 may write a request for information into MEM 10112 and notify IOS 10116, by way of IOJP Bus 10132, that such a request has been made. IOS 10116 will then read the request and transfer the desired information from ED 10124 into MEM 10112 through IOS 10116 in the manner described above. In such operations, IOS 10116 and JP 10114 operate together as a memory manager wherein the memory space addressable by JP 10114 is termed virtual memory space, and includes both MEM 10112 memory space and all external devices to which IOS 10116 has access.

As previously described, DP 10118 provides a second interface between Computer System 10110 and the external world by way of DPIO Bus 10136. DP 10118 allows DU 10134, for example a CRT and keyboard unit or a teletype, to perform all functions which are conventionally provided by a hard (i.e., switches and lights) console. For example, DP 10118 allows DU 10134 to exercise control of Computer System 10110 for such purposes as system initialization and start up, execution of diagnostic processes, and fault monitoring and identification. DP 10118 has read and write access to most memory and register portions within each of IOS 10116, MEM 10112, FU 10120, and EU 10122 by way of DP Bus 10138. Memories and registers in CS 10110 can therefore be directly loaded or initialized during system start up, and can be directly read or loaded with test and diagnostic signals for fault monitoring and identification. In addition, as described further below, microinstructions may be loaded into JP 10114's microcode circuitry at system start up or as required.

Having described the general structure and operation of Computer System 10110, certain features of Computer System 10110 will next be briefly described to aid in understanding the following, more detailed descriptions of these and other features of Computer System 10110.

c. Definition of Certain Terms

Certain terms are used relating to the structure and operation of CS 10110 throughout the following discussions. Certain of these terms will be discussed and defined first, to aid in understanding the following descriptions. Other terms will be introduced in the following descriptions as required.

A *procedure* is a sequence of operational steps, or instructions, to be executed to perform some operation. A procedure may include data to be operated upon in performing the operation.

A *program* is a static group of one or more procedures. In general, programs may be classified as user programs, utility programs, and operating system programs. A user program is a group of procedures generated by and private to one particular user of a group of users interfacing with CS 10110. Utility programs are commonly available to all users; for example, a compiler comprises of a set of procedures for compiling a user language program into an S-language program. Operating system programs are groups of procedures internal to CS 10110 for allocation and control of CS 10110 resources. Operating system programs also define interfaces within CS 10110. For example, as will be discussed further below all operands in a program are referred to by "NAME". An operating system program translates operand NAME into the physical locations of the operands in MEM 10112. The NAME translation program thus defines the interface between operand NAME (name space addresses) and MEM 10112 physical addresses.

A *process* is an independent locus of control passing through physical, logical or virtual address spaces, or, more particularly, a path of execution through a series of programs (i.e., procedures). A process will generally include a user program and data plus one or more utility programs (e.g., a compiler) and operating system programs necessary to execute the user program.

An *object* is a uniquely identifiable portion of "data space" accessible to CS 10110. An object may be regarded as a container for information and may contain data or procedure information or both. An object may contain for example, an entire program, or set of procedures, or a single bit of data. Objects need not

Petitioner Apple Inc. - Exhibit 1004, p. 5195

be contiguously located in the data space accessible to CS 10110, and the information contained in an object need not be contiguously located in that object.

A *domain* is a state of operation of CS 10110 for the purposes of CS 10110's protection mechanisms. Each domain is defined by a set of procedures having access to objects within that domain for their execution. Each object has a single domain of execution in which it is executed if it is a procedure object, or used, if it is a data object. CS 10110 is said to be operating in a particular domain if it is executing a procedure having that domain of execution. Each object may belong to one or more domains; an object belongs to a domain if a procedure executing in that domain has potential access to the object. CS 10110 may, for example have four domains: User domain, Data Base Management System (DBMS) domain, Extended Operating System (EOS) domain, and Kernel Operating System (KOS) domain. User domain is the domain of execution of all user provided procedures, such as user or utility procedures. DBMS domain is the domain of execution for operating system procedures for storing, retrieving, and handling data. EOS domain is the domain of execution of operating system procedures defining and forming the user level interface with CS 10110, such as procedures for controlling an executing files, processes, and I/O operations. KOS domain is the domain of execution of the low level, secure operating system which manages and controls CS 10110's physical resources. Other embodiments of CS 10110 may have fewer or more domains than those just described. For example, DBMS procedures may be incorporated into the EOS domain or EOS domain may be divided by incorporating the I/O procedures into an I/O domain. There is no hardware enforced limitation on the number of, of boundaries between, domains in CS 10110. Certain CS 10110 hardware functions and structures are, however, dependent upon domains.

A *subject* is defined, for purposes of CS 10110's protection mechanisms, as a combination of the current principle (user), the current process being executed, and the domain the process is currently being executed in. In addition to principle, process, and domain, which are identified by UIDs, subject may include a Tag, which is a user assigned identification code used where added security is required. For a given process, principle and process are constant but the domain is determined by the procedure currently being executed. A process's associated subject is therefore variable along the path of execution of the process.

Having discussed and defined the above terms, certain features of CS 10110 will next be briefly described.

d. Multi-Program Operation

CS 10110 is capable of concurrently executing two or more programs and selecting the sequence of execution of programs to make most effective use of CS 10110's resources. This is referred to as multiprogramming. In this regard, CS 10110 may temporarily suspend execution of one program, for example when a resource or certain information required for that program is not immediately available, and proceed to execute another program until the required resource or information becomes available. For example, particular information required by a first program may not be available in MEM 10112 when called for. JP 10114 may, as discussed further below, suspend execution of the first program, transfer a request for that information to IOS 10116, and proceed to call and execute a second program. IOS 10116 would fetch the requested information from ED 10124 and transfer it into MEM 10112. At some time after IOS 10116 notifies JP 10114 that the requested information is available in MEM 10112, JP 10114 could suspend execution of the second program and resume execution of the first program.

e. Multi-Language Operation

As previously described, CS 10110 is a multiple language machine. Each program written in a high level user language, such as COBOL or FORTRAN, is compiled into a corresponding Soft (S) Language program. That is, in terms of a conventional computer system, each user level language has a corresponding machine language, classically defined as an assembly language. In contrast to classical assembly languages, S-Languages are mid-level languages wherein each command in a user's high level language is replaced by, in general, two or three S-Language instructions, referred to as SINs. Certain SINs may be shared by two or more high level user languages. CS 10110, as further described in following discussions, provides a set, or dialect, of microcode instructions (S-Interpreters) for each S-Language. S-Interpreters interpret SINs and provide corresponding sequences of microinstructions for detailed control of CS 10110. CS 10110's instruction set and operation may therefore be tailored to each user's program, regardless of the particular user language, so as to most efficiently execute the user's program. Computer System 10110 may, for example, execute programs in both FORTRAN and COBOL with comparable efficiency. In addition, a user may write a program in more than one high level user language without loss of efficiency. For example, a user may write a portion of his program in COBOL, but may wish to write certain portions in FORTRAN. In such cases, the COBOL portions would be compiled into COBOL SINs and executed with the COBOL dialect S-Interpreter. The FORTRAN portions would be compiled into FORTRAN SINs and executed with a FORTRAN dialect S-Interpreter. The present embodiment of CS 10110 utilizes a uniform format for all SINs. This feature allows simpler S-Interpreter structures and increases efficiency of SIN interpretation because it is not necessary to provide means for interpreting each dialect individually.

Petitioner Apple Inc. - Exhibit 1004, p. 5196

f. Addressing Structure

Each object created for use in, or by operation of, a CS 10110 is permanently assigned a Unique Identifier (UID). An object's UID allows that object to be uniquely identified and located at any time, regardless of which particular CS 10110 it was created by or for or where it is subsequently located. Thus each time a new object is defined, a new and unique UID is allocated, much as social security numbers are allocated to individuals. A particular piece of information contained in an object may be located by a logical address comprising the object's UID, an offset from the start of the object of the first bit of the segment, and the length (number of bits) of the information segment. Data within an object may therefore be addressed on a bit granular basis. As will be described further in following discussions, UID's are used within a CS 10110 as logical addresses, and, for example, as pointers. Logically, all addresses and pointers in CS 10110 are UID addresses and pointers. As previously described and as described below, however, short, temporary unique identifiers, valid only within JP 10114 and referred to as Active Object Numbers are used within JP 10114 to reduce the width of address buses and amount of address information handled.

An object becomes active in CS 10110 when it is transferred from backing store CED 10124 to MEM 10112 for use in executing a process. At this time, each such object is assigned an Active Object Number (AON). AONs are short unique identifiers and are related to the object's UIDs through certain CS 10110 information structures described below. AONs are used only within JP 10114 and are used in JP 10114, in place of UIDs, to reduce the required width of JP 10114's address buses and the amount of address data handled in JP 10114. As with UID logical addresses, a piece of data in an object may be addressed through a bit granular AON logical address comprising the object's AON, an offset from the start of the object of the first bit of the piece, and the length of the piece.

The transfer of logical addresses, for example pointers, between MEM 10112 (UIDA) and JP 10114 (AONs) during execution of a process requires translations between UIDs and AONs. As will be described in a later discussion, this translation is accomplished, in part, through the information structures mentioned above. Similarly, translation of logical addresses to physical addresses in MEM 10112, to physically access information stored in MEM 10112, is accomplished through CS 10110 information structures relating AON logical addresses to MEM 10112 physical addresses.

Each operand appearing in a program is assigned a Name when the program is compiled. Thereafter, all references to the operands are through their assigned Names. As will be described in detail in a later discussion, CS 10110's addressing structure includes a mechanism for recognizing Names as they appear in an Instruction stream and Name Tables containing directions for resolving Names to AON logical addresses. AON logical addresses may then be evaluated, for example translated into a MEM 10112 physical address, to provide actual operands. The use of Names to identify operands in the instructions stream (process) (1) allows a complicated address to be replaced by a simple reference of uniform format; (2) does not require that an operation be directly defined by data type in the instruction stream; (3) allows repeated references to an operand to be made in an instruction stream by merely repeating the operand's Name; and, (4) allows partially completed Name to address translations to be stored in a cache to speed up operand references. The use of Names thereby substantially reduces the volume of information required in the Instruction stream for operand references and increases CS 10110 speed and efficiency by performing operands references through a parallel operating, underlying mechanism.

Finally, CS 10110 address structure incorporates a set of Architectural Base Pointers (ABPs) for each process. ABPs provide an addressing framework to locate data and procedure information belonging to a process and are used, for example, in resolving Names to AON logical addresses.

g. Protection Mechanism

CS 10110's protection mechanism is constructed to prevent a user from (1) gaining access to or disrupting another user's process, including data, and (2) interfering with or otherwise subverting the operation of CS 10110. Access rights to each particular active object are dynamically granted as a function of the currently active subject. A subject is defined by a combination of the current principle (user), the current process being executed, and the domain in which the process is currently being executed. In addition to principle, process, and domain, subject may include a Tag, which is a user assigned identification code used where added security is required. For a given process, the principle and process are constant but the domain is determined by the procedure currently being executed. A process's associated subject is therefore variable along the path of execution of the process.

In a present embodiment of CS 10110, procedures having KOS domain of execution have access to objects in KOS, EOS, DBMS, and User domains; procedures having EOS domain of execution have access to objects in EOS, DBMS, and User domains; procedures having DBMS domain of execution have access to objects in DBMS and User domains; and procedures having User domain of execution have access only to objects in User domain. A user cannot, therefore, obtain access to objects in KOS domain of execution and cannot influence CS 10110's low level, secure operating system. The user's process may, however, call for execution a procedure having KOS domain of execution. At this point the process's subject is in the KOS domain and the procedure will have access to certain objects in KOS domain.

In a present embodiment of CS 10110, also described in a later discussion, each object has associated with it an Access Control List (ACL). An ACL contains an Access Control Entry (ACE) for each subject having access to that object. ACEs specify, for each subject, access rights a subject has with regard to that object.

Petitioner Apple Inc. - Exhibit 1004, p. 5197

There is normally no relationship, other than that defined by an object's ACL, between subjects and objects. CS 10110, however, supports Extended Type Objects having Extended ACLs wherein a user may specifically define which subjects have what access rights to the object.

In another embodiment of CS 10110, described in a following discussion, access rights are granted on a dynamic basis. In executing a process, a procedure may call a second procedure and pass an argument to the called procedure. The calling procedure will also pass selected access rights to that argument to the called procedure. The passed access rights exist only for the duration of the call.

In the dynamic access embodiment, access rights are granted only at the time they are required. In the ACL embodiment, access rights are granted upon object creation or upon specific request. In either embodiment, each procedure to which arguments may be passed in a cross-domain call has associated with it an Access Information Array (AIA). A procedure's AIA states what access rights a calling procedure (subject) must have before the called procedure can operate on the passed argument. CS 10110's protection mechanisms compare the calling procedure's access rights to the rights required by the called procedure. This ensures that a calling procedure may not ask a called procedure to do what the calling procedure is not allowed to do. Effectively, a calling procedure can pass to a called procedure only the access rights held by the calling procedure.

Having described the general structure and operation and certain features of CS 10110, those and other features of CS 10110 operation will next be described in greater detail.

B. Computer System 10110 Information Structures and Mechanisms (Figs. 102, 103, 104, 105)

CS 10110 contains certain information structures and mechanisms to assist in efficient execution of processes. These structures and mechanisms may be considered as falling into three general types. The first type concerns the processes themselves, i.e., procedure and data objects comprising a user's process or directly related to execution of a user's process. The second type are for management, control, and execution of processes. These structures are generally shared by all processes active in CS 10110. The third type are CS 10110 micromachine information structures and mechanisms. These structures are concerned with the eternal operation of the CS 10110 micromachine and are private to the CS 10110 micro-machine.

a. Introduction (Fig. 102)

Referring to Fig. 102, a pictorial representation of CS 10110 (MEM 10112, FU 10120, and EU 10122) is shown with certain information structures and mechanisms depicted therein. It should be understood that these information structures and mechanisms transcend or "cut across" the boundaries between MEM 10112, FU 10120, EU 10122, and IOS 10116. Referring to the upper portion of Fig. 103 Process Structures 10210 contains those information structures and mechanisms most closely concerned with individual processes, the first and third types of information structures described above. Process Structures 10210 reside in MEM 10112 and Virtual Processes 10212 include Virtual Processes (VP) 1 through N. Virtual Processes 10212 may contain, in a present embodiment of CS 10110, up to 256 VP's. As previously described, each VP includes certain objects particular to a single user's process, for example stack objects previously described and further described in a following description. Each VP also includes a Process Object containing certain information required to execute the process, for example pointers to other process information.

Virtual Processor State Blocks (VPSBs) 10218 include VPSBs containing certain tables and mechanisms for managing execution of VPs selected for execution by CS 10110.

A particular VP is bound into CS 10110 when a Virtual Process Dispatcher, described in a following discussion selects that VP as eligible for execution. The selected VPs Process Object, as previously described, is swapped into a VPSB. VPSBs 10218 may contain, for example 16 or 32 State Blocks so that CS 10110 may concurrently execute to 16 or 32 VPs. When a VP assigned to a VPSB is to be executed, the VP is swapped onto the information structures and mechanisms shown in FU 10120 and EU 10122. FU Register and Stack Mechanism (FURSM) 10214 and EU Register and Stack Mechanism (EURSM) 10216, shown respectively in FU 10120 and EU 10122, comprise register and stack mechanisms used in execution of VPs bound to CS 10110. These register and stack mechanisms, as will be discussed below, are also used for certain CS 10110 process management functions. Procedure Objects (POs) 10213 contains Procedure Objects (POs) 1 to N of the processes executing in CS 10110.

Addressing Mechanisms (AM) 10220 are a part of CS 10110's process management system and are generally associated with Computer System 10110 addressing functions as described in following discussions. UID/AON Tables 10222 is a structure for relating UID's and AON's, previously discussed. Memory Management Tables 10224 includes structures for (1) relating AON logical addresses and MEM 10112 physical addresses; (2) managing MEM 10112's physical address space; (3) managing transfer of information between MEM 10112 and CS 10110's backing store (ED 10124) and, (4) activating objects into CS 10110; Name Cache (NC) 10226 and Address Translation Cache (ATC) 10228 are acceleration mechanisms for storing addressing information relating to the VP currently bound to CS 10110. NC 10226, described further below, contains information relating operand Names to AON addresses. ATC 10228, also discussed further below, contains information relating AON addresses to MEM 10112 physical addresses.

Protection Mechanisms 10230, depicted below AM 10220, include protection Tables 10232 and Protection Cache (PC) 10234. Protection Tables 10232 contain information regarding access rights to each

Petitioner Apple Inc. - Exhibit 1004, p. 5198

object active in CS 10110. PC 10234 contains protection information relating to certain objects of the VP currently bound to CS 10110.

Microinstruction Mechanisms 10236, depicted below PM 10230, includes Micro-code (M Code) Store 10238, FU (Micro-code) M Code Structure 10240, and EU Micro-code (M Code) Structure 10242. These structures contain microinstruction mechanisms and tables for interpreting SINs and controlling the detailed operation of CS 10110. Micro-instruction Mechanisms 10232 also provide microcode tables and mechanisms used, in part, in operation of the low level, secure operating system that manages and controls CS 10110's physical resources.

Having thus briefly described certain CS 10110 information structures and mechanisms with the aid of Fig. 102, those information structures and mechanisms will next be described in further detail in the order mentioned above. In these descriptions it should be noted that, in representation of MEM 10112 shown in Fig. 102 and in other figures of following discussions, the addressable memory space of MEM 10112 is depicted. Certain portions of MEM 10112 address space have been designated as containing certain information structures and mechanisms. These structures and mechanisms have real physical existence in MEM 10112, but may vary in both location and volume of MEM 10112 address space they occupy. Assigning position of a single, large memory to contain these structures and mechanisms allows these structures and mechanisms to be reconfigured as required for most efficient operation of CS 10110. In an alternate embodiment, physically separate memories may be used to contain the structures and mechanisms depicted in MEM 10112, rather than assigned portions of a single memory.

b. Process Structure 10210 (Figs. 103, 104, 105)

Referring to Fig. 103, a partial schematic representation of Process Structures 10210 is shown. Specifically, Fig. 103 shows a Process (P) 10310 selected for execution, and its associated Procedure Objects (POs) in Process Objects (POs) 10213. P 10310 is represented in Fig. 103 as including four procedure objects in POs 10213. It is to be understood that this representation is for clarity of presentation; a particular P 10310 may include any number of procedure objects. Also for clarity of presentation, EURSM 10216 is not shown as EURSM 10216 is similar to FURSM 10214. EURSM 10216 will be described in detail in the following detailed discussons of CS 10110's structure and operation.

As previously discussed, each process includes certain data and procedure object. As represented in Fig. 103 for P 10310 the procedure objects reside in POs 10213. The data objects include Static Data Areas and stack mechanisms in P 10310. POs, for example KOS Procedure Object (KOSPO) 10318, contain the various procedures of the process, each procedure being a sequence of SINs defining an operation to be performed in executing the process. As will be described below, Procedure Objects also contain certain information used in executing the procedures contained therein. Static Data Areas (SDAs) are data objects generally reserved for storing data having an existence for the duration of the process. P 10310's stack mechanisms allow stacking of procedures for procedure calls and returns and for swapping processes in and out of JP 10114. Macro-Stacks (MAS) 10328 to 10334 are generally used to store automatic data (data generated during execution of a procedure and having an existence for the duration of that procedure). Although shown as separate from the stacks in P 10310, the SDAs may be contained with MASs 10328 to 10334. Secure Stack (SS) 10336 stores, in general, CS 10110 micro-machine state for each procedure called. Information stored in SS 10336 allows machine state to be recovered upon return from a called procedure, or when binding (swapping) a VP into CS 10110.

As shown in P 10310, each process is structured on a domain basis. A P 10310 may therefore include, for each domain, one or more procedure objects containing procedures having that domain as their domain of execution, an SDA and an MAS. For example, KOS domain of P 10310 includes KOSPO 10318, KOSSDA 10326, and KOSMAS 10334. P 10310's SS 10336 does not reside in any single domain of P 10310, but instead is a stack mechanism belonging to CS 10110 micromachine.

Having described the overall structure of a P 10310, the individual information structures and mechanisms of a P 10310 will next be described in greater detail.

1. Procedure Objects (Fig. 103)

KOSPO 10318 is typical of CS10110 procedure objects and will be referred to for illustration in the following discussion. Major components of KOSPO 10318 are Header 10338, External Entry Descripter (EED) Area 10340, Internal Entry Descripter (IED) Area 10342, S-Op Code Area 10344, Procedure Environment Descripter (PED) 10348, Name Table (NT) 10350, and Access Information Array (AIA) Area 10352.

Header 10338 contains certain information identifying PO 10318 and indicating the number of entries in EED area 10340, discussed momentarily.

EED area 10340 and IED area 10342 together contain an Entry Descripter (ED) for each procedure in KOSPO 10318. KOSPO 10318 is represented as containing Procedures 1, 2, and 11, of which Procedure 11 will be used as an example in the present discussion. EDs effectively comprise an index through certain all information in KOSPO 10318 can be located. IEDs form an index to all KOSPO 10318 procedures which may be called only from other procedures contained in KOSPO 10318. EEDs form an index to all KOSPO 10318 procedures which may be called by procedures external to KOSPO 10318. Externally callable procedures are distinguished aid, as described in a following discussion of CS 10110's protection mechanisms, in

Petitioner Apple Inc. - Exhibit 1004, p. 5199

confirming external calling procedure's access rights.

Referring to ED 11, ED for procedure 11, three fields are shown therein. Procedure Environment Descripter Offset (PEDO) field indicates the start, relative to start of KOSPO 10318, of Procedure 11's PED in PED Area 10348. As will be discussed further below, a procedure's PED contains a set of pointers for locating information used in the execution of that procedure. PED Area 10348 contains a PED for each procedure contained in 10318. In the present embodiment of CS 10110, a single PED may be shared by two or more procedures. Code Entry Point (CEP) field indicates the start, relative to Procedure Base Pointer (PBP) which will be discussed below, of Procedure 11's SIN Code and SIN Code Area 10344. Finally, ED 11's Initial Frame Size (IFS) field indicates the required Initial Frame Size of the KOSMAS 10334 frame storing Procedure 11's automatic data.

PED 11, Procedure 11's PED in PED Area 10348, contains a set of pointers for locating information used in execution of Procedure 11. The first entry in PED 11 is a header containing information identifying PED 11. PED 11's Procedure Base Pointer (PBP) entry is a pointer providing a fixed reference from which other information in PO 10318 may be located. In a specific example, Procedure 11's CEP indicates the location, relative to PBP, of the start of Procedure 11's S-Op code in S-Op Code Area 10344. As will be described further below, PBP is a CS 10110 Architectural Base Pointer (ABP). CS 10110's ABP's are a set of architectural pointers used in CS 10110 to facilitate addressing of CS 10110's address space. PED 11's Static Data Pointer (SDP) entry points to data, in PO 10318, specifying certain parameters of P 10310's KOSSDA 10326. Name Table Pointer (NTP) entry is a pointer indicating the location, in NT 10350, of Name Table Entry's (NTE's) for Procedure 11's operands. NT 10350 and NTE's will be described in greater detail in the following discussion of Computer System 10110's Addressing Structure. PED 11's S-Interpreter Pointer (SIP) entry is a pointer, discussed in greater detail in a following discussion of CS 10110's microcode structure, pointing to the particular S-Interpeter (SINT) to be used in interpreting Procedure 11's SIN Code.

Referring finally to AIA 10352, AIA 10352 contains, as previously discussed, information pertaining to access rights required of any external procedure calling a 10318 procedure. There is an AIA 10352 entry for each PO 10318 procedure which may be called by an external procedure. A particular AIA entry may be shared by one or more procedures having an ED in EED Area 10340. Each EED contains certain information, not shown for clarity of presentation, indicating that that procedure's corresponding AIA entry must be referred to, and the calling procedure's access rights confirmed, whenever that procedure is called.

2. Stack Mechanism (Figs. 104, 105)

As previously described, P10310's stack mechanisms include SS 10336, used in part for storing machine state, and MAS's 10328 to 10334, used to store local data generated during execution of P 10310's procedures. P 10310 is represented as containing an MAS for each CS 10110 domain. In an alternate embodiment of CS 10110, a particular P 10310 will include MAS's only for those domains in which that P 10310 is executing a procedure.

Referring to MAS's 10328 to 10334 and SS 10336, P 10310 is represented as having had eleven procedure calls. Procedure 0 has called Procedure 1, Procedure 1 has called Procedure 2, and so on. Each time a procedure is called, a corresponding stack frame is constructed on the MAS of the domain in which the called procedure is executed. For example, Procedures 1, 2, and 11 execute in KOS domain; MAS frames for Procedures 1, 2, and 11 therefore are placed on KOSMAS 10334. Similarly, Procedures 3 and 9 execute in EOS domain, so that their stack frames are placed on EOSMAS 10332. Procedures 5 and 6 execute in DBMS domain, so that their stack frames are placed on DBMSMAS 10330. Procedures 4, 7, 8, and 10 execute in User domain with their stack frames being placed on USERMAS 10328. Procedure 11 is the most recently called procedure and procedure 11's stack frame on KOSMAS 10334 is referred to as the current frame. Procedure 11 is the procedure which is currently being executed when VP 10310 is bound to CS 10110.

SS 10336, which is a stack mechanism of CS 10110 micromachine, contains a frame for each of Procedures 1 to 11. Each SS 10336 frame contains, in part, CS 10110 operating state for its corresponding procedure.

Referring to Fig. 104, a schematic representation of a typical MAS, for example KOSMAS 10334, is shown. KOSMAS 10334 includes Stack Header 10410 and a Frame 10412 for each procedure on KOSMAS 10334. Each Frame 10412 includes a Frame Header 10414, and may contain a Linkage Pointer Block 10416, a Local Pointer Block 10418, and a Local (Automatic) Data Block 10420.

KOSMAS 10334 Stack Header 10410 contains at least the following information:

(1) an offset, relative to Stack Header 10410, indicating the location of Frame Header 10414 of the first frame on KOSMAS 10334;

(2) a Stack Top Offset (STO) indicating location, relative to start of KOSMAS 10334, of the top of KOSMAS 10334; top of KOSMAS 10334 is indicated by pointer STO pointing to the top of the last entry of Procedure 11 Frame 10412's Local Data Block 10420;

(3) an offset, relative to start of KOSMAS 10334, indicating location of Frame Header 10414 of the current top frame of KOSMAS 10334; in Fig. 104 this offset is represented by Frame Pointer (FP), an ABP discussed further below;

(4) the VP 10310's UID;

(5) a UID Pointer indicating location of certain domain environment information, described further in a

Petitioner Apple Inc. - Exhibit 1004, p. 5200

following discussion;

(6) a signaller pointer indicating the location of certain routines for handling certain CS 10110 operating system faults;

(7) a UID pointer indicating location of KOSSDA 10326; and

(8) a frame label sequencer containing pointers to headers of frames in other domains; these pointers are used in executing non-local go-to operations.

KOSMAS 10334 Stack Header 10410 thereby contains information for locating certain important points in KOSMAS 10334's structure, and for locating certain information pertinent to executing procedures in KOS domain.

Each Frame Header 10414 contains at least the following information:

(1) offsets, relative to the Frame Header 10414, indicating the locations of Frame Headers 10414 of the previous and next frames of KOSMAS 10334;

(2) an offset, relative to the Frame Header 10414, indicating the location of the top of that Frame 10412;

(3) information indicating the number of passed arguments contained in that Frame 10412;

(4) a dynamic back pointer, in UID/Offset format, to the previous Frame 10412 if that previous Frame 10412 resides in another domain;

(5) a UID/Offset pointer to the environmental descripter of the procedure calling that procedure;

(6) a frame label sequence containing information indicating the locations of other Frame Headers 10414 in KOSMAS 10334; this information is used to locate other frames in KOSMAS 10334 for the purpose of executing local go-to operations. Frame Headers 10414 thereby contain information for locating certain important points in KOSMAS 10334 structure, and certain data pertinent to executing the associated procedures. In addition, Frame Headers 10414, in combination with Stack Header 10410, contain information for linking the activation records of each VP 10310 MAS, and for linking together the activation records of the individual MAS's.

Linkage Pointer Blocks 10416 contain pointers to arguments passed from a calling procedure to the called procedure. For example, Linkage Pointer Block 10416 of Procedure 11's Frame 10412 will contain pointers to arguments passed to Procedure 11 from Procedure 10. The use of linkage pointers in CS 10110's addressing structure will be discussed further in a following discussion of CS 10110's Addressing Structure. Local Data Pointer Blocks 10418 contain pointers to certain of the associated procedure's local data. Indicated in Fig. 104 is a pointer, Frame Pointer (FP), pointing between top most Frame 10412's Linkage Pointer Block 10416 and Local Data Pointer Block 10418. FP, described further in following discussions, is an ABP to MAS Frame 10412 of the process's current procedure.

Each Frame 10412's Local (Automatic) Data Block 10420 contains certain of the associated procedure's automatic data.

As described above, at each procedure call a MAS frame is constructed on top of the MAS of the domain in which the called procedure is executed. For example, when Procedure 10 calls Procedure 11 a Frame Header 10414 for Procedure 11 is constructed and placed on KOSMAS 10334. Procedure 11's linkage pointers are then generated, and placed in Procedure 11's Linkage Pointer Block 10416. Next Procedure 11's local pointers are generated and placed in Procedure 11's Local Pointer Block 10418. Finally, Procedure 11's local data is placed in Procedure 11's Local Data Block 10420. During this operation, USERMAS 10328's frame label sequence is updated to include an entry pointing to Procedure 11's Frame Header 10414. KOSMAS 10334's Stack Header 10410 is updated with respect to STO to the new top of KOSMAS 10334. Procedure 2's Frame Header 10414 is updated with respect to offset to Frame Header 10414 of Procedure 11 Frame 10412, and with respect to frame label sequence indicating location of Procedure 11's Frame Header 10414. As Procedure 11 is then the current procedure, FP is updated to a point between Linkage Pointer Block 10416 and Local Pointer Block 10418 of Procedure 11's Frame 10412. Also, as will be discussed below, a new frame is constructed on SS 10336 or Procedure 11. CS 10110 will then proceed to execute Procedure 11. During execution of Procedure 11, any further local data generated may be placed on the top of Procedure 11's Local Data Block 10420. The top of stack offset information in Procedure 11's Frame Header 10414 and in KOSMAS 10334 Stack Header 10410 will be updated accordingly.

MAS's 10328 to 10334 thereby provide a per domain stack mechanism for storing data pertaining to individual procedures, thus allowing stacking of procedures without loss of this data. Although structured on a domain basis, MAS's 10328 to 10334 comprise a unified logical stack structure threaded together through information stored in MAS stack and frame headers.

As described above and previously, SS 10336 is a CS 10110 micromachine stack structure for storing, in part, CS 10110 micromachine state for each stacked VP 10310 procedure. Referring to Fig. 105, a partial schematic representation of a SS 10336 Stack Frame 10510 is shown. SS 10336 Stack Header 10512 and Frame Headers 10514 contain information similar to that in MAS Stack Headers 10410 and Frame Headers 10414. Again, the information contained therein locates certain points within SS 10336 structure, and threads together SS 10336 with MAS's 10328 to 10334.

SS 10336 Stack Frame 10510 contains certain information used by the CS 10110 micromachine in executing the VP 10212 procedure with which this frame is associated. Procedure Pointer Block 10516 contains certain pointers including ABPs, used by CS 10110 micromachine in locating information within VP 10310's information structures. Micro-Routine Frames (MRFs) 10518 together comprise Micro-Routine Stack (MRS) 10520 within each SS 10336 Stack Frame 10510. MRS Stack 10520 is associated with the

Petitioner Apple Inc. - Exhibit 1004, p. 5201

internal operation of CS 10110 microroutines executed during execution of the VP 10212 procedure associated with the Stack Frame 10510. SS 10336 is thus a dual function CS 10110 micromachine stack. Pointer Block 10516 entries effectively define an interface between CS 10110 micromachine and the current procedure of the current process. MRS 10520 comprise a stack mechanism for the internal operations of CS 10110 micromachine.

Having briefly described Virtual Processes 10212, FURSM 10214 will be described next. As stated above, EURSM 10216 is similar in operation to FURSM 10214 and will be described in following detailed descriptions of CS 10110 structure and operation.

3. FURSM 10214 (Fig. 103)

Referring again to Fig. 103, FURSM 10214 includes CS 10110 micromachine information structures used internally to CS 10110 micromachine in executing the procedures of a P 10310. When a VP, for example P 10310, is to be executed, certain information regarding that VP is transferred from the Virtual Processes 10212 to FURSM 10214 for use in executing that procedure. In this respect, FURSM 10214 may be regarded as an acceleration mechanism for the current Virtual Process 10212.

FURSM 10214 includes General Register File (GRF) 10354, Micro Stack Pointer Register Mechanism (MISPR) 10356, and Return Control Word Stack (RCWS) 10358. GRF 10354 includes Global Registers (GRs) 10360 and Stack Registers (SRs) 10362. GRs 10360 include Architectural Base Registers (ABRs) 10364 and Micro-Control Registers (MCRs) 10366. Stack Registers 10362 include Micro-Stack (MIS) 10368 and Monitor Stack (MOS) 10370.

Referring first to GRF 10354, and assuming for example that Procedure 11 of P 10310 is currently being executed, GRF 10354 primarily contains certain pointers to P 10310 data used in execution of Procedure 11. As previously disussed, CS 10110's addressing structure includes certain Architectural Base Pointers (ABP's) for each procedure. ABPs provide a framework for excessing CS 10110's address space. The ABPs of each procedure include a Frame Pointer (FP), a Procedure Base Pointer (PBP), and a Static Data Pointer (STP). As discussed above with reference to KOSPO 10318, these ABPs reside in the procedure's PEDs. When a procedure is called, these ABP's are transferred from that procedure's PED to ABR's 10364 and reside therein for the duration of that procedure. As indicated in Fig. 103, FP points between Linkage Pointer Block 10416 and Local pointer Blocks 10418 of Procedure 11's Frame 10412 on KOSMAS 10334. PBP points to the reference point from which the elements of KOSPO 10318 are located. SDP points to KOSSDA 10326. If Procedure 11 calls, for example, a Procedure 12, Procedure 11's ABPs will be transferred onto Procedure Pointer Block 10516 of SS 10336 Stack Frame 10510 for Procedure 11. Upon return to Procedure 11, Procedure 11's ABPs will be transferred from Procedure Pointer Block 10516 to ABR's 10364 and execution of Procedure 11 resumed.

MCRs 10336 contain certain pointers used by CS 10110 micromachine in executing Procedure 11. CS 10110 micromachine pointers indicated in Fig. 103 include Program Counter (PC), Name Table Pointer (NTP), S-Interpreter Pointer (SIP), Secure Stack Pointer (SSP), and Secure Stack Top Offset (SSTO). NTP and SIP have been previously described with reference to KOSPO 10318 and reside in KOSPO 10318. NTP and SIP are transferred into MCR's 10366 at start of execution of Procedure 11. PC, as indicated in Fig. 103, is a pointer to the Procedure 11 SIN currently being executed by CS 10110. PC is initially generated from Procedure 11's PBP and CEP and is thereafter incremented by CS 10110 micromachine as Procedure 11's SIN sequences are executed. SSP and SSTO are, as described in a following discussion, generated from information contained in SS 10336's Stack Header 10512 and Frame Headers 10514. As indicated in Fig. 103 SSP points to start of SS 10336 while SSTO indicates the current top frame on SS 10336, whether Procedure Pointer Block 10516 or a MRF 10518 of MRS 10520, by indicating an offset relative to SSP. If Procedure 11 calls a subsequent procedure, the contents of MCR's 10366 are transferred into Procedure 11's Procedure Pointer Block 10516 on SS 10336, and are returned to MCR's 10366 upon return to Procedure 11.

Registers 10360 contain further pointers, described in following detailed discussions of CS 10110 operation, and certain registers which may be used to contain the current procedure's local data.

Referring now to Stack Registers 10362, MIS 10368 is an upward extension, or acceleration, of MRS 10520 of the current procedure. As previously stated, MRS 10520 is used by CS 10110 micromachine in executing certain microroutines during execution of a particular procedure. MIS 10368 enhances the efficiency of CS 10110 micromachine in executing these microroutines by accelerating certain most recent MRFs 10518 of that procedure's MRS 10520 into FU 10120. MIS 10368 may contain, for example, up to the eight most recent MRFs 10518 of the current procedures MRS 10520. As various microroutines are called or returned from, MRS 10520 MRF's 10518 are transferred accordingly between SS 10336 and MIS 10368 so that MIS 10368 always contains at least the top MRF 10518 of MRS 10520, and at most eight MRFs 10518 of MRS 10520. MISPR 10356 is a CS 10110 micromachine mechanism for maintaining MIS 10368. MISPR 10356 contains a Current Pointer, a Previous Pointer, and a Bottom Pointer. Current Pointer points to the top-most MRF 10518 on MIS 10368. Previous Pointer points to the previous MRF 10518 on MIS 10368, and Bottom Pointer points to the bottom-most MRF 10518 on MIS 10368. MISPR 10356's Current, Previous and Bottom Pointers are updated as MRFs 10518 are transferred between SS 10336 and MIS 10368. If Procedure 11 calls a subsequent procedure, all Procedure 11 MRFs 10518 are transferred from MIS 10368 to Procedure 11's MRS 10520 on SS 10336. Upon return to Procedure 11, up to seven of Procedure 11's MRFs 10518

Petitioner Apple Inc. - Exhibit 1004, p. 5202

frames are returned from SS 10336 to MIS 10368.

Referring to MOS 10370, MOS 10370 is a stack mechanism used by CS 10110 micromachine for certain microroutines for handling fault or error conditions. These microroutines always run to completion, so that MOS 10370 resides entirely in FM 10120 and is not an extension of a stack residing in a P 10310 in MEM 10112. MOS 10370 may contain, for example, eight frames. If more than eight successive fault or error conditions occur, this is regarded as a major failure of CS 10110. Control of CS 10110 may then be transferred to DP 10118. As will be described in a following discussion, diagnostic programs in DP 10118 may then be used to diagnose and locate the CS 10110 faults or errors. In other embodiments of CS 10110 MOS 10370 may contain more or fewer stack frames, depending upon the degree of self diagnosis and correction capability desired for CS 10110.

RCWS 10358 is a two-part stack mechanism. A first part operates in parallel with MIS 10368 and a second part operates in parallel with MOS 10370. As previously described, CS 10110 is a microcode controlled system. RCWS is a stack for storing the current microinstruction being executed by CS 10110 micromachine when the current procedure is interrupted by a fault or error condition, or when a subsequent procedure is called. That portion of RCWS 10358 associated with MIS 10368 contains an entry for each MRF 10518 residing in MIS 10368. These RCWS 10358 entries are transferred between SS 10336 and MIS 10368 in parallel with their associated MRFs 10518. When resident in SS 10336, these RCWS 10358 entries are stored within their associated MRFs 10518. That portion of RCWS 10358 associated with MOS 10370 similarly operates in parallel with MOS 10370 and, like MOS 10370, is not an extension of an MEM 10112 resident stack.

In summary, each process active in CS 10110 exists as a separate, complete, and self-contained entity, or Virtual Process, and is structurally organized on a domain basis. Each Virtual Process includes, besides procedure and data objects, a set of MAS's for storing local data of that processes procedures. Each Virtual Process also includes a CS 10110 micromachine stack, SS 10336, for storing CS 10110 micromachine state pertaining to each stacked procedure of the Virtual Process. CS 10110 micromachine includes a set of information structures, register 10360, MIS 10368, MOS 10370, and RCWS 10358, used by CS 10110 micromachine in executing the Virtual Process's procedures. Certain of these CS 10110 micromachine information structures are shared with the currently executing Virtual Process, and thus are effectively acceleration mechanisms for the current Virtual Process, while others are completely internal to CS 10110 micromachine.

A primary feature of CS 10110 is that each process' macrostacks and secure stack resides in MEM 10112. CS 10110's macrostack and secure stacks are therefore effectively unlimited in depth.

Yet another feature of CS 10110 micromachine is the use of GRF 10354. GRF 10354 is, in an embodiment of CS 10110, a unitary register array containing for example, 256 registers. Certain portions, or address locations, of GRF 10354 are dedicated to, respectively, GRs 10360, MIS 10368, and MOS 10370. The capacities of GR 10360, MIS 10368, and MOS 10370, may therefore be adjusted, as required for optimum CS 10110 efficiency, by reassignment of GRF 10354's address space. In other embodiments of CS 10110, GRs 10360, MIS 10368, and MOS 10370 may be implemented a functionally separate registers arrays.

Having briefly described the structure and operation of Process Structures 10210, VP State Block 10218 will be described next below.

C. Virtual Processor State Blocks and Virtual Process Creation (Fig. 102)

Referring again to Fig. 102, VP State Blocks 10218 is used in management and control of processes. VP State Blocks 10218 contains a VP State Block for each Virtual Process (VP) selected for execution by CS 10110. Each such VP State Block contains at least the following information: (1) the state, or identification number of a VP;

(2) entries identifying the particular principle and particular process of the VP;

(3) an AON pointer to that VP's secure stack (e.g., SS 10336);

(4) the AON's of that VP's MAS stack objects (e.g., MAS's 10328 to 10334); and,

(5) certain information used by CS 10110's VP Management System.

The information contained in each VP State Block thereby defines the current state of the asociated VP.

A Process is loaded into CS 10110 by building a primitive access record and loading this access record into CS 10110 to appear as an already existing VP. A VP is created by creating a Process Object, including pointers to macro- and secure-stack objects created for that VP, micromachine state entries, and a pointer to the user's program. CS 10110's KOS then generates Macro- and Secure-Stack Objects with headers for that process and, as described further below, loads protection information regarding that process' objects into protection Structures 10230. CS 10110's KOS then copies this primitive machine state record into a vacant VPSB selected by CS 10110's VP Manager, thus binding the newly created VP into CS 10110. At that time a KOS Initializer procedure completes creation of the VP for example by calling in the user's program through a compiler. The newly creatd VP may then be executed by CS 10110.

Having briefly described VP State Blocks 10218 and creation of a VP, CS 10110's Addressing Structures 10220 will be decribed next below.

Petitioner Apple Inc. - Exhibit 1004, p. 5203

D. Addressing Structure 10220 (Figs. 103, 106, 107, 108)

1. Objects, UID's, AON's, Names, and Physical Addresses (Fig. 106)

As previously described, the data space accessible to CS 10110 is divided into segments, or containers, referred to as objects. In an embodiment of CS 10110, the addressable data space of each object has a capacity of 2^{32} bits of information and is structured into 2^{18} pages with each page containing 2^{14} bits of information.

Referring to Fig. 106A, a schematic representation of CS 10110's addressing structure is shown. Each object created for use in, or by operation of, a CS 10110 is permanently assigned a unique identifier (UID). An object's UID allows an object to be uniquely identified and located at any future point in time. Each UID is an 80 bit number, so that the total addressable space of all CS 10110's includes 2^{80} objects wherein each object may contain up to 2^{32} bits of information. As indicated in Fig. 106, each 80 bit UID is comprised of 32 bits of Logical Allocation Unit Identifier (LAUID) and 48 bits of Object Serial Number (OSN). LAUIDs are associated with individual CS 10110 systems. LAUIDs identify the particular CS 10110 system generating a particular object. Each LAUID is comprised of a Logical Allocation Unit Group Number (LAUGN) and a Logical Allocation Unit Serial Number (LAUSN). LAUGNs are assigned to individual CS 10110 systems and may be guaranteed to be unique to a particular system. A particular system may, however, be assigned more than one LAUGN so that there may be a time varying mapping between LAUGNs and CS 10110 systems. LAUSNs are assigned within a particular system and, while LAUSNs may be unique within a particular system, LAUSNs need not be unique between systems and need not map onto the physical structure of a particular system.

OSNs are associated with individual objects created by an LAU and are generated by an Architectural Clock in each CS 10110. Architectural clock is defined as a 64 bit binary number representing increasing time. Least significant bit of architectural clock represents increments of 600 picoseconds, and most significant bit represents increments of 127 years. In the present embodiment of CS 10110, certain most significant and least significant bits of architectural clock time are disregarded as generally not required practice. Time indicated by architectural clock is measured relative to an arbitrary, fixed point in time. This point in time is the same for all CS 10110s which will ever be constructed. All CS 10110s in existence will therefore indicate the same architectural clock time and all UIDs generated will have a common basis. The use of an architectural clock for generation of OSNs is advantageous in that it avoids the possibility of accidental duplication of OSNs if a CSC 10110 fails and is subsequently reinitiated.

As stated above, each object generated by or for use in a CS 10110 is uniquely identified by its associated UID. By appending Offset (O) and Length (L) information to an object's UID, a UID logical address is generated which may be used to locate particular segments of data residing in a particular object. As indicated in Fig. 106, O and L fields of a UID logical address are each 32 bits. O and L fields can therefore indicate any particular bit, out of 2^{32-1} bits, in an object and thus allow bit granular addressing of information in objects.

As indicated in Fig. 106 and as previously described, each object active in CS 10110 is assigned a short temporary unique identifier valid only within JP 10114 and referred to as an Active Object Number (AON). Because fewer objects may be active in a CS 10110 than may exist in a CS 10110's address space, AON's are, in the present embodiment of CS 10110, 14 bits in length. A particular CS 10110 may therefore contain up to 2^{14} active objects. An object's AON is used within JP 10114 in place of that object's UID. For example, as discussed above with reference to process structures 10210, a procedure's FP points to start of that procedure's frame on its process' MAS. When that FP is residing in SS 10336, it is expressed as a UID. When that procedure is to be executed, FP is transferred from SS 10336 to ABR's 10364 and is translated into the corresponding AON. Similarly, when that procedure is stacked, FP is returned to SS 10336 and in doing so is translated into the corresponding UID. Again, a particular data segment in an object may be addressed by means of an AON logical address comprising the object's AON plus associated 32 bit Offset (O) and Length (L) fields.

Each operand appearing in a process is assigned a Name and all references to a process's operands are through those assigned Names. As indicated in Fig. 106B, in the present embodiment of CS 10110 each Name is an 8, 12, or 16 bit number. All Names within a particular process will be of the same length. As will be described in a following discussion, Names appearing during execution of a process may be resolved, through a procedure's Name Table 10350 or through Name Cache 10226, to an AON logical address. As described below, an AON logical address corresponding to an operand Name may then be evaluated to a MEM 10112 physical address to locate the operand referred to.

The evaluation of AON logical addresses to MEM 10112 physical addresses is represented in Fig. 106C. An AON logical address's L field is not involved in evaluation of an AON logical address to a physical address and, for purposes of clarity of presentation, is therefore not represented in Fig. 106C. AON logical address L field is to be understood to be appended to the addresses represented in the various steps of the evaluation procedure shown in Fig. 106C.

As described above, objects are 2^{32} bits structured into 2^{18} pages with each page containing a 2^{14} bits of data. MEM 10112 is similarly physically structured into frames with, in the present embodiment of CS 10110, each frame containing 2^{14} bits of data. In other embodiments of CS 10110, both pages and frames may be of different sizes but the translation of AON logical addresses to MEM 10112 physical addresses will be similar to that described momentarily.

Petitioner Apple Inc. - Exhibit 1004, p. 5204

An AON logical address O field was previously described as a 32 bit number representing the start, relative to start of the object, of the addressed data segment within the object. The 18 most significant bits of O field represent the number (P) of the page within the object upon which the first bit of the addressed data occurs. The 14 least significant bits of O field represent the offset (O_p), relative to the start of the page, within that page of the first bit of the addressed data. AON logical address O field may therefore, as indicated in Fig. 106C, be divided into an 18 bit page (P) field and a 14 bit offset within page (O_p) field. Since, as described above, MEM 10112 physical frame size is equal to object page size, AON logical address O_p field may be used directly as an offset within frame (O_f) field of the physical address. As will be described below, an AON logical address AON and P fields may then be related to the frame number (FN) of the MEM 10112 frame in which that page resides, through Addressing Mechanisms 10220.

Having briefly described the relationships between UIDs, UID Logical Addresses, Names, AONs, AON Logical Addresses, and MEM 10112 Physical Addresses, Addressing Mechanisms 10220 will be described next below.

2. Addressing Mechanisms 10220 (Fig. 107)

Referring to Fig. 107, a schematic representation of Computer System 10110's Addressing Mechanisms 10220 is shown. As previously described, Addressing Mechanisms 10220 comprise UID/AON Tables 10222, Memory Management Tables 10224, Name Cache 10226, and Address Translation Unit 10228.

UID/AON Tables 10222 relate each object's UID to its assigned AON and include AOT Hash Table (AOTHT) 10710, Active Object Table (AOT) 10712, and Active Object Table Annex (AOTA) 10714.

An AON corresponding to a particular UID is determined through AOTHT 10710. The UID is hashed to provide a UID index into AOTHT 10710, which then provides the corresponding AON. AOTHT 10710 is effectively an acceleration mechanism of AOT 10712 to, as just described, provide rapid translation of UIDs to AONs. AONs are used as indexes into AOT 10712, which provides a corresponding AOT Entry (AOTE). An AOTE as described in following detailed discussions of CS 10110, includes, among other information, the UID corresponding to the AON indexing the AOTE. In addition to providing translation between AONs and UIDs, the UID of an AOTE may be compared to an original UID to determine the correctness of an AON from AOTHT 10710.

Associated with AOT 10712 is AOTA 10714. AOTA 10714 is an extension of AOT 10712 and contains certain information pertaining to active objects, for example the domain of execution of each active procedure object.

Having briefly described CS 10110's mechanism for relating UIDs and AONs, CS 10110's mechanism for resolving operand Names to AON logical addresses will be described next below.

3. Name Resolution (Figs. 103, 108)

Referring first to Fig. 103, each procedure object in a VP, for example KOSPO 10318 in VP 10310, was described as containing a Name Table (NT) 10350. Each NT 10350 contains a Name Table Entry (NTE) for each operand whose Name appears its procedure. Each NTE contains a description of how to resolve the corresponding Name to an AON Logical Address, including fetch mode information, type of data referred to by that Name, and length of the data segment referred to.

Referring to Fig. 108, a representation of an NTE is shown. As indicated, this NTE contains seven information fields: Flag, Base (B), Predisplacement (PR), Length (L), Displacement (D), Index (I), and Inter-element Spacing (IES). Flag Field, in part, contains information describing how the remaining fields of the NTE are to be interpreted, type of information referred to by the NTE, and how that information is to handled when fetched from MEM 10112. L Field, as previously described, indicates length, or number of bits in, the data segment. Functions of the other NTE fields will be described during the following discussions.

In a present embodiment of CS 10110, there are five types of NTE: (1) base (B) is not a Name, address resolution is not indirect; (2) B is not a Name, address resolution is indirect; (3) B is a Name, address resolution is indirect; (4) B is a Name, address resolution is indirect. A fifth type is an NTE selecting a particular element from an array of elements. These five types of NTE and their resolution will be described below, in the order mentioned.

In the first type, B is not a Name and address resolution is not indirect, B Field specifies an ABR 10364 containing an AON plus offset (AON/O) Pointer. The contents of D Field are added to the O Field of this pointer, and the result is the AON Logical Address of the operand. In the second type, B is not a Name and address resolution is indirect, B Field again specifies an ABR 10364 containing an AON/O pointer. The contents of PR Field are added to the O Field of the AON/O pointer to provide an AON Logical Address of a Base Pointer. The Base Pointer AON Logical Address is evaluated, as described below, and the Base Pointer fetched from MEM 10112. The contents of D Field are added to the O Field of the Base Pointer and the result is the AON Logical Address of the operand.

NTE types 3 and 4 correspond, respectively to NTE types 1 and 2 and are resolved in the same manner except that B Field contains a Name. The B Field Name is resolved through another NTE to obtain an AON/O pointer which is used in place of the ABR 10364 pointers referred to in discussion of types 1 and 2.

The fifth type of NTE is used in references to elements of an array. These array NTEs are resolved in the

Petitioner Apple Inc. - Exhibit 1004, p. 5205

same manner as NTE types 1 through 4 above to provide an AON Logical Address of the start of the array. I and IES Fields provide additional information to locate a particular element in the array. I Field is always Name which is resolved to obtain an operand value representing the particular element in the array. IES Field provides information regarding spacing between elements of the array, that is the number of bits between adjacent element of the array. IES Field may contain the actual IES value, or it may contain a Name which is resolved to an AON Logical Address leading to the inter-element spacing value. The I and IES values, obtained by resolving the I and IES Fields as just described, are multiplied together to determine the offset, relative to the start of the array, of the particular element referred to by the NTE. This within array offset is added to the O Field of the AON Logical Address of the start of the array to provide the AON Logical Address of the element.

In the current embodiment of CS 10110, certain NTE fields, for example B, D, and Flag fields, always contain literals. Certain other fields, for example, IES, D, PRE, and L fields, may contain either literals or names to be resolved. Yet other fields, for example I field, always contain names which must be resolved.

Passing of arguments from a calling procedure to a called procedure has been previously discussed with reference to Virtual Processes 10212 above, and more specifically with regard to MAS's 10328 to 10334 of VP 10310. Passing of arguments is accomplished through the calling and called procedure's Name Tables 10350. In illustration, a procedure W(a, b, c) may wish to pass arguments a, b, and c to procedure X(u, v, w), where arguments, v and w correspond to arguments a, b, and c. At compilation, NTEs are generated for arguments a, b, and c in procedure W's procedure object, and NTEs are generated for arguments u, v and w in Procedure X's procedure object. Procedure X's NTEs for u, v, and w are constructed to resolve to point to pointers in Linkage Pointer Block 10416 of Procedure X's Frame 10412 in MAS. To pass arguments a, b, and c from Procedure W to Procedure X, the NTEs of arguments a, b, and c are resolved to AON Logical Addresses (i.e., AON/O form). Arguments a, b, and c's AON Logical Addresses are then translated to corresponding UID addresses which are placed in Procedure X's Linkage Pointer Block 10416 at those places pointed to by Procedure X's NTEs for u, v, and w. When Procedure X is executed, the resolution of Procedure X's NTEs for u, v, and w will be resolved to locate the pointers, in Procedure X's Linkage Pointer Block 10416 to arguments a, b, and c. When arguments are passed in this manner, the data type and length information are obtained from the called procedure's NTEs, rather than the calling procedure's NTEs. This allows the calling procedure to pass only a portion of, for example, arguments a, b, or c, to the called procedure and thus may be regarded as a feature of CS 10110's protection mechanisms.

Having briefly described resolution of Names to AON/Offset addresses, and having previously described translation of UID addresses to AON addresses, the evaluation of AON addresses to MEM 10112 physical addresses will be described next below.

4. Evaluation of AON Addresses to Physical Addresses (Fig. 107)

Referring again to Fig. 107, a partial schematic representation of CS 10110's Memory Management Table 10224 is shown. Memory Hash Table (MHT) 10716 and Memory Frame Table (MFT) 10718 are concerned with translation of AON addresses into MEM 10112 physical addresses and will be discussed first. Working Set Matrix (WSM) 10720 and Virtual Memory Manager Request Queue (VMMRQ) 10722 are concerned with management of MEM 10112's available physical address base and will be discussed second. Active Object Request Queue (AORQ) 10728 and Logical Allocation Unit Directory (LAUD) 10730 are concerned with locating inactive objects and management of which objects are active in CS 10110 and will be discussed last.

Translation of AON/O Logical Addresses to MEM 10112 physical addresses was previously discussed with reference to Fig. 106C. As stated in that discussion, objects are divided into pages. Correspondingly, the AON/O Logical Address' O Field is divided into an 18 bit page number (P) Field and a 14 bit offset within a page (O_p) Field. MEM 10112 is structured into frames, each of which in the present embodiment of CS 10110 is equal to a page of an object. An AON/O address' O_p Field may therefore be used directly as an offset within frame (O_f) of the corresponding physical address. The AON and P fields of an AON address must, however, be translated into a MEM 10112 frame represented by a corresponding Frame Number (FN).

Referring now to Fig. 107, an AON address' AON and P Fields are "hashed" to generate an MHT index which is used as an index into MHT 10716. Briefly, "hashing" is a method of indexing, or locating, information in a table wherein indexes to the information are generated from the information itself through a "hashing function". A hashing function maps each piece of information to the corresponding index generated from it through the hashing function. MHT 10716 then provides the corresponding FN of the MEM 10112 frame in which that page is stored. FNs are used as indexes into MFG 10718, which contains, for each FN, an entry describing the page stored in that frame. This information includes the AON and P of the page stored in that MEM 10112 frame. An FN from MHT 10716 may therefore be used as an index into MFT 10718 and the resulting AON/P of MFT 10718 compared to the original AON/P to confirm the correctness of the FN obtained from MHT 10716. MHT 10716 is an effectively acceleration mechanism of MFT 10718 to provide rapid translation of AON address to MEM 10112 physical addresses.

MFT 10718 also stores "used" and "modified" information for each page in MEM 10112. This information indicates which page frames stored therein have been used and which have been modified.

Petitioner Apple Inc. - Exhibit 1004, p. 5206

This information is used by CS 10110 in determining which frames may be deleted from MEM 10112, or are free, when pages are to be written into MEM 10112 from backing store (ED 10124). For example, if a page's modified bit indicates that that page has not been written into, it is not necessary to write that page back into backing store when it is deleted from MEM 10112; instead, that page may be simply erased.

Referring finally to ATU 10228, ATU 10228 is an acceleration mechanism for MHT 10716. AON/O addresses are used directly, without hashing, as indexes into ATU 10228 and ATU 10228 correctly provides corresponding FN and O outputs. A CS 10110 mechanism, described in a following detailed discussion of CS 10110 operation, continually updates the contents of ATU 10228 so that ATU 10228 contain the FN's and O_p (O_i) of the pages most frequently referenced by the current process. If ATU 10228 does not contain a corresponding entry for a given AON input, an ATU fault occurs and the FN and O information may be obtained directly from MHT 10716.

Referring now to WSM 10720 and VMMRQ 10722, as previously stated these mechanisms are concerned with the management of MEM 10112's available address space. For example, if MHT 10716 and MFT 10718 do not contain an entry for a page referenced by the current procedure, an MHT/MFT fault occurs and the reference page must be fetched from backing store (ED 10124) and read into MEM 10112. WSM 10720 contains an entry for each page resident in MEM 10112. These entries are accessed by indexes comprising the Virtual Processor Number (VPN) of the virtual process making a page reference and the P of the page being referenced. Each WSM 10720 entry contains 2 bits stating whether the particular page is part of a VP's working set, that is, used by that VP, and whether that page has been referenced by that VP. This information, together with the information contained in that MFT 10718 entries described above, is used by CS 10110's Virtual Memory Manager (VMM) in transferring pages into and out of MEM 10112.

CS 10110's VMM maintains VMMRQ 10722, which is used by VMM to control transfer of pages into and out of MEM 10112. VMMRQ 10722 includes Virtual Memory Request Counter (VMRC) 10724 and a Queue of Virtual Memory Request Entries (VMREs) 10726. As will be discussed momentarily, VMRC 10724 tracks the number of currently outstanding request for pages. Each VMRE 10726 describes a particular page which has been requested. Upon occurrence of a MHT/MFT (or page) fault, VMRC 10724 is incremented, which initiates operation of CS 10110's VMM, and a VMRE 10726 is placed in the queue. Each VMRE 10726 comprises the VPN of the process requesting the page and the AON/O of the page requested. At this time, the VP making the request is swapped out of JP 10114 and another VP bound to JP 10114. VMM allocates MEM 10112 frame to contain the requested page, using the previously described information in MFT 10718 and WSM 10720 to select this frame. In doing so, VMM may discard a page currently resident in MEM 10112 for example, on the basis of being the oldest page, an unused page, or an unmodified page which does not have to be written back into backing store. VMM then requests an I/O operation to transfer the requested page into the frame selected by the VMM. While the I/O operation is proceeding, VMM generates new entries in MHT 10716 and MFT 10718 for the requested page, cleans the frame in MEM 10112 which is to be occupied by that page, and suspends operation. IOS 10116 will proceed to execute the I/O operation and writes the requested page directly into MEM 10112 in the frame specified by VMM. IOS 10116 then notifies CS 10110's VMM that the page now resides in memory and can be referenced. At some later time, that VP requesting that page will resume execution and repeat that reference. Going first to ATU 10228, that VP will take an ATU 10228 fault since VP 10212 has not yet been updated to contain that page. The VP will then go to MHT 10716 and MFT 10718 for the required information and, concurrently, WSM 10720 and ATU 10228 will be updated.

In regard to the above operations, each VP active in CS 10110 is assigned a Page Fault Frequency Time Factor (PFFT) which is used by CS 10110's VMM to adjust that VP's working set so that the interval between successive page faults for that VP lies in an optimum time range. This assists in ensuring CS 10110's VMM is operating most efficiently and allows CS 10110's VMM to be tuned as required.

The above discussions have assumed that the page being referenced, whether from a UID/O address, an AON/O address, or a Name, is resident in an object active in CS 10110. While an object need not have a page in MEM 10112 to be active, the object must be active to have a page in MEM 10112. A VP, however, may reference a page in an object not active in CS 10110. If such a reference is made, the object must be made active in CS 10110 before the page can be brought into MEM 10112. The result is an operation similar to the page fault operation described above. CS 10110 maintains an Active Object Manager (AOM), including Active Object Request Queue (AORQ) 10728, which are similar in operation to CS 10110's VMM and VMMRQ 10722. CS 10110's AOM and AORQ 10728 operate in conjunction with AOTHT 10710 and AOT 10712 to locate inactive objects and make them active by assigning them AON's and generating entries for them in AOTHT 10710, AOT 10712, and AOTA 10714.

Before a particular object can be made active in CS 10110, it must first be located in backing store (ED 10124). All objects on backing store are located through a Logical Allocation Unit Directory (LAUD) 10730, which is resident in backing store. An LAUD 10730 contains entries for each object accessible to the particular CS 10110. Each LAUD 10730 entry contains the information necessary to generate an AOT 10712 entry for that object. An LAUD 10730 is accessed through a UID/O address contained in CS 10110's VMM. A reference to an LAUD 10730 results in MEM 10112 frames being assigned to that LAUD 10730, and LAUD 10730 being transferred into MEM 10112. If an LAUD 10730 entry exists for the referenced inactive object, the LAUD 10730 entry is transferred into AOT 10712. At the next reference to a page in that object, AOT 10712 will provide the AON for that object but, because the page has not yet been transferred into MEM

Petitioner Apple Inc. - Exhibit 1004, p. 5207

10112, a page fault will occur. This page fault will be handled in the manner described above and the referenced page transferred into MEM 10112.

Having briefly described the structure and operation of CS 10110's Addressing Structure, including the relationship between UIDs, Names, AONs, and Physical Addresses and the mechanisms by which CS 10110 manages the available address space of MEM 10112, CS 10110's protection structures will be described next below.

E. CS 10110 Protection Mechanisms (Fig. 109)

Referring to Fig. 109, a schematic representation of Protection Mechanisms 10230 is shown. Protection Tables 10232 include Active Primitive Access Matrix (APAM) 10910, Active Subject Number Hash Table (ASNHT) 10912, and Active Subject Table (AST) 10914. Those portions of Protection Mechanism 10230 resident in FU 10120 include ASN Register 10916 and Protection Cache (PC) 10234.

As previously discussed, access rights to objects are arbitrated on the basis of subjects. A subject has been defined as a particular combination of a principle, Process, and Domain (PPD), each of which is identified by a corresponding UID. Each object has associated with it an Access Control List (ACL) 10918 containing an ACL Entry (ACLE) for each subject having access rights to that object.

When an object becomes active in CS 10110 (i.e., is assigned an AON) each ACLE in that object's ACL 10918 is written into APAM 10910. Concurrently, each subject having access rights to that object, and for which there is an ACLE in that object's ACL 10918, is assigned an Active Subject Number (ASN). These ASNs are written into ASNHT 10912 and their corresponding PPDs are written into AST 10914. Subsequently, the ASN of any subject requesting access to that object is obtained by hashing the PPD of that subject to obtain a PPD index into ASNHT 10912. ASNHT 10912 will in turn provide a corresponding ASN. An ASN may be used as an index into AST 10914. AST 10914 will provide the corresponding PPD, which may be compared to an original PPD to confirm the accuracy of the ASN.

As described above, APAM 10910 contains an ACL 10918 for each object active in CS 10110. The access rights of any particular active subject to a particular active object are determined by using that subject's ASN and that object's AON as indexes into APAM 10910. APAM 10910 in turn provides a 4 bit output defining whether that subject has Read (R) Write (W) or Execute (E) rights with respect to that object, and whether that particular entry is Valid (V).

ASN Register 10916 and PC 10234 are effectively acceleration mechanisms of Protection Tables 10232. ASN Register 10916 stores the ASN of a currently active subject while PC 10234 stores certain access right information for objects being used by the current process. PC 10234 entries are indexed by ASNs from ASN register 10916 and by a mode input from JP 10114. Mode input defines whether the current procedure intends to read, write, or execute with respect to a particular object having an entry in PC 10234. Upon receiving ASN and mode inputs, PC 10234 provides a go/nogo output indicating whether that subject has the access rights required to execute the intended operation with respect to that object.

In addition to the above mechanism, each procedure to which arguments may be passed in a cross-domain call has associated with it an Access Information Array (AIA) 10352, as discussed with reference to Virtual Processes 10212. A procedure's AIA 10352 states what access rights a calling procedure (subject) must have to a particular object (argument) before the called procedure can operate on the passed argument. CS 10110's protection mechanisms compare the calling procedures access rights to the rights required by the called procedure. This insures the calling procedure may not ask a called procedure to do what the calling procedure is not allowed to do. Effectively, a calling procedure can pass to a called procedure only the access rights held by the calling procedure.

Finally, PC 10234, APAM 10910, or AST 10914 faults (i.e., misses) are handled in the same manner as described above with reference to page faults in discussion of CS 10110's Addressing Mechanisms 10220. As such, the handling of protection misses will not be discussed further at this point.

Having briefly described structure and operation of CS 10110's Protection Mechanisms 10230, CS 10110's Micro-Instruction Mechanisms 10236 will be described next below.

F. CS 10110 Micro-Instruction Mechanism (Fig. 110)

As previously described, CS 10110 is a multiple language machine. Each program written in a high level user language is compiled into a corresponding S-Language program containing instructions expressed as SINs. CS 10110 provides a set, or dialect, of microcode instructions, referred to as S-Interpreters (SINTs) for each S-Language. SINTs interpret SINs and provide corresponding sequences of microinstructions for detailed control of CS 10110.

Referring to Fig. 110, a partial schematic representation of CS 10110's Micro-Instruction Mechanisms 10236 is shown. At system initialization all CS 10110 microcode, including SINTs and all machine assist microcode, is transferred from backing store to Micro-Code Control Store (mCCS) 10238 in MEM 10112. The Micro-Code is then transferred from mCCS 10238 to FU Micro-Code Structure (FUmC) 10240 and EU Micro-Code Structure (EUmC) 10242. EUmC 10242 is similar in structure and operation to FUmC 10240 and thus will be described in following detailed descriptions of CS 10110's structure and operation. Similarly, CS 10110 machine assist microcode will be described in following detailed discussions. The present discussion will concern CS 10110's S-Interpreter mechanisms.

CS 10110's S-Interpreters (SINTs) are loaded into S-Interpret Table (SITT) 11012, which is represented

Petitioner Apple Inc. - Exhibit 1004, p. 5208

in Fig. 110 as containing S-Interpreters 1 to N. Each SIT contains one or more sequences of micro-code; each sequence of microcode corresponds to a particular SIN in that S-Language dialect. S-Interpreter Dispatch Table (SDT) 11010 contains S-Interpreter Dispatchers (SDs) 1 to N. There is one SD for each SINT in SITT 11012, and thus a SD for each S-Language dialect. Each SD comprises a set of pointers. Each pointer in a particular SD corresponds to a particular SIN of that SD's dialect and points to the corresponding sequence of microinstructions for interpreting that SIN in that dialect's SIT in SITT 11012. In illustration, as previously discussed when a particular procedure is being executed the SIP for that procedure is transferred into one of mCR's 10366. That SIP points to the start of the SD for the SIT which is to be used to interpret the SINs of that procedure. In Fig. 110, the SIP in mCRs 10366 is shown as pointing to the start of SD2. Each S-Op appearing during execution of that procedure is an offset, relative to the start of the selected SD, pointing to a corresponding SD pointer. That SD pointer in turn points to the corresponding sequence of microinstructions for interpreting that SIN in the corresponding SIT in SITT 11012. As will be described in following discussions, once the start of a microcode sequence for interpreting an SIN has been selected, CS 10110 micromachine then proceeds to sequentially call the microinstructions of that sequence from SITT 11012 and use those microinstructions to control operation of CS 10110.

G. Summary of Certain CS 10110 Features and Alternate Embodiments

The above Introductory Overview has described the overall structure and operation and certain features of CS 101, that is, CS 10110. The above Introduction has further described the structure and operation and further features of CS 10110 and, in particular, the physical implementation and operation of CS 10110's information, control, and addressing mechanisms. Certain of these CS 10110 features are summarized next below to briefly state the basic concepts of these features as implemented in CS 10110. In addition, possible alternate embodiments of certain of these concepts are described.

First, CS 10110 is comprised of a plurality of independently operating processors, each processor having a separate microinstruction control. In the present embodiment of CS 10110, these processors include FU 10120, EU 10122, MEM 10112 and IOS 10116. Other such independently operating processors, for example, special arithmetic processors such as an array processor, or multiple FU 10120's, may be added to the present CS 10110.

In this regard, MEM 10112 is a multiport processor having one or more separate and independent ports to each processor in CS 10110. All communications between CS 10110's processors are through MEM 10112, so that MEM 10112 operates as the central communications node of CS 10110, as well as performing memory operations. Further separate and independent ports may be added to MEM 10112 as further processors are added to CS 10110. CS 10110 may therefore be described as comprised of a plurality of separate, independent processors, each having a separate microinstruction control and having a separate and independent port to a central communications and memory node which in itself is an independent processor having a separate and independent microinstruction control. As will be further described in a following detailed description of MEM 10112, MEM 10112 itself is comprised of a plurality of independently operating processors, each performing memory related operations and each having a separate microinstruction control. Coordination of operations between CS 10110's processors is achieved by passing "messages" between the processors, for example, SOP's and descriptors.

CS 10110's addressing mechanisms are based, first, upon UID addressing of objects. That is, all information generated for use in or by operation of a CS 10110, for example, data and procedures, is structured into objects and each object is assigned a permanent UID. Each UID is unique within a particular CS 10110 and between all CS 10110's and is permanently associated with a particular object. The use of UID addressing provides a permanent, unique addressing means which is common to all CS 10110's, and to other computer systems using CS 10110's UID addressing.

Effectively, UID addressing means that the address (or memory) space of a particular CS 10110 includes the address space of all systems, for example disc drives or other CS 10110s, to which that particular CS 10110 has access. UID addressing allows any process in any CS 10110 to obtain access to any object in any CS 10110 to which it has physical access, for example, another CS 10110 on the other side of the world. This access is constrained only by CS 10110's protection mechanism. In alternate embodiments of CS 10110, certain UIDs may be set aside for use only within a particular CS 10110 and may be unique only within that particular CS 10110. These reserved UIDs would, however, be a limited group known to all CS 10110 systems as not having uniqueness between systems, so that the unique object addressing capability of CS 10110's UID addressing is preserved.

As previously stated, AONs and physical descriptors are presently used for addressing within a CS 10110, effectively as shortened UIDs. In alternate embodiments of CS 10110, other forms of AONs may be used, or AONs may be discarded entirely and UIDs used for addressing within as well as between CS 10110s.

CS 10110's addressing mechanisms are also based upon the use of descriptors within and between CS 10110s.

Each descriptor includes an AON or UID field to identify a particular object, an offset field to specify a bit granular offset within the object, and a length field to specify a particular number of bits beginning at the specified offset. Descriptors may also include a type, or format field identifying the particular format of the data referred to by the descriptor. Physical descriptors are used for addressing MEM 10112 and, in this

Petitioner Apple Inc. - Exhibit 1004, p. 5209

case, the AON or UID field is replaced by a frame number field referring to a physical location in MEM 10112.

As stated above, descriptors are used for addressing within and between the separate, independent processors (FU 10120, EU 10122, MEM 10112, and IOS 10116) comprising CS 10110, thereby providing common, system wide bit granular addressing which includes format information. In particular, MEM 10112 responds to the type information fields of descriptors by performing formatting operations to provide requestors with data in the format specified by the requestor in the descriptor. MEM 10112 also accepts data in a format specified in a descriptor and reformats that data into a format most efficiently used by MEM 10112 to store the data.

As previously described, all operands are referred to in CS 10110 by Names wherein all Names within a particular S-Language dialect are of a uniform, fixed size and format. A K value specifying Name size is provided to FU 10120, at each change in S-Language dialect, and is used by FU 10120 in parsing Names from the instruction stream. In an alternate embodiment of CS 10110, all Names are the same size in all S-Language dialects, so that K values, and the associated circuitry in FU 10120's parser, are not required.

Finally, in descriptions of CS 10110's use of SOPs, FU 10120's microinstruction circuitry was described as storing one or more S-Interpreters. S-Interpreters are sets of sequences of microinstructions for interpreting the SOPs of various S-Language dialects and providing corresponding sequences of microinstructions to control CS 10110. In an alternate embodiment of CS 10110, these S-Interpreters (SITT 11012) would be stored in MEM 10112. FU 10120 would receive SOPs from the instruction stream and, using one or more S-Interpreter Base Pointers (that is, architectural base pointers pointing to the SITT 11012 in MEM 10112), address the SITT 11012 stored in MEM 10112. MEM 10112 would respond by providing, from the SITT 11012 in MEM 10112, sequences of microinstructions to be used directly in controlling CS 10110. Alternately, the SITT 11012 in MEM 10112 could provide conventional instructions usable by a conventional CPU, for example, Fortran or machine language instructions. This, for example, would allow FU 10120 to be replaced by a conventional CPU, such as a Data General Corporation Eclipse®.

Having briefly summarized certain features of CS 10110, and alternate embodiments of certain of these features, the structure and operation of CS 10110 will be described in detail below.

2. DETAILED DESCRIPTION OF CS 10110 MAJOR SUBSYSTEMS

(Figs. 201—206, 207—274)

Having previously described the overall structure and operation of CS 10110, the structure and operation of CS 10110's major subsystems will next be individually described in further detail. As previously discussed, CS 10110's major subsystems are, in the order in which they will be described, MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118. Individual block diagrams of MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118 are shown in, respectively, Figures 201 through 205. Figures 201 through 205 may be assembled as shown in Fig. 206 to construct a more detailed block diagram of CS 10110 corresponding to that shown in Fig. 101. For the purposes of the following descriptions, it is assumed that Figs. 201 through 205 have been assembled as shown in Fig. 206 to construct such a block diagram. Further diagrams will be presented in following descriptions as required to convey structure and operation of CS 10110 to one of ordinary skill in the art.

As previously described, MEM 10112 is an intelligent, priortizing memory having separate and independent ports MIO 10128 and MJP 10140 to, respectively, IOS 10116 and JP 10114. MEM 10112 is shared by and is accessible to both JP 10114 and IOS 10116 and is the primary memory of CS 10110. In addition, MEM 10112 is the primary path for information transferred between the external world (through IOS 10116) and JP 10114.

As will be described further below, MEM 10112 is a two-level memory providing fast access to data stored therein. MEM 10112 first level is comprised of a large set of random access arrays and MEM 10112 second level is comprised of a high speed cache whose operation is generally transparent to memory users, that is JP 10114 and IOS 10116. Information stored in MEM 10112, in either level, appears to be bit addressable to both JP 10114 and IOS 10116. In addition, MEM 10112 presents simple interfaces to both JP 10114 and IOS 10116. Due to a high degree of pipe lining (concurrent and overlapping memory operations) MEM 10112 interfaces to both JP 10114 and IOS 10116 appear as if each HP 10114 and IOS 10116 have full access to MEM 10112. This feature allows data transfer rates of up to, for example, 63.6 megabytes per second from MEM 10112 and 50 megabytes per second to MEM 10112.

In the following descriptions, certain terminology used on those descriptions will be introduced first, followed by description of MEM 10112 physical organization. Then MEM 10112 port structures will be described, followed by descriptions of MEM 10112's control organization and control flow. Next, MEM 10112's interfaces to JP 10114 and IOS 10116 will be described. Following these overall descriptions the major logical structures of MEM 10112 will be individually described, starting at MEM 10112's interfaces to JP 10114 and IOS 10116 and proceeding inwardly to MEM 10112's first (or bulk) level of data stored. Finally, certain features of MEM 10112 microcode control structure will be described.

A. MEM 10112 (Figs. 201, 206, 207—237)

a. Terminology

Certain terms are used throughout the following descriptions and are defined here below for reference

Petitioner Apple Inc. - Exhibit 1004, p. 5210

by the reader.

A word is 32 bits of data

A byte is 8 bits of data

A block is 128 bits of data (that is, 4 words).

A block is always aligned on a block boundary, that is the low order 7 bits of logical or physical address are zero (see Chapter 1, Sections A.f and D. Descriptions of CS 10110 Addressing).

The term aligned refers to the starting bit address of a data item relative to certain address boundaries. A starting bit address is block aligned when the low order 7 bits of starting bit address are equal to zero, that is the starting bit address falls on a boundary between adjacent blocks. A word align starting bit address means that the low order 5 bits of starting bit address are zero, the starting bit address points to a boundary between adjacent words. A byte aligned starting bit address means that the low order 3 bits of starting bit address are zero, the starting bit address points to a boundary between adjacent bytes.

Bit granular data has a starting bit address falling within a byte, but not on a byte boundary, or the address is aligned on a byte boundary but the length of the data is bit granular, that is not a multiple of 8 bits.

b. MEM 10112 Physical Structure (Fig. 201)

Referring to Fig. 201, a partial block diagram of MEM 10112 is shown. Major functional units of MEM 10112 are Main Store Bank (MSB) 20110, including Memory Arrays (MA's) 20112, Bank Controller (BC) 20114, Memory Cache (MC) 20116, including Bypass Write File (BYF) 20118, Field Isolation Unit (FIU) 20120, and Memory Interface Controller (MIC) 20122.

MSB 20110 comprises MEM 10112's first or bulk level of storage. MSB 20110 may include from one to, for example, 16 MA 20112's. Each MA 20112 may have a storage capacity, for example, 256 K-byte, 512 K-byte, 1 M-byte, or 2 M-bytes of storage capacity. As will be described further below, MA 20112's of different capacities may be used together in MSB 20110. Each MA 20112 has a data input connected in parallel to Write Data (WD) Bus 20124 and a data output connected in parallel to Read Data (RD) Bus 20126. MA's 20112 also have control and address ports connected in parallel to address and control (ADCTL) Bus 20128. In particular, Data Inputs 20124 of Memory Arrays 20112 are connected in parallel to Write Data (WD) Bus 20126, and Data Outputs 20128 of Memory Arrays 20112 are connected in parallel to Read Data (RD) Bus 20130. Control Address Ports 20132 of Memory Arrays 20112 are connected in parallel to Address and Control (ADCTL) Bus 20134.

Data Output 20136 of Bank Controller 20114 is connected to WD Bus 20126 and Data Input 20138 of BC 20114 is connected to RD Bus 20130. Control and Address Port 20140 of BC 20114 is connected to ADCTL Bus 20134. BC 20114's Data Input 20142 is connected to MC 20116's Data Output 20144 through Store Back Data (SBD) Bus 20146. BC 20114's Store Back Address Input 20148 is connected to MC 20116 Store Back Address Output 20150 through Store Back Address (SBA) Bus 20152. BC 20114's Read Data Output 20154 is connected to MC 20116's Read Data Input 20156 through Read Data Out (RDO) Bus 20158. BC 20114's Control Port 20160 is connected to Memory Control (MCNTL) Bus 20164.

MC 20116 has Output 20166 connected to MIO Bus 10131 through MIO Port 10128, and Port 20168 connected to MOD Bus 10144 through MJP Port 10140. Control Port 20170 of MC 20116 is connected to MCNTL Bus 20164. Input 20172 of BYF 20118 is connected to IOM Bus 10130 through MIO Port 10128, and Output 20176 is connected to SBD Bus 20146 through Bypass Write In (BWI) Bus 20178.

Finally, FIU 20120 has an Output 20180 and an Input 20182 connected to, respectively, MIO Bus 10129 and IOM Bus 10130 through MIO Port 10128. Input 20184 and Port 20186 are connected to, respectively, JPD Bus 10142 and MOD Bus 10144 through MJP Port 10140. Control Port 20188 is connected to MCNTL Bus 20164. Referring finally to MIC 20122, MIC 20122 has Control Port 20190 and Input 20192 connected to, respectively, IOMC Bus 10131 and IOM Bus 10130 through MIO Port 10128. Control Port 20194 and Input 20196 are connected, respectively, to JPMC Bus 10147 and Physical Descriptor (PD) Bus 10146 through MJP Port 10140. Control Port 20198 is connected to MCNTL Bus 20164.

c. MEM 10112 General Operation

Referring first to MEM 10112's interface to IOS 10116, this interface includes MIO Bus 10129, IOM Bus 10130, and IOMC Bus 10131. Read and Write Addresses and data to be written into MEM 10112 are transferred from IOS 10116 to MEM 10112 through IOM Bus 10130. Data read from MEM 10112 is transferred to IOS 10116 through MIO Bus 10129. IOMC 10131 is a Bi-directional Control bus between MEM 10112 and IOS 10116 and, as described further below, transfers control signals between MEM 10112 and IOS 10116 to control transfer of data between MEM 10112 and IOS 10116.

MEM 10112's interface to JP 10114 is MJP Port 10140 and includes JPD Bus 10142, MOD Bus 10144, PD Bus 10146, and JPMC Bus 10147. Physical descriptors, that is MEM 10112 physical read and write addresses, are transferred from JP 10114 to MEM 10112 through PD Bus 10146. S Ops, that is sequences of S Instructions and operand names, are transferred from MEM 10112 to JP 10114 through MOD Bus 10144 while data to be written into MEM 10112 from JP 10114 is transferred from JP 10114 to MEM 10112 through JPD Bus 10142. JPMC Bus 10147 is a By-directional Control bus for transferring command and control signals between MEM 10112 and JP 10114 for controlling transfer of data between MEM 10112 and JP 10114. As will be described further below, MJP Port 10140, and in particular MOD Bus 10144 and PD Bus

Petitioner Apple Inc. - Exhibit 1004, p. 5211

10146, is generally physically organized as a single port that operates as a dual port. In a first case, MJP Port 10140 operates as a Job Processor Instruction (JI) Port for transferring S Ops from MEM 10112 to JP 10114. In a second case, MOD 10144 and PD 10146 operate as a Job Processor Operand (JO) Port for transfer of operands, from MEM 10112 to JP 10114, while JPD Bus 10142 and PD Bus transfer operands from JP 10114 to MEM 10112.

Referring to MSB 20110, MSB 20110 contains MEM 10112's first, or bulk, level of storage capacity. MSB 20110 may contain from one to, for example, 16 MA's 20112. Each MA 20112 contains a dynamic, random access memory array and may have a storage capacity of, for example 256 Kilo-bytes, 512 Kilo-bytes, 1 Mega-bytes, or 2 Mega-bytes. MEM 10112 may therefore have a physical capacity of up to, for example, 16 Mega-bytes of bulk storage. As will be described further below. MA 20112's of different capacity may be used together in MSB 20110, for example, four 2 Mega-byte MA 20112's and four 1 Megabyte MA 20112's.

BC 20114 controls operation of MA's 20112 and is the path for transfer of data to and from MA's 20112. In addition, BC 20114 performs error detection and correction on data transferred into and out of MA's 20112, refreshes data stored in MA's 20112, and, during a refresh operations, performs error detection and correction of data stored in MA's 20112.

MC 20116 comprises MEM 10112's second, or cache, level of storage capacity and contains, for example 8 Kilo-bytes of high speed memory. MC 20116, including BYF 20118, is also the path for data transfer between MSB 20110 (through BC 20114) and JP 10114 and IOS 10116. In general, all read and write operations between JP 10114 and IOS 10116 are through MC 20116. IOS 10116 may, however, perform read and write operations of complete blocks by-passing MC 20116. Block write operations from IOS 10116 are accomplished through BYF 20118 while block read operations are performed through a data transfer path internal to MC 20116 and shown and described below. All read and write operations between MEM 10112 and JP 10114, however, must be performed through the cache internal to MC 20116, as will be shown and described further below.

As also shown and described below, FIU 20120 includes write data registers for receiving data to be written into MEM 10112 from JP 10114 and IOS 10116, and circuitry for manipulating data read from MSB 20110 so that MEM 10112 appears as a bit addressable memory. FIU 20120, in addition to providing bit addressability of MEM 10112, performs right and left alignment of data, zero fill of data, sign extension operations, and other data manipulation operations described further below. In performing these data manipulation operations on data read from MEM 10112 to JP 10114, MOD Bus 10144 is used as a data path internal to MEM 10112 for transferring of data from MC 20116 to FIU 20120, and from FIU 20120 to MC 20116. That is, data to be transferred to JP 10114 is read from MC 20116, transferred through MOD Bus IC144 to FIU 20120, manipulated by FIU 20120, and transferred from FIU 20120 to JP 10114 through MOD Bus 10144.

MIC 20122 contains circuitry controlling operation of MEM 10112 and, in particular, controls MEM 10112's interface with JP 10114 and IOS 10116. MIC 20122 receives MEM 10112 read and write request, that is read and write addresses through PD Bus 10146 and IOM Bus 10130 and control signals through JPMC Bus 10147 and IOMC Bus 10131, and provides control signals to BC 20114, MC 20116, and FIU 20120 through MCNTL Bus 20164.

Having described the overall structure and operation of MEM 10112, the structure and operation of MEM 10112's Port, MIO Port 10128, and MJP Port 10140, will be described next, followed by descriptions of MEM 10112's control structure and the control and flow of MEM 10112 read and write requests.

d. MEM 10112 Port Structure

MEM 10112 port structure is designed to provide a simple interface to JP 10114 and IOS 10116. While providing fast and flexible operation in servicing MEM 10112 read and write requests from JP 10114 and IOS 10116. In this regard, MEM 10112, as will be described further below, may handle up to 4 read and write requests concurrently and up to, for example, a 63.6 M-byte per second data rate. In addition MEM 10112 is capable of performing bit granular addressing, block read and write operations, and data manipulations, such as alignment and filling, to enable JP 10114 and IOS 10116 to operate most efficiently.

MEM 10112 effectively services requests from three ports. These ports are MIO Port 10128 to IOS 10116, hereafter referred to as IO Port, and JI and JO Ports, described above, to JP 10114. These three ports share the entire address base of MEM 10112, but IOS 10116, for example, may be limited from making full use of MEM 10112's address space. Each port has a different set of allowed operations. For example, JO Port can use a bit granular addresses but can reference only 32 bits of data on each request. JI Port can make read requests only to word align 32 bit data items. IO Port may reference bit granular data, and, as described further below, may read or write up to 16 bytes on each read or write request. The characteristics of each of these ports will be discussed next below.

1. IO Port Characteristics

IOS 10116 may access MEM 10112 in either of two modes. The first mode is block transfers by-passing or through the cache in MC 20116, and the second is non-block transfer through the cache and MC 20116.

Block by-passes may occur for both read and write operations. A read or write operation is eligible for a block by-pass if the data is on block boundaries, is 16 bytes long, and the read or write request is not accompanied by a control signal indicating that an encache (load into MC 20116's cache) operation is to be

Petitioner Apple Inc. - Exhibit 1004, p. 5212

performed. A by-pass operation takes place only if the block address, that is the physical address of the block in MEM 10112 does not address a currently encached block, that is the block is not present in MC 20116's cache. If the block is encached in MC 20116's cache, the read or write transfer is to MC 20116's cache.

Partial block references, that is non-full block transfers will go through MC 20116's cache. If a cache miss occurs, that is the reference data is not present in MC 20116's cache, MEM 10112's control structures transfer the data to or from MSB 20110 and update MC 20116's cache. It should be noted that partial blocks may be as short as one byte, or up to 15 bytes long. A starting byte address may be anywhere within a block, but the partial block's length may not cross a block boundary.

Bit length transfers, that is transfers of data items having a length of 1 to 16 bits and not a multiple of a byte, or where address is not on a byte boundary, go through MC 20116's cache. These operations may cross byte, word, or block boundaries but may not cross page boundaries. These specific operations requested by IO port determines whether a read or write request is a partial block or bit length transfer.

2. JO Port Characteristics

All read or write requests from JO Port must go through MC 20116's cache; by-pass operations may not be performed. The data transferred between MEM 10112 and JP 10114 is always 32 bits in length but, of the 32 bits passed, from zero to 32 bits may be valid data. JP 10114 determines the location of valid data within the 32 bits by referring to certain FIU specification bits provided as part of the read or write request. As will be described further below, FIU specification bits, and other control bits, are provided to MIC 20122 by JP 10114 through JPMC Bus 10147 when each read or write request is made.

While MEM 10112 does not perform block by-pass operations to JP 10114, MEM 10112 may perform a cache read-through operation. Such operations occur on a JP 10114 read request wherein the requested data is not present in MC 20116's cache. If the JP 10114 read request is for a full word, which is word aligned, MEM 10112's Load Manager, discussed below, transfers the requested data directly to JP 10114 while concurrently loading the requested data into MC 20116's cache. This operation is referred to as a "hand-off" operation. These operations may also be performed by IO Port for 16 bit half words aligned on the right hand half word of a 32 bit word, or if a full block is handed left and loaded into MC 20116's cache.

3. JI Port Characteristics

All JI Port requests are satisfied through MC 20116's cache; MEM 10112 does not perform by-pass operations to JI Port. JI Port requests are always read requests for full-word aligned words and are handed off, as described above, if a cache miss occurs. In most other respects, JI Port requests are similar to JO Port requests.

Having described the overall structure and operation of MEM 10112, including MEM 10112's input and output ports to JP 10114 and IOS 10116, MEM 10112's control structure will be described next below.

e. MEM 10112 Control Structure and Operation (Fig. 207)

Referring to Fig. 207, a more detailed block diagram of MIC 20116 is shown. Fig. 207 will be referred to in conjunction with Fig. 201 in the following discussion of MEM 10112's control structure.

1. MEM 10112 Control Structure

Referring first to Fig. 207, MCNTL Bus 20164 is represented as including MCNTL-BC Bus 20164A, MCNTL-MC Bus 20164B, and MCNTL-FIU Bus 20164C. Buses 20164A, 20164B, and 20164C are branches of MCNTL Bus 20164 connected to, respectively, BC 20114, MC 20116, and FIU 20120. Also represented in Fig. 207 are PD Bus 10146 and JPMC Bus 10147 to JP 10114, and IOM Bus 10130 and IOMC Bus 10131 to IOS 10116.

JO Port Address Register (JOPAR) 20710 and JI Port Address Register (JIPAR) 20712 have inputs connected from PD Bus 10146. IO Port Address Register (IOPAR) 20714 has an input connected from IOM Bus 10130. Port Control Logic (PC) 20716 has a bi-directional input/outputs connected from JPC 10147 and IOMC Bus 10131. By-pass Read/Write Control Logic (BR/WC) 20718 has a bidirectional input/output connected from IOMC Bus 10131.

Outputs of JOPAR 20710, JIPAR 20712, and IOPAR 20714 are connected to inputs of Port Request Multiplexer (PRMUX) 20720 through, respectively, Buses 20732, 20734, 20736. PRMUX 20720's output in turn is connected to Bus 20738. Branches of Bus 20738 are connected to inputs of Load Pointers (LP) 20724, Miss Control (MISSC) 20726, and Request Manager (RM) 20722, and to Buses MCNTL-MC 20164B and MCNTL-FIU 20164C.

Outputs of PC 20716 are connected to inputs of JOPAR 20710, JIPAR 20712, IOPAR 20714, PRMUX 20720, and LP 20724 through Bus 20738. Bus 20740 is connected between an input/output of PC 20716 and in input/output of RM 20722.

An output of BR/WC 20718 is connected to MCNTL-MC Bus 20164B through Bus 20742. Inputs of BR/WC 20718 are connected from outputs of RM 20722 and Read Queue (RQ) 20728 through, respectively, Buses 20744 and 20746.

RM 20722 has outputs connected to MCNTL-BC Bus 20164A, MCNTL-FIU Bus 20164C, and input of MISSC 20726, and an input of LP 20724 through, respectively, Buses 20748, 20750, 20752, and 20754.

Petitioner Apple Inc. - Exhibit 1004, p. 5213

MISSC 20726's output is connected to MCNTL-BC Bus 20164A. Outputs of LP 20724 are connected to MCNTL-MC Bus 20164B and to an input of LM 20730 through, respectively, Buses 20756 and 20758. RQ 20728's input is connected from MCNTL-MC Bus 20164B through Bus 20760 and RQ 20728 has outputs connected to an input of LP 20724, through Bus 20762, and as previously described to an input of BR/WC 20718 through Bus 20746. Finally, LM 20730's output is connected to MCNTL-MC Bus 20164B through Bus 20764.

Having described the structure of MIC 20716 with reference to Fig. 207, and having previously described the structure of MEM 10112 with reference to Fig. 201, MEM 10112's control structure operation will next be described with reference to both figures 201 and 207.

2. MEM 10112 Control Operation

Referring first to Fig. 207, JOPAR 20710, JIPAR 20712, and IOPAR 20714 are, as previously described, connected from PD Bus 10146 from JP 10114 and IOM Bus 10130 from IOS 10116. JPAR 20710, JIPAR 20712, and IOPAR 20714 receive read and write request addresses from JP 10114 and IOS 10116 and store these addresses for subsequent service by MEM 10112. As will be described further below, these address inputs from JP 10114 and IOS 10116 include FIU information specifying what data manipulation operations must be performed by FIU 20120 before requested data is transferred to the requestor or written into MEM 10112, information regarding the destination data read from MEM 10112 is to be provided to, information regarding the type of operation to be performed by MEM 10112, and information regarding operand length. Request address information received and stored in JOPAR 20710, JIPAR 20712, and IOPAR 20714 is retained therein until MEM 10112 has initiated service of the corresponding requests. MEM 10112 will accept further request address information into a given port register only after a previous request into that port has been serviced or aborted. Address information outputs from JOPAR 20710, JIPAR 20712, and IOPAR 20714 is transferred through PRMUX 20720 to Bus 20738 and from there to RM 20722, MC 20116, and FIU 20120 as service of individual requests is initiated. As will be described below, this address information will be transferred through PRMUX 20720 and Bus 20738 to LP 20724 for use in servicing a cache miss upon occurrence of a MC 20116 miss.

PC 20716 receives command and control signals pertinent to each requested memory operation from JP 10114 and IOS 10116 through JPMC Bus 10147 and IOSC Bus 10131. PC 20716 includes request arbitration logic and port state logic. Request arbitration logic determines the sequence in which IO, JI, JO ports are serviced, and when each port is to be serviced. In determining the sequence of port service, request arbitration logic uses present port state information for each port from the port state logic, information from JPMC Bus 10147 and IOMC Bus 10131 regarding each incoming request, and information from RM 20722 concerning the present state of operation of MEM 10112. Port state logic selects each particular port to be serviced and, by control signals through Bus 20738, enables transfer of each port's request address information from JOPAR 20710, JIPAR 20712, and IOPAR 20714 through PRMUX 20720 to Bus 20738 for use by the remainder of MEM 10112's control logic in servicing the selected port. In addition to request information received from JP 10114 and IOS 10116 through JPMC Bus 10147 and IOMC Bus 10131, port state logic utilizes information from RM 20722 and, upon occurrence of a cache miss, from LM 20730 (for clarity of presentation, this connection is not represented in Fig. 207). Port state logic also controls various port state flag signals, for example port availability signals, signals indicating valid requests, and signals indicating that various ports are waiting service.

RM 20722 controls execution of service for each request. RM 20722 is a microcode controlled "micromachine" executing programs called for by requested MEM 10112 operations. Inputs of RM 20722 include request address information from IOPAR 20714, JIPAR 20212, and JOPAR 20210, including information regarding the type of MEM 10112 operation to be performed in servicing a particular request, interrupt signals from other MEM 10112 control elements, and, for example, start signals from PC 20716's request arbitration logic. RM 20722 provides control signals to FIU 20120, MC 20116, and most other parts of MEM 10112's control structure.

Referring to Fig. 201, MC 20116's cache is, for example, an 8 Kilo-byte, four set associative cache used to provide rapid access to a subset of data stored in MSB 20110. The subset of MSB 20110 data stored in MC 20116's cache at any time is the data most recently used by JP 10114 or IOS 10116. MC 20116's cache, described further below, includes tag store comparison logic for determining encached addresses, a data store containing corresponding encached data, and registers and logic necessary to up-date cache contents upon occurrence of a cache miss. Registers and logic for servicing cache misses includes logic for determining the least recently used cache entry and registers for capture and storage of information regarding missed cache references, for example modify bits and replacement page numbers. Inputs to MC 20116 are provided from RM 20722, LM 20730 (discussed further below), FIU 20120, MSB 20110 (through BC 20114), LP 20724 (described further below) and address information from PRMUX 20720. Outputs of MC 20116 include data and go to FIU 20120 (through MOD Bus 10144), the data requestors (JP 10114 and IOS 10116), and a MC 20116 Write Back File (described further below).

As previously described, FIU 20120 includes logic necessary to make MEM 10112 appear bit addressable. In addition, FIU 20120 includes logic for performing certain data manipulation operations as required by the requestors (JP 10114 or IOS 10116). Data is transferred into FIU 20120 from MC 20116 through that portion of MOD Bus 10144 internal to MEM 10112, is manipulated as required, and is then

Petitioner Apple Inc. - Exhibit 1004, p. 5214

transferred to the requestor through MOD Bus 10144 or MIO Bus 10129. In the case of writes requiring read-modify-write of encached data, the data is transferred back to MC 20116 through MOD Bus 10144 after manipulation. In general, data manipulation operations include locating requested data onto selected MOD Bus 10144 or MIO Bus 10139 lines and filling unused bus lines as specified by the requestor. Data inputs to FIU 20120 may be provided from MC 20116 or JP 10114 through MOD Bus 10144 or from IOS 10116 through IOM Bus 10130. Data outputs from FIU 20120 may be provided to MC 20116, JP 10114, or IOS 10116 through these same buses. Control information is provided to FIU 20120 from RM 20722 through Bus 20748 and MCNTL-FIU Bus 20164C. Address information may be provided to FIU 20120 from JOPAR 20710, JIPAR 20712, or IOPAR 20714 through PRMUX 20720, Bus 20738, and MCNTL-FIU Bus 20164C.

Returning to Fig. 207, MISSC 20726 is used in handling MC 20116 misses. In the event of a request referring to data not in MC 20116's cache, MISSC 20726 stores block address of the reference and type of operation to be performed, this information being provided from an address register in MC 20116 and from RM 20722. MISSC 20726 utilizes this information in generating a command to BC 20114, through MCNTL-BC Bus 20164A, for a data read from MSB 20110 to obtain the referenced data. BC 20114 places this command in a queue, or register, and subsequently executes the commanded read operation. MISSC 20726 also generates an entry into RQ 20728 (described further below) indicating the type of operation to be performed when referenced data is subsequently read from MSB 20110.

RQ 20728 is, for example, a three-level deep queue storing information indicating operations associated with data being read from MSB 20110. Two kinds of operation may be indicated: block by-pass reads and cache loads. If a cache load is specified, that is a read and store to MC 20116's cache, is indicated, RM 20722 is interrupted and forced to place other MEM 10112 operations in idle until cache load is completed. A block by-pass read operation results in by-pass read control (described below) assuming control of the data from MSB 20110. Inputs to RQ 20728 are control signals from RM 20752, MISSC 20726, and BC 20114. RQ 20728 provides control outputs to LP 20724 (described below) LM 20730 (described below) RM 20722, and by-pass read control (described below).

LP 20724 is a set of registers for storing information necessary for servicing MC 20116 misses that result in order to load MC 20116's tag store. LM 20730 uses this information when data stored in MSB 20110 and read from MSB 20110 to service a MC 20116 cache miss, becomes available through BC 20114. Inputs to LP 20724 include the address of the missing reference, provided from JOPAR 20710, JIPAR 20712, or IOPAR 20714 through PRMUX 20720 and Bus 20738, commands from RM 20722, and a control signal from RQ 20728. LP 20724 outputs include addresses of missed references to MC 20116, through Bus 20756 and MNCTL-MC 20164B, and command signals to LM 20730 and BR/WC 20718.

LM 20730, referred to above, controls loading of MC 20116's cache with data from MSB 20110 after occurrence of a cache miss. RQ 20728, referred to above, indicates, for each data read from MSB 20110, whether the data read is the result of a MC 20116 cache miss. If the data is read from MSB 20110 as a result of a cache miss, LM 20730 proceeds to issue a sequence of control signals for loading the data from MSB 20110 and its associated address into MC 20116's cache. This data is transferred into MC 20116's cache data store while the block address, from LP 20724 is transferred into the tag store (described in the following discussion) of MC 20116's cache. If the transfer of data into MC 20116's cache replaces data previously resident in that cache, and that previous data is "dirty", that is has been written into so as to be different from an original copy of the data stored on MSB 20110, the modified data resident in MC 20116's cache must be written back into MSB 20110. This operation is performed through a Write Back File contained in MC 20116 and described below. In the event of such an operation, LM 20730 initiates a write back operation by MC 20116 and BC 20114, also as described below.

As will be described further in a following description, all MC 20116 cache load operations are full 4 word blocks. A request resulting in a MC 20116 cache miss may result in a "hand-off", that is a read operation of a full 4 word block. Handoff operations also may be of single 32 bit words wherein a 32 bit word aligned word is transferred from JP 10114 or a 16 bit operand aligned on the right half-word is transferred from IOS 10116. In such a handoff operation, LM 20730 will send a valid request signal to the requesting port and a handoff operation will be performed. Otherwise, a waiting signal will be sent to the requesting port and the request will re-enter the priority queue of PC 20716 for subsequent execution. To accomplish these operations, LM 20730 receives input from RQ 20728, (not shown in Fig. 207 for clarity of presentation) and LP 20724. LM 20730 provides outputs to port state logic of PC 20716, to MC 20116, MC 20116's Write Back File and MC 20116's Write Back Address Register and to BC 20114.

Referring to Fig. 201, as previously discussed IOS 20116 may request a full block write operation directly to MSB 20110. Such a by-pass write request may be honored if the block being transferred is not encached in MC 20116's cache. In such a case, RM 20722 will initiate the transfer setting up By-pass Write Control logic in BR/WC 20718, and may then pass control of the operation over to BR/WC 20718's By-Pass Write Control logic for completion. By-pass Write Control may then accept the remaining portion of the data block from IOS 10116, generating appropriate hand shaking signals through IOMC Bus 10131, and load the data block into BYF 20118 and MC 20116. MISSC 20726 will provide a by-pass write command to BC 20114, through MNCTL-PC Bus 20164A. BC 20114 will then transfer the data block from BYF 20118 and into MA's 20112 and MSB 20110.

As previously described, BYF 20118 receives data from IOM Bus 10130 and provides data output to BC 20114 through BWY Bus 20178 and SBD Bus 20146. BYF 20118 is capable of simultaneously accepting data

Petitioner Apple Inc. - Exhibit 1004, p. 5215

from IOM Bus 10130 while reading data out to BC 20114. Control of writing data into BYF 20118 is provided from BR/WC 20718's By-Pass Write Control logic.

IOS 10116 may, as previously described, request a full block read operation by-passing MC 20116's cache. In such a case, BR/WC 20718's by-pass read control handles data transfer to IOS 10116 and generates required hand shaking signals to IOS 10116 through IOMC Bus 10131. The data path for by-pass read operations is through a data path internal to MC 20116, rather than through BYF 20118. This internal data path is RDO Bus 20158 to MIO Bus 10129.

As previously described, BC 20114 manages all data transfers to and from MA's 20112 in MSB 20110. BC 20114 receives requests for data transfers from RM 20722 in an internal queue register. All data transfers to and from MSB 20110 are full block transfers with block aligned addresses. On data write operations, BC 20114 receives data from BWF 20118 or from MC 20116's Write Back File and transfers the data into MA's 20112. During read operations, BC 20114 fetches the data block from MA's 20112 and places the data block on RDO Bus 20158 while signalling to MIC 20122 that the data is available. As described above, MIC 20122 tracks and controls transfer of data and BYF 20118, MC 20116, and MC 20116's Write Back File, and directs data read from MSB 20110 to the appropriate destination, MC 20116's Data Store, JP 10114, or IOS 10116.

In addition to the above operations, BC 20114 controls refresh of MA's 20112 and performs error detection and correction operations. In this regard, BC 20114 performs two error detection and correction operations. In the first, BC 20114 detects single and double bit errors in data read from MSB 20110 and corrects single bit errors. In the second, BC 20114 reads data stored in MA's 20112 during refresh operations and performs single bit error detection. Whenever an error is detected, during either read operations or refresh operations, BC 20114 makes a record of that error in an error log contained in BC 20114 (described further in a following description). Both JP 10114 and IOS 10116 may read BC 20114's error log, and information from BC 20114's error log may be recorded in a CS 10110 maintenance log and to assist in repair and trouble shooting of CS 10110. BC 20114's error log may be addressed directly by RM 20722 and data from BC 20114's error log is transferred to JP 10114 or IOS 10116 in the same manner as data stored in MSB 20110.

Referring finally to MA's 20112, each MA 20112 contains an array of dynamic semiconductor random access memories. Each MA 20112 may contain 256 Kilo-bytes, 512 Kilo-bytes, 1 Mega-bytes, or 2 Mega-bytes of data storage. The storage capacity of each MA 20112 is organized as segments of 256 Kilo-bytes each. In addressing a particular MA 20112, BC 20114 selects that particular MA 20112 as will be described further below. BC 20114 concurrently selects a segment within that MA 20112, and a block of four words within that segment. Each word may comprise 39 bits of information, 32 bits of data and 7 bits of error correcting code. The full 39 bits of each MA 20112 word are transferred between BC 20114 and MA's 20112 during each read and write operation. Having briefly described the general structure and operation of MEM 10112, certain types of operations which may be performed by MEM 10112 will be described next below.

f. MEM 10112 Operations

MEM 10112 may perform two general types of operation. The first type are data transfer operations and the second type are memory maintenance operations. Data transfer operations may include read, write, and read and set. Memory maintenance operations may include read error log, repair block, and flush cache. Except during a flush cache operation, the existence of MC 20116 and its operation is invisible to the requestors, that is JP 10114 and IOS 10116.

A MEM 10112 read operation transfers data from MS 10112 to a requestor, either JP 10114 or IOS 10116. A read data transfer is asynchronous in that the requestor cannot predict elapsed time between submission of a memory operation request and return of requested data. Operation of a requestor in MEM 10112 is coordinated by a requested data available signal transmitted from MEM 10112 to the requestor.

A MEM 10112 write operation transfers data from either JP 10114 or IOS 10116 to MEM 10112. During such operations, JP 10114 is not required to wait for a signal from MEM 10112 that data provided to MEM 10112 from JP 10114 has been accepted. JP 10114 may transfer data to MEM 10112's JO Port whenever a JO Port available signal from MEM 10112 is present; read data is accepted immediately without further action or waiting required of JP 10114. Word write operations from IOS 10116 are performed in a similar manner. On block write operations, however, IOS 10116 is required to wait for a data taken signal from MEM 10112 before sending the 2nd, 3rd and 4th words of a block.

MEM 10112 has a capability to perform "lock bit" operations. In such operations, a bit granular read of the data is performed and the entire operand is transmitted to the requestor. At the same time, the most significant bit of the operand, that is the Lock Bit, is set to one in the copy of data stored in MEM 10112. In the operand sent to the requestor, the lock bit remains at its previous value, the value before the current read and set operation. Test and set operations are performed by performing read and set operations wherein the data item length is specified to be one bit.

As previously described, MEM 10112 performs certain maintenance operations, including error detection. MEM 10112's Error Log in BC 20114 is a 32 bit register containing an address field and an error code field. On a first error to occur, the error type and in some cases, such as ERCC errors on read data stored in MSB 20110, the address of the data containing the error are stored in BC 20114's Error Log Register. An interrupt signal indicating detection of an error is raised at the same that information

Petitioner Apple Inc. - Exhibit 1004, p. 5216

regarding the error is stored in the Error Log. If multiple errors occur before Error Log is read and reset, the information regarding the first error will be retained and will remain valid. The Error Log code field will, however, indicate that more than one error has occurred.

JP 10114 may request a read Error Log operation referred to as a "Read Log and Reset" operation. In this operation, MEM 10112 reads the entire contents of Error Log to JP 10114, resets Error Log Register, and resets the interrupt signal indicating presence of an error. IOS 10116, as discussed further below, is limited to reading 16 bits at a time from MEM 10112. It therefore requires two read operations to read Error Log. First read operation to IOS 10116 reads an upper 16 bits of Error Log data and does not reset Error Log. The second read operation is performed in the same manner as a JP 10114 Read Log and Reset operation, except that only the low order 16 bits of Error Log are read to IOS 10116.

MEM 10112 performs repair block operations to correct parity or ERCC errors in data stored in MC 20116's Cache or in data stored in MA's 20112. In a repair block procedure, parity bits for data stored in MC 20116's Cache, or ERCC check bits of data stored in MA's 20112, are modified to agree with the data bits of data stored therein. In this regard, repaired uncorrectible errors, such as two bit errors of data in MA's 20112, will have good ERCC and parity values. Until a repair block operation is performed, any read request directed to bad data, that is data having parity or ERCC check bits indicating invalid data, will be flagged as invalid. Repair block operations therefore allow such data to be read as valid, for example to be used in a data correction operation. Errors are ignored and not logged in BC 20114's Error Log in repair block operations. A write operation into an area containing bad data may be accomplished if MEM 10112's internal operation does not require a read-modified-write procedure. Only byte aligned writes of integral byte length data residing in MC 20116 and word aligned writes of integral word lengths of data in MSP 20110 do not require read-modified-write operation. By utilizing such write operations, it is therefore possible to overwrite bad data by use of normal write operations before or instead of repair block operations.

MEM 10112 performs a cache flush operation in event of a power failure, that is when MEM 10112 goes into battery back-up operation. In such an event, only MA's 20112 and BC 20114 remain powered. Before JP 10114 and IOS 10116 lose power, JP 10114 and IOS 10116 must transfer to MEM 10112 any data, including operating state, to be saved. This is accomplished by using a series of normal write operations. After conclusion of these write operations, both JP 10114 and IOS 10116 transmit a flush cache request to MEM 10112. Upon receiving two flush cache requests, MEM 10112 flushes MC 20116's Cache so that all dirty data encached in MC 20116's Cache is transferred into MA's 20112 before power is lost. If only JP 10114 or IOS 10116 is operating, DP 10118 will detect this fact and will have transmitted an enabling signal (FLUSHOK) to MEM 10112 during system initialization. FLUSHOK enables MEM 10112 to perform cache flush upon receiving a single flush cache request. After a cache flush operation, no further MEM 10112 operations are possible until Dp 10118 resets a power failure lock-out signal to enable MEM 10112 to resume normal operation.

Having described MEM 10112's overall structure and operation and certain operations which may be performed by MEM 10112, MEM 10112's interfaces to JP 10114 and IOS 10116 will be described next below.

g. MEM 10112 Interfaces to JP 10114 and IOS 10116 (Figs. 209, 210, 211, 204)

As previously described, MJP Port 10140 and MIO Port 10128 logically function as three independent ports. These ports are an IO Port to IOS 10116, a JP Operand Port to JP 10114 and a JP Instruction Port to JP 10114. Referring to Figs. 209, 210, and 211, diagramic representations of IO Port 20910, JP Operand (JPO) Port 21010, and JP Instruction (JPI) port 21110 are shown respectively.

IO Port 20910 handles all IOS 10116 requests to MEM 10112, including transfer of both instructions and operands. JPO Port 21010 is used for read and write operations of operands, for example numeric values, to and from JP 10114. JPI Port 21110 is used to read SINs, that is SOPs and operand NAMEs, from MEM 10112 to JP 10114. Memory service requests to a particular port are serviced in the order that the requests are provided to the Port. Serial order is not maintained between requests to different ports, but ports may be serviced in the order of their priority. In one embodiment of the present invention, IO Port 20910 is accorded highest priority, followed by JPO port 21010, and lastly by JPI Port 21110, with requests currently contained in a port having priority over incoming requests. As described above and will be described in more detail in following descriptions, MEM 10112 operations are pipelined. This pipelining allows interleaving of requests from IO Port 20910, JPO Port 21010, and JPI port 21110, as well as overlapping service of requests at a particular port. By overlapping operations it is meant that one operation servicing a particular port begins before a previous operation servicing that port has been completed.

1. IO Port 20910 Operating Characteristics (Figs. 209, 204)

Referring first to Fig. 209, a diagramic representation of IO port 20910 is shown. Signals are transmitted between IO Port 20910 and IOS 10116 through MIO Bus 10129, IOM Bus 10130, and IOMC Bus 10131. MIO Bus 10129 is a unidirectional bus having inputs from MC 20116 and FIU 20120 and dedicated to transfers of data and instructions from MEM 10112 to IOS 10118. IOM Bus 10130 is likewise a unidirectional bus and is dedicated to the transfer, from IOS 10118 to MEM 10112, of read addresses, write addresses, and data to be written into MEM 10112. IOM Bus 10130 provides inputs to BYF 20118, FIU 20120, and MIC 20122. IOMC Bus 10131 is a set of dedicated signal lines for the exchange of control signals between IOS 10118 and MEM

Petitioner Apple Inc. - Exhibit 1004, p. 5217

10112.

Referring first to MIO Bus 10129, MIO Bus 10129 is a 36 bit bus receiving read data inputs from MC 20116's Cache and from FIU 20120. A single read operation from MEM 10112 to IOS 10116 transfers one 32 bit word (or 4 bytes) of data (MIO(0—31)) and four bits of odd parity (MIOP(0—3)), or one parity bit per byte.

Referring next to IOM Bus 10130, a single transfer from IOS 10116 to MEM 10112 includes 36 bits of information which may comprise either a memory request comprising a physical address, a true length, and command bits. These memory requests and data are multiplexed onto IOM 10130 by IOS 10116.

Data transfers from IOS 10116 to MEM 10112 each comprise a single 32 bit data word (IOM(0—31)) and four bits of odd parity (IOMP(0—3)) or one parity bit per byte. Such data transfers are received by either BYF 20118 or FIU 20120.

Each IOS 10116 memory request to MEM 10112, as described above, an address field, a length field, and an operation code field. Address and length fields occupy the 32 IOM Bus 10130 lines used for transfer of data to MEM 10112 in IOS 10116 write operations. Length field includes four bits of information occupying bits (IOM(03)) of IOM Bus 10130 and address field contains 27 bits of information occupying bits (IOM(4—31)) of IOM Bus 10130. Together, address and length field specify a physical starting address and true length of the particular data item to be written into or read from MEM 10112. Operation code field specifies the type of operation to be performed by MEM 10112. Certain basic operation codes comprise 3 bits of information occupying bits (IOMP (32—36)) of IOM Bus 10130; as described above. These same lines are used for transfer of parity bits during data transfers. Certain operations which may be requested of MEM 10112 by IOS 10116 are, together with their corresponding command code fields, are;

000 = read,
001 = read and set,
010 = write,
011 = error,
100 = read error log (first half),
101 = read error log (second half) and reset,
110 = repair block, and
111 = flush cache.

Two further command bits may specify further operations to be performed by MEM 10112. A first command bit, indicates to MEM 10112 during write operations whether it is desirable to encache the data being written into MEM 10112 in MC 20116's Cache. IOS 10116 may set this bit to zero if reuse of the data is unlikely, thereby indicating to MEM 10112 that MEM 10112 should avoid encaching the data. IOS 10116 may set this bit to one if the data is likely to be reused, thereby indicating to MEM 10112 that it is preferable to encache the data. A second command bit is referred to a CYCLE. CYCLE command bit indicates to MEM 10112 whether a particular data transfer is a single cycle operation, that is a bit granular word, or a four cycle operation, that is a block aligned block or a byte aligned partial block.

IOMC 10131 includes a set of dedicated lines for exchange of control signals between IOS 10116 and MEM 10112 to coordinate operation of IOS 10116 and MEM 10112. A first such signal is Load IO Request (LIOR) from IOS 10116 to MEM 10112. When IOS 10116 wishes to load a memory request into MEM 10112, IOS 10116 asserts LIOR to MEM 10112. IOS 10116 must assert LIOR during the same system cycle during which the memory request, that is address, length, and command code fields, are valid.

If LIOR and IO Port Available (IOPA) signals, described below, are asserted during the same clock cycle, MEM 10112's port is loaded from IOS 10116 and IOPA is dropped, indicating the request has been accepted. If a load of a request is attempted and IOPA is not asserted, MEM 10112 remains unaware of the request, LIOR remains active, and the request must then be repeated when IOPA is asserted.

IOPA is a signal from MEM 10112 to IOS 10116 which is asserted by MEM 10112 when MEM 10112 is available to accept a new request from IOS 10116. IOPA may be asserted while a previous request from IOS 10116 is completing operation if the address, length, and operation code fields of the previous request are no longer required by MEM 10112, for example in servicing bypass operations.

IO Data Taken (TIOMD) is a signal from MEM 10112 to IOS 10116 indicating that MEM 10112 has accepted data from IOS 10116. IOS 10116 places a first data word on IOM Bus 10130 on the next system clock cycle after a write request is loaded; that is, LIOR has been asserted, a memory request presented, and IOPA dropped. MEM 10112 then takes that data word on the clock edge beginning the next system clock cycle. At this point, MEM 10112 asserts TIOMD to indicate the data has been accepted. On a single word operations TIOMD is not used by IOS 10116 as a first data word is always accepted by MEM 10112 if IO Port 20910 was available. On block operations, a first data word is always taken but a delay may occur between acceptance of first and second words. IOS 10116 is required to hold the second word valid on IOM Bus 10130 until MEM 10112 responds with TIOMD to indicate that the block operation may proceed.

Data Available for IO (DAVIO) is a signal asserted by MEM 10112 to IOS 10116 indicating that data requested by IOS 10116 is available. DAVIO is asserted by MEM 10112 during the system clock cycle in which MEM 10112 places the requested data on MIO Bus 10129. In any single word type transfer, DAVIO is active for a single system clock transfer. In block type transfers, DAVIO is normally active for four consecutive system clock cycles. Upon event of a single cycle "bubble" resulting from detection and

Petitioner Apple Inc. - Exhibit 1004, p. 5218

correction of an ERCC error by BC 20114, DAVIO will remain high for four non-consecutive system clock cycles and with a single cycle bubble, a non-assertion, in DAVIO corresponding to the detection and correction of the error.

IO Memory Interrupt (IMINT) is a signal asserted by MEM 10112 to IOS 10116 when BC 20114 places a record of a detected error in BC 20114's Error Log, as described above.

Previous MIO Transfer Invalid (PMIOI) signal is similarly a signal asserted by MEM 10112 to IOS 10116 regarding errors in data read from MEM 10112 to IOS 10116. If an uncorrectible error appears in such data, that is an error in two or more data bits, the incorrect data is read to IOS 10116 and PMIOI signal asserted by MEM 10112. Correctible, or single bit, errors in data do not result in assertion of PMIOI. MEM 10112 will assert PMIOI to IOS 10116 of the next system clock cycle following MEM 10112's assertion of DAVIO.

Having described MEM 10112's interface to IOS 10116, and certain operations which IOS 10116 may request of MEM 10112, certain MEM 10112 operations within the capability of the interface will be described next. First, operand transfers, for example of numeric data, between MEM 10112 and IOS 10116 may be bit granular with any length from one to sixteen bits. Operand transfers may cross boundaries within a page but may not cross physical page boundaries. As previously described, MIO Bus 10129 and IOM Bus 10130 are capable of transferring 32 bits of data at a time. The least significant 16 bits of these buses, that is bits 16 to 31, will contain right justified data during operand transfers. The contents of the most significant 16 bits of these buses is generally not defined as MEM 10112 generally does not perform fill operations on read operations to IO Port 20910, nor does IOS 10116 fill unused bits during write operations. During a read or write operation, only those data bits indicated by length field in the corresponding memory request are of significance. In all cases, however, parity must be valid on all 32 bits of MIO Bus 10129 and IOM Bus 10130.

Referring to Fig. 204, IOS 10116 includes Data Channels 20410 and 20412 each of which will be described further in a following detailed description of IOS 10116. Data Channels 20410 and 20412 each possess particular characteristics defining certain IO Port 20910 operations. Data Channel 20410 operates to read and write block aligned full and partial blocks. Full blocks have block aligned addresses and lengths of 16 bytes. Partial blocks have byte aligned addresses and lengths of 1 to 15 bytes; a partial block transfer must be within a block, that is not cross block boundaries. A full 4 word block will be transferred between IOS 10116 and MEM 10112 in either case, but only those blocks indicated by length of field in a corresponding MEM 10112 request are of actual significance in a write operation. Non-addressed bytes in such operations may contain any information so long as parity is valid for the entire data transfer. Data Channel 20412 preferably reads or writes 16 bits at a time on double byte boundaries. Such reads and writes are right justified on MIO Bus 10129 and IOM Bus 10130. The most significant 16 bits of these buses may contain any information during such operations so long as parity is valid for the entire 32 bits. Data Channel 20412 operations are similar to IOS 10116 operand read and write operations with double byte aligned addresses and lengths of 16 bits. Finally, instructions, for example controlling IOS 10116 operation, are read from MEM 10112 to IOS 10116 a block at a time. Such operations are identical to a full block data read.

Having described the operating characteristics of IO Port 20910, the operating characteristics of JPO Port 21010 will be described next.

2. JPO Port 21010 Operating Characteristics (Fig. 210)

Referring to Fig. 210, a diagramic representation of JPO Port 21010 is shown. As previously described, JPO Port 21010 is utilized for transfer of operands, for example numeric data, between MEM 10112 and JP 10114. JPO Port 21010 includes a request input (address, length, and operation information) to MIC 20122 from 36 bit PD Bus 10146, a write data input to FIU 20120 from 32 bit JPD Bus 10142, a 32 bit read data output from MC 20116 and FIU 20120 to 32 bit MOD Bus 10144, and bi-directional control inputs and outputs between MIC 20122 and JPMC Bus 10147.

Referring first to JPO Port 21010's read data output to MOD Bus 10144, MOD Bus 10144 is used by JPO Port 21010 to transfer data, for example operands, to JP 10114. MOD Bus 10144 is also utilized internal to MEM 10112 as a bidirectional bus to transfer data between MC 20116 and FIU 20120. In this manner, data may be transferred from MC 20116 to FIU 20120 where certain data format operations are performed on the data before the data is transferred to JP 10114 through MOD Bus 10144. Data may also be used to transfer data from FIU 20120 to MC 20116 after a data format operation is performed in a write operation. Data may also be transferred directly from MC 20116 to JP 10114 through MOD Bus 10144. Internal to MEM 10112, MOD Bus 10144 is a 36 bit bus for concurrent transfer of 32 bits of data, MOD Bus 10144 bits (MOD(0—31)), and 4 bits of odd parity, 1 bit per byte, MOD Bus 10144 bits (MODP(0—3)). External to MEM 10112, MOD Bus 10144 is a 32 bit bus, comprising bits (MOD(0—31)); parity bits are not read to JP 10114.

Data is written into MEM 10112 through JPD Bus 10142 to FIU 20120. As just described, data format operations may then be performed on this data before it is transferred from FIU 20120 to MC 20116 through MOD Bus 10144. In such operations, JPD Bus 10142 operates as a 32 bit bus carrying 32 bits of data, bits (JPD (0—31)), with no parity bits. JO Port 21010 generates parity for JPD Bus 10142 data to be written into MEM 10112 as this data is transferred into MEM 10112.

Memory requests are also transmitted to MEM 10112 from JP 10114 through JPD Bus 10142, which operates in this regard as a 40 bit bus. Each such request includes an address field, a length field, an FIU

Petitioner Apple Inc. - Exhibit 1004, p. 5219

field specifying data formating operations to be performed, operation code field, and a destination code field specifying destination of data read from MEM 10112. Address field includes a 13 bit physical page number field, (JPPN(0—12)), and a 14 bit physical page offset field, (JPPO(0—13)). Length field includes 6 bits of length information, (JLNG(0—5)), and expresses true length of the data item to be written to or read from MEM 10112.

As JPD Bus 10142 and MOD Bus 10144 are each capable of transferring 32 bits of data in a single MEM 10112 read or write cycle, 6 bits of length information are required to express true length. As will be described in a following description, JP 10114 may provide physical page offset and length information directly to MEM 10112, performs logical page number to physical page number translations, and may perform a Protection Mechanism 10230 check on the resulting physical page number. As such, MEM 10112 expects to receive (JPPN(0—12)) later than (JPPO(0—13)) and (JLNG(0—5)). (JPPO(0—13)) and (JLNG(0—5)) should, however, be valid during the system clock cycle in which a JP 10114 memory request is loaded into MEM 10112.

Operation code field provided to MEM 10112 from JP 10114 is a 3 bit code, (JMCMD(0—2)) specifying an operation to be formed by MEM 10112. Certain operations which JP 10114 may request of MEM 10112, and their corresponding operation codes, are:

000 = read;
001 = read and set;
010 = write;
011 = error;
100 = error;
101 = read error log and reset;
110 = repair block; and,
111 = flush cache.

Two bit FIU field, (JFIU(0—1)) specifies data manipulation operations to be performed in executing read and write operations. Among the data manipulation operations which may be requested by JP 10114, and their FIU fields, are:

00 = right justified, zero fill;
01 = right justified, sign extend;
10 = left justify, zero fill; and,
11 = left justify, blank fill.

For write operations, JPO Port 21010 may respond only to the most significant bit of FIU field, that is the FIU field bit specifying alignment.

Finally, destination field is a two bit field specifying a JP 10114 destination for data read from MEM 10112. This field is ignored for write operations to MEM 10112. A first bit of destination field, JPMDST, identifies the destination to be FU 10120, and the second field, EBMDST, specifies EU 10120 as the destination.

JPMC Bus 10147 includes dedicated lines for exchange of control signals between JPO Port 21010 and JP 10114. Among these control signals is Load JO Request (LJOR), which is asserted by JP 10114 when JP 10114 wishes to load a request into MEM 10112. LJOR is asserted concurrently with presentation of the memory request to MEM 10112 through PD Bus 10146. JO Port Available (JOPA) is asserted by MEM 10112 when JPO Port 21010 is available to accept a new memory request from JP 10114. If LJOR and JOPA are asserted concurrently, MEM 10112 accepts the memory request from JP 10114 and MEM 10112 drops JOPA to indicate that memory request has been accepted. As previously discussed, MEM 10112 may assert JOPA while a previous request is being executed and the PD Bus 10146 information, that is the memory request previously provided concerning the previous request, is no longer required.

If JP 10114 submits a memory request and JOPA is not asserted by MEM 10112, MEM 10112 does not accept the request and JP 10114 must resubmit that request when JOPA is asserted. Because, as described above, JPPN field of a memory request from JP 10114 may arrive late compared to the other fields of the request, MEM 10112 will delay loading of JPPN field for a particular request until the next system clock cycle after the request was initially submitted. MEM 10112 may also obtain this JPPN field at the same time it is being loaded into the port register by by-passing the port register.

JP 10114 may abort a memory request upon asserting Abort JP Request (ABJR). ABJR will be accepted by MEM 10112 during system clock cycle after accepting memory request from JP 10114 and ABJR will result in cancellation of the requested operation. A single ABJR line is provided for both JPO Port 21010 and JPI Port 21110 because, as described in a following description, MEM 10112 may accept only a single request from JP 10114, to either JPO Port 21010 or to JPI port 21110, during a single system clock cycle.

Upon completion of an operand read operation requested through JPO Port 21010 MEM 10112 may assert either of two data available signals to JP 10114. These signals are data available for FA(DAVFA) and data available for EB(DAVEB). As previously described, a part of each read request from JP 10114 includes a destination field specifying the intended destination of the requested data. As will be described further

Petitioner Apple Inc. - Exhibit 1004, p. 5220

below, MEM 10112 tracks such destination information for read requests and returns destination information with a corresponding information in the form of DAVFA and DAVEB. DAVFA indicates a destination in FU 10120 while DAVEB indicates a destination in EU 10122. MEM 10112 may also assert signal zero filled (ZFILL) specifying whether read data for JPO Port 21010 is zero filled. ZFILL is valid only when DAVEB is asserted.

For JPO Port 21010 write request, the associated write data word should be valid on same system clock cycle as the request, or one system clock cycle later. JP 10114 asserts Load JP Write Data (LJWD) during the system clock cycle when JP 10114 places valid write data on JPD Bus 10142.

As previously discussed, when MEM 10112 detects an error in servicing a JP 10114 request MEM 10112 places a record of this error in MC 20116's Error Log. When an entry is placed in Error Log for either JPO Port 21010 or IO Port 20910, MEM 10112 asserts an interrupt flag signal indicating a valid Error Log entry is present. DP 10118 detects this flag signal and may direct the flag signal to either JP 10114 or IOS 10116, or both. IOS 10116 or JP 10114, as selected by DP 10118, may then read and reset Error Log and reset the flag. The interrupt flag signal is not necessarily directed to the requestor, JP 10114 or IOS 10116, whose request resulted in the error.

If an uncorrectible MEM 10112 error, that is an error in two or more bits of a single data word, is detected in a read operation the incorrect data is read to JP 10114 and an invalid data signal asserted. A signal, Previous MOD Transfer Invalid (PMODI), is asserted by MEM 10112 on the next system clock cycle following either DAVFA or DAVEB. PMODI is not asserted for single bit errors, instead the data is corrected and the corrected data read to JP 10114.

Having described JPO Port 21010's structure, and characteristics, JPI Port 21110 will be described next below.

3. JPI Port 21110 Operating Characteristics (Fig. 211)

Referring to Fig. 211, a diagramic representation of JPI Port 21110 is shown. JPI port 21110 includes an address input from PD Bus 10146 to FIU 20120, a data output to MOD Bus 10144 from MC 20116, and bi-directional control inputs and outputs from MIC 20122 to JPMC Bus 10147. As previously described, a primary function of JPI Port 21110 is the transfer of SOPs and operand NAMEs from MEM 10112 to JP 10114 upon request from JP 10114. JPI Port thereby performs only read operations wherein each read operation is a transfer of a single 32 bit word having a word aligned address.

Referring to JPI Port 21110 input from PD Bus 10146, read requests to MEM 10112 by JP 10114 for SOPs and operand NAMEs each comprise a 21 bit word address. As described above, each JPI Port 21110 read operation is of a single 32 bit word. As such, the five least significant bits of address are ignored by MEM 10112. For the same reason, a JPI Port 21110 request to MEM 10112 does not include a length field, an operation code field, an FIU field, or a destination code field. Length, operation code, and FIU code fields are not required since JPI Port 21110 performs only a single type of operation and destination code field is not required because destination is inherent in a JPI Port 21110 request.

The 32 bit words read from MEM 10112 in response to JPI Port 21110 requests are transferred to JP 10114 through MC 20116's 32 bit output to MOD Bus 10144. As in the case of JPO 21010 read outputs to JP 10114, JPI Port 21110 does not provide parity information to JP 10114.

Control signals exchange between JP 10114 and JPI Port 21110 through JPMC Bus 10147 include Load JI Request (LJIR) and JI Port Available (JIPA), which operate in the same manner as discussed with reference to JPO Port 21010. As previously described, JPO Port 21010 and JPI Port 21110 share a single Abort JP Request (ABJR) command. Similarly, JPO Port 21010 and JPI Port 21110 share previous MOD Transfer Invalid (PMODI) from MEM 10112. As described above, a JPI port 21110 request does not include a destination field as destination is implied. MEM 10112 does, however, provide a Data Available Signal (DAVFI) to JP 10114 when a word read from MEM 10112 in response to a JPI Port 21110 request is present on MOD Bus 10144 and valid.

Having described the overall structure and operation of MEM 10112, and the structure and operation of MEM 10112's interface to JP 10114 and IOS 10116, the structure and operation of FIU 20120 MEM 10112 will next be described in further detail.

h. FIU 20120 (Figs. 201, 230, 231)

As previously described, FIU 20120 performs certain data manipulation operations, including those operations necessary to make MEM 10112 bit addressable. Data manipulation operations may be performed on data being written into MEM 10112, for example, JP 10114 through JPD Bus 10142 or from IOS 10116 through IOM Bus 10130. Data manipulations operations may also be performed on data being read from MEM 10112 to JPD 10114 or IOS 10116. In case of data read to JP 10114, MOD Bus 10144 is used both as a MEM 10112 internal bus, in transferring data from MC 20116 to FIU 20120 for manipulation, and to transfer manipulated data from MEM 10112 to JP 10114. In case of data read to IOS 10116, MOD Bus 10144 is again used as MEM 10112 internal bus to read data from MC 20116 to FIU 20120 for subsequent manipulation. The manipulated data is then read from FIU 20120 to IOS 10116 through MIO Bus 10129.

Certain data manipulation operations which may be performed by FIU 20120 have been previously described. In general, a data manipulation operation consists of four distinct operations, and FIU 20120 may manipulate data in any possible manner which may be achieved through performing any combination

Petitioner Apple Inc. - Exhibit 1004, p. 5221

of these operations. These four possible operations are selection of data to be manipulated, rotation or shifting of that data, masking of that data, and transfer of that manipulated data to a selected destination. Each FIU 20120 data input will comprise a thirty-two bit data word and, as described above, may be selected from input provided from JPD Bus 10142, MOD Bus 10144, and IOM Bus 10130. In certain cases, an FIU 20120 data input may comprise two thirty-two bit words, for example, when a cross word operation is performed generating an output comprised of bits from each of two different thirty-two bit words. Rotation or shifting of a selected thirty-two bit data word enables bits within a selected word to be repositioned with respect to word boundaries. When used in conjunction with the masking operation, described momentarily, rotation and shifting may be reiterably performed to transfer any selected bits in a word to any selected locations in that word. As will be described further below, a masking operation allows any selected bits of a word to be affectively erased, thus leaving only certain other selected bits, or certain selected bits to be forced to predetermined values. A masking operation may be performed, for example, to zero fill or sign extend portions of a thirty-two bit word. In conjunction with a rotation or shifting operation, a masking operation may, for example, select a single bit of a thirty-two bit input word, position that bit in any selected bit location, and force all other bits of that word to zero. Each output of FIU 20120 is a thirty-two bit data word and, as described above, may be transferred on to MOD Bus 10144 or onto MIO Bus 10129. As will be described below, selection of a particular sequence of the above four operations to be performed on a particular data word is determined by control inputs provided from MIC 20122. These control inputs from MIC 20122 are decoded and executed by microinstruction control logic included within FIU 20120.

Referring to Fig. 230, a partial block diagram of FIU 20120 is shown. As indicated therein, FIU 20120 includes Data Manipulation Circuitry (DMC) 23010 and FIU Control Circuitry (FIUC) 23012. Data Manipulation Circuitry 23010 in turn includes FIUIO circuitry (FIUIO) 23014, Data Shifter (DS) 23016, Mask Logic (MSK) 23018, and Assembly Register (AR) logic 23020. Data manipulation circuitry 23010 will be described first followed by FIUC 23012. In describing data manipulation circuitry 23010, FIUIO 23014 will be described first, followed by DS 23016, MSK 23018, and AR 23020, in that order.

Referring to FIUIO 23014, FIUIO 23014 comprises FIU 20120's data input and output circuitry. Job Processor Write Data Register (JWDR) 23022, IO System Write Data Register (IWDR) 23024, and Write Input Data Register (RIDR) 23026 are connected from, respectively, JPD Bus 10142, IOM Bus 10130, and MOD Bus 10144 for receiving data word inputs from, respectively, JP 10114, IOS 10116, and MC 20116. JWDR 23022, IWDR 23024 and RIDR 23026 are each thirty-six bit registers comprised, for example, of SN74S374 registers. Data words transferred into IWDR 23024 and RIDR 23026 are each, as previously described, comprised of a thirty-two data word plus four bits of parity. Data inputs from JP 10114 are, however, as previously described, thirty-two bit data words without parity. Job Processor Parity Generator (JPPG) 23028 associated with JWDR 23022 is connected from JPD Bus 10142 and generates four bits of parity for each data input to JWDR 23022. JWDR 23022's thirty-six bit input thereby comprises thirty-two bits of data, directly from JPD Bus 10142, plus a corresponding four bits of parity from JPPG 23028.

Data words, thirty-two bits of data plus four bits of parity, are transferred into JWDR 23022, IWDR 23024, or RIDR 23026 when, respectively, input enable signals Load JWD (LJWD), Load IWD (LIWD) or Load RID (LRID) are asserted. LJWD is provided from FU 10120 while LIWD and LRID are provided from MIC 20122.

Data words resident in JWDR 23022, IWDR 23024, or RIDR 23026 may be selected and transferred onto FIU 20120's Internal Data (IB) Bus 23030 by output enable signals JWD Enable Output (JWDEO), IWD Enable Output (IWDEO), an RID Enable Output (RIDEO). JWDEO, IWDEO, and RDIEO are provided from FIUC 23012 described below.

As will be described further below, manipulated data words from DS 23016 or AR 23020 will be transferred onto, respectively, Data Shifter Output (DSO) Bus 23032 or Assembly Register Output (ASYRO) Bus 23034 for subsequent transfer onto MOD Bus 10144 or MIO Bus 10129. Each manipulated data word appearing on DSO Bus 23032 or ASYRO Bus 23034 will be comprised of 32 bits of data plus 4 bits of parity. Manipulated data words present on DSO Bus 23032 may be transferred onto MOD Bus 10144 or MIO Bus 10129 through, respectively, DSO Bus To MOD Bus Driver Gate (DSMOD) 23036 or BSO Bus To MIO Bus Driver Gate (DSMIO) 23038. Manipulated data words present on ASYRO Bus 23034 may be transferred onto MOD Bus 10144 or MIO Bus 10129 through, respectively, ASYRO Bus To MOD Bus Driver Gate (ASYMOD) 23040 or ASYRO Bus To MIO Bus Driver Gate (ASYMIO) 23042. DSMOD 23036, DSMIO 23038, ASYMOD 23040, and ASYMIO 23042 are each comprised of, for example, SN74S244 drivers. A manipulated data word on DSO Bus 23032 be transferred through DSMOD 23036 to MOD Bus 10144 when driver gate enable signal Driver Shift To MOD (DRVSHFMOD) to DSMOD 23036 is asserted. Similarly, a manipulated data word on DSO Bus 23032 will be transferred through DSMIO 23038 to MIO Bus 10129 when driver gate enable signal Drive Shift Through MIO Bus (DRVSHFMIO) to DSMIO 23038 is asserted. Manipulated data words present on ASYRO Bus 23034 may be transferred onto MOD Bus 10144 or MIO Bus 10129 when, respectively, driver gate enable signal Drive Assembly To Mod Bus (DRVASYMOD) to ASYMOD 23040 or Drive Assembly To MIO Bus (DRVASYMIO) to ASYMIO 23042 are asserted. DRVSHFMOD, DRVSHFMIO, DRVASYMOD, and DRVASYMIO are provided, as described below, from FIUC 23012.

Registers IARM 23044 and BARMR 23046, which will be described further in a following description of DP 10118, are used by DP 10118 to, respectively, write data words onto IB 23030 and to Read data words

Petitioner Apple Inc. - Exhibit 1004, p. 5222

from MOD Bus 10144, for example manipulated data words from FIU 20120. Data word written into IARMR 23044 from DP 10118, that is 32 bits of data and 4 bits of parity, will be transferred onto IB Bus 23030 when register enable output signal IARM enable output (IARMEO) from FIUC 23012 is asserted. Similarly, a data word present on MOD Bus 10144, comprising 32 bits of data plus 4 bits of parity, will be written into BARMR 23046 when load enable signal Load BARMR (LDBARMR) to BARMR 23046 is asserted by MIC 20122. A data word written into BARMR 23046 from MOD Bus 10144 may then subsequently be read to DP 10118. IARMR 23044 and BARMR 23046 are similar to JWDR 23022, IWDR 23024, and IRDR 23026 and may be comprised, for example, of SN74S299 registers.

Referring finally to IO Parity Check Circuit (IOPC) 23048, IOPC 23048 is connected from IB Bus 23030 to receive each data word, that is 32 bits of data plus 4 bits of parity, appearing on IB Bus 23030. IOPC 23048 confirms parity and data validity of each data word appearing on IB Bus 23030 and, in particular, determines validity of parity and data of data words written into FIU 20120 from IOS 10116. IOPC 23048 generates output Parity Error (PER), previously discussed, indicating a parity error in data words from IOS 10116.

Referring to DS 23016, DS 23016 includes Byte Nibble Logic (BYNL) 23050, Parity Rotation Logic (PRL) 23052, and Bit Scale Logic (BSL) 23054. BYNL 23050, PRL 23052, and BSL 23054 may respectively be comprised of, for example, 25S10 shifters. BYNL 23050 is connected from IB Bus 23030 for receiving and shifting the 32 data bits of a data word selected and transferred onto IB Bus 23030. PRL 23052 is a 4 bit register similarly connected from IB Bus 23030 to receive and shift the 4 parity bits of a data word selected and transferred onto IB Bus 23030. Outputs of BYNL 23050 and PRL 23052 are both connected onto DSO Bus 23032, thus providing a 36 bit FIU 20120 data word output directly from BYNL 23050 and PRL 23052. BYNL 23050's 32 bit data output is also connected to BSL 23054's input. BSL 23054's 32 bit output is in turn provided to MSK 23018.

As previously described, DS 23016 performs data manipulation operations involving shifting of bits within a data word. In general, data shift operations performed by DS 23016 are rotations wherein data bits are right shifted, with least significant bits of data word being shifted into most significant bit position and most significant bits being translated towards least significant bit positions. DS 23016 rotation operations are performed in two stages. First stage is performed by BYNL 23050 and PRL 23052 and comprises right rotations on a nibble basis (a nibble is defined as 4 bits of data). That is, BYNL 23050 right shifts a data word by an integral number of 4 bit increments. A right rotation on a nibble by nibble basis may, for example, be performed when RM 20722 asserts FLIPHALF previously described. FLIPHALF is asserted for IOS 10116 half word read operations wherein the request data resides in the most significant 16 bits of a data word from MC 20116. BYNL 23050 will perform a right rotation of 4 nibbles to transfer the desired 16 bits of data into the least significant 16 bits of BYNL 23050's output. Resulting BYNR 23050 output, together PRL 23052's parity bit output would then be transferred through DSO 23050 to MIO Bus 10129. In addition to performing data shifting operations, DS 23016 may transfer a data word, that is the 32 bits of data, directly to MSK 23018 when data manipulation to be performed does not require data shifting, that is shifts of 0 bits may be performed.

Because data bits are shifted by BYNL 23050 on a nibble basis, the relationship between the 32 data bits of a word and the corresponding 4 parity bits may be maintained if parity bits are similarly right rotated by an amount corresponding to right rotation of data bits. This relationship is true if the data word is shifted in multiples of 2 nibbles, that is 8 bits or 1 byte. PRL 23052 right rotates the 4 parity bits of a data word by an amount corresponding to right rotation of the corresponding 32 data bits in BYNL 23050. Right rotated outputs of BYNL 23050 and PRL 23052 therefore comprise a valid data word having 32 bits of data and 4 bits of parity wherein the parity bits are correctly related to the data bits. A right rotated data word output from BYNL 23050 and PRL 23052 may be transferred onto DSO Bus 23032 for subsequent transfer to MOD Bus 10144 or MIO Bus 10129 as described above. DSO 23032 is used as FIU 20120's output data path for byte write operations and "rotate read" operations wherein the required manipulation of a particular data word requires only an integral numer of right rotations by bytes. Amount of right rotation of 32 bits of data in BYNL 23050 and 4 bits of parity in PRL 23052 is controlled by input signal shift (SHFT) (0—2) to BYNL 23050 and PRL 23052. As will be described below, SHFT (0—2) is generated, together with SHFT (3—4) controlling BSL 23054, by FIUC 23012. BYNL 23050 and PRL 23052, like BSL 23054 described below, are parallel shift logic chips and entire rotation operation of BYNL 23050 and PRL 23052 or BSL 23054 may be performed in a single clock cycle.

Second stage of rotation is performed by BSL 23054 which, as described above, receives the 32 data bits of a data word from BYNL 23050. BSL 23054 performs right rotation on a bit by bit basis with the shift amount being selectable between 0—3 bits. Therefore, BSL 23054 may rotate bits through nibble boundaries. BYNL 23050 and BSL 23054 therefore comprise a data shifting circuit capable of performing bit-by-bit right rotation by an amount from 1 bit to a full 32 bit right rotation.

Referring now to MSK 23018, MSK 23018 is comprised of 5 32 bit Mask Word Generators (MWG's) 23056 to 23064. MSK 23018 generates a 32 bit output to AR 23020 by selectively combining 32 bit mask word outputs of MWG's 23056 to 23064. Each mask word generated by one of MWG's 23056 to 23064 is effectively comprised a bit by bit combination of a set of enabling bits and a predetermined 32 bit mask word, generated by FIUC 23012 and MIC 20122. MWG's 23058 to 23064 are each comprised of for example, open collector NAND gates for performing these functions, while NWG 23056 is comprised of a PROM.

Petitioner Apple Inc. - Exhibit 1004, p. 5223

As just described, outputs of MWG's 23056 to 23064 are all open collector circuits so that any selected combination of mask word outputs from MWG's 23056 to 23064 may be ORed together to comprise the 32 bit output of MSK 23018.

MWG 23056 to MWG 23064 generate, respectively, mask word outputs Locked Bit Word (LBW) (0—31), Sign Extended Word (SEW) (0—31), Data Mask Word (DMW) (0—31), Blank Fill Word (BWF) (0—31), and Assembly Register Output (ARO) (0—31). Referring first to MWG 23064 and ARO (0—31), the contents of Assembly Register (ASYMR) 23066 in AR 23020 are passed through MWG 23064 upon assertion of enabling signal Assembly Output Register (ASYMOR). ARO (0—31) is thereby a copy of the contents of ASYMR 23066 and MWG 23064 allows the contents of ASYMR 23066 to be ORed with the selected combination of LBW (0—31), SEW (0—31), DMW (0—31), or BFW (0—31).

DMW (0—31) from MWG 23060 is generated by ANDing enable Input Data Mask (DMSK) (0—31) with the 32 bit output of DS 23016. DMSK (0—31) is a 32 bit enabling word generated, as described below, by FIUC 23012. FIUC 23012 may generate 4 different DMSK (0—31) patterns. Referring to Fig. 231, the 4 DMSKs (0—31) which may be generated by FIUC 20132 are shown. DMSKA (0—31) is shown in Line A of Fig. 231. In DMSKA (0—31) all bits to the left of but not including a bit designated by Left Bit Address (LBA) and all bits to the right of and not including a bit designated by Right Bit Address (RBA) are 0. All bits between, and including, those bits designated by LBA and RBA are 1's. DMSKB (0—31) is shown in Line B of Fig. 231 and is DMSKA (0—31) inverted. DMSKC (0—31) and DMSKD (0—31) are shown, respectively, in Lines C and D of Fig. 231 and are comprised of, respectively, all 0's or all 1's. As stated above DMSK (0—31) is ANDed with the 32 bit output of DS 23016. As such, DMSKC (0—31) may be used, for example, to inhibit DS 23016's output while DMSKD (0—31) may be used, for example, to pass DS 23016's output to AR 23020. DMSKA (0—31) and DMSKB (0—31) may be used, for example, to gate selected portions of DS 23016's output to AR 23020 where, for example, the selected portions of DS 23016's output may be ORed with other mask word outputs MSK 23018.

Referring next to MWG 23062, MWG 23062 generates BFW (0—31). BFW (0—31) is used in a particular operation wherein 32 bit data words containing 1 to 4 ASCII blanks are required to be generated wherein 1 bit/byte contains a logic one and remaining bits contain logic zeros. In this case, the ASCII blank bytes may contain logic 1's in bit positions 2, 10, 18, and 26.

Referring again to Fig. 231, Line E therein shows 32 bit right mask (RMSK) (0—31) which may be generated by FIUC 23012. In the most general case, RMSK contains zeros in all bit positions to the left of and including a bit position designated by RBA. When used in a blank fill operation, bit positions 2, 10, 18, and 26 may be selected to contain logic 1's depending upon those byte positions containing logic 1's, that is in those bytes containing ASCII blanks; these bytes to the right of RBA are determined by RMSK (0—31). RMSK (0—31) is enabled through MWG 23062 as BWF (0—31) when MWG 23062 is enabled by blank fill (BLNKFILL) provided from FIU 23012.

As described above, MWG's 23058 to 23064 and in particular MWG's 23060 and MWG 23062 are NAND gate operations. Therefore, the outputs of MWGs 23056 through 23064 are active low signals. The inverted output of ASYMR 23066 is used as an output to ASYRO 23034 to invert these outputs to active high.

MWG 23058, generating SEW (0—31), is used in generating sign extended or filled words. In sign extended words, all bit spaces to the left of the most significant bit of a 32 bit data word are filled with the sign bit of the data contained therein, the left most bits of the 32 bit word are filled with 1's or 0's depending on whether that word's sign bit indicates that the data contained therein is a positive or negative number.

Sign Select Multiplexor (SIGNSEL) 23066 is connected to receive the 32 data bits of a word present on IB Bus 23030. Sign Select (SGNSEL) (0—4) to SIGNSEL 23066 is derived from SBA (0—4), that is from SBA Bus 21226 from PRMUX 20720. As previously described, SBA (0—4) is Starting Bit Address identifying the first or most significant bit of a data word. When a data word contains a signed number, most significant bit contains sign bit of that number. SGNSEL (0—4) input to SIGNSEL 23066 is used as a selection input and, when SIGNSEL is enabled by Sign Extend (SIGNEXT) from FIU 23012, selects the sign bit on IB Bus 23030 and provides that sign bit as an input to MWG 23058.

Sign bit input to MWG 23058 is ANDed with each bit of left hand mask (LMSK) (0—31) from FIUC 23012. Referring again to Fig. 231, LMSK (0—31) is shown on Line F thereof. LMSK (0—31) contains all 0's to the right of and including the bit space identified by LBA and 1's in all bit spaces to the left of that bit space identified by LBA. SEW (0—31) will therefore contain sign bit in all bit spaces to the left of the most significant bit of the data word present on output of MWG 23058. The data word on IB Bus 23030 may then be passed through DS 23016 and subjected to a DMSK operation wherein all bits to the left of the most significant bit are forced to 0. SEW (0—31) and DMW (0—31) outputs of MWG's 23058 and 23060 may then be ORed to provide the desired find extended word output.

LBW (0—31), provided by MWG 23056, is used in locked bit operations wherein the most significant data bit of a data word is in MEM 10112 forced to logic 1. SIGNSEL (0—4) is an address input to MWG 23056 and, as previously described, indicates most significant data bit of a data word present on an IB Bus 23030. MWG 23056 is enabled by input Lock (LOCK) from FIUC 23012 and the resulting LBW (0—31) will contain a single logic 1 in the bit space of the most significant data bit of the data word present on IB Bus 23030. The data word present on IB Bus 23030 may then be passed through DS 23016 and MWG 23060 to be ORed with LBW (0—31) so that that data words most significant data bit is forced to logic 1.

Referring to AR 23020, AR 23020 includes ASYMR 23066, which may be comprised for example of a

Petitioner Apple Inc. - Exhibit 1004, p. 5224

SN74S175 registers, and Assembly Register Parity Generator (ASYPG) 23070. As previously described, ASYMR 23066 is connected from MSK 23018 32 bit output. A 32 bit word present on MSK 23018's output will be transferred into ASYMR 23066 when ASYMR 23066 is enabled by Assembly Register Load (ASYMLD) from MIC 20122. The 32 bit word generated through DS 23016 and MSK 23018 will then be present on ASYRO Bus 23034 and may, as described above, then be transferred onto MOD Bus 10144 or MIO Bus 10129. ASYPG 23070 is connected from ASYMR 23066 32 bit output and will generate 4 parity bits for the 32 bit word presently on the 32 data lines of ASYRO Bus 23034. ASYPG 23070's 4 bit parity output is bused on the 4 parity bit lines of ASYRO Bus 23034 and accompany the 32 bit data word present thereon.

Having described structure and operation of Data Manipulation Circuitry 23010, FIUC 23012 will be described next below.

Referring again to Fig. 230, FIUC 23012 provides pipelined microinstruction control of FIU 20120. That is, control signals are received from MIC 20122 during a first clock cycle and certain of the control signals are decoded by microinstruction logic to generate further FIUC 23012 control signals. During the second clock cycle, control signals received and generated during first clock cycle are provided to DMC 23010, some of which are further decoded to provide yet other control signals to control operation of FIUC 23012. FIUC 23012 includes Initial Decode Logic (IDL) 23074, Pipeline Registers (PPLR) 23072, Final Decoding Logic (FDL) 23076, and Enable Signal Pipeline Register (ESPR) 23098 with Enable Signal Decode Logic (ESDL) 23099.

IDL 23074 and Control Pipeline Register (CPR) 23084 of PPLR 23072 are connected from control outputs of MIC 20122 to receive control signals therefrom during a first clock cycle as described above. IDL 23074 provides outputs to control pipeline registers Right Bit Address Register (RBAR) 23086, Left Bit Address Register (LBAR) 23088 and Shift Register (SHFR) 23090 of PPLR 23072. CPR 23084 and SHFR 23090 provide control outputs directly to DMC 23010. As described above these outputs control DMC 23010 during second clock cycle.

CPR 23084, RBAR 23086, and LBAR 23088 provide outputs to FDL 23076 during second clock cycle and FDL 23076 in turn provides certain outputs directly to DMC 23010.

ESPR 23098 and ESDL 23099 receive enable and control signals from MIC 20122 and in turn provide enable and control signals to DMC 23010 and certain other portions of MEM 10112 circuitry.

IDL 23074 and FDL 23076 may be comprised, for example, of PROMs. CPR 23084, RBAR 23086, LBAR 23088, SHFR 23090, and ESPR 23098 may be comprised, for example, of SN74S194 registers. ESDL 23099 may be comprised of, for example, compatible decoders, such as logic gates.

Referring first to IDL 23074, IDL 23074 performs an initial decoding of circuitry control signals from MIC 20122 and provides further control signals used by FIUC 23012 in controlling FIU 20120. IDL 23074 is comprised of read-only memory arrays Right Bit Address Decoding Logic (RBADL) 23078, Left Bit Address Decoding Logic (LBADL) 23080, and Shift Amount Decoding Logic (SHFAMTDL) 23082. RBADL 23078 receives, as address inputs, Final Bit Address (FBA) (0—4), Bit Length Number (BLN) (0—4), and Starting Bit Address (SBA) (0—4). FBA, BLN and SBA define, respectively, the final bit, length, and starting bit of a requested data item as previously discussed with reference to PRMUX 20720. RBADL 23078 also receives chip select enable signals Address Translation Chip Select (ATCS) 00, 01, 02, 03, 04, and 15 from MIC 20122 and, in particular, RM 20722. When FIU 20120 is required to execute certain MSK 23018 operations, inputs FBA (0—4), BLN (0—4), and SBA (0—4), together with an ATCS input, are provided to RBADL 23078 from MIC 20122. RBADL 23078 in turn provides output RBA (Right Bit Address) (0—4), which has been described above with reference to DMSK (0—31) and RMSK (0—31). LBADL 23080 is similar to RBADL 23078 and is provided with inputs BLN (0—4), FBA (0—4), SBA (0—4), and ATCS 06, 07, 08, 09, and 05 from MIC 20122. Again, for certain MSK 23018 operations, LBADL 23080 will generate Left Bit Address (LBA) (0—4), which has been previously discussed above with reference to DMSK (0—31) and LMSK (0—31).

RBA (0—4) and LBA (0—4) are, respectively, transferred to RBAR 23086 and LBAR 23088 at start of second clock cycle by Pipeline Load Enable signal PIPELD provided from MIC 20122. RBAR 23086 and LBAR 23088 in turn respectively provide outputs Register Right Address (RRAD) (0—4) and Register Left Address (RLAD) (0—4) as address inputs to Right Mask Decode Logic (RMSKDL) 23092, Left Mask Decode Logic (LMSKDL) 23094, and FDL 23076 at start of second clock cycle. RRAD (0—4) and RLAD (0—4) correspond respectively to RBA (0—4) and LBA (0—4).

RMSKDL 23092 and LMSKDL 23094 are ROM arrays, having, as just described, RRAD (0—4) and RLAD (0—4) as, respectively, address inputs and Mask Enable (MSKENBL) from CPR 23084 as enable inputs. Together, RMSKDL 23092 and LMSKDL 23094 generate, respectively, RMSK (0—31) and LMSK (0—31) to MSK 23018. RMSK (0—31) and LMSK (0—31) are provided as inputs to Exclusive Or/Exclusive Nor gating (XOR/XNOR) 23096. XOR/XNOR 23096 also receives enable and selection signal Out Mask (OUTMSK) from CPR 23084. RMSK (0—31) and LMSK (0—31) inputs to XOR/XNOR 23096 are used, as selected by OUTMSK from CPR 23084, to generate a selected DMSK (0—31) as shown in Fig. 231. DMSK (0—31) output of XOR/XNOR 23096 is provided, as described above, to MSK 23018.

Referring again to IDL 23074, SHFAMTBL 23082 decodes certain control inputs from MIC 20122 to generate, through SHFR 23090, control inputs SHFT (0—4) and SGNSEL (0—4) to, respectively, DS 23016, SIGNSEL 23068 and MWG 23056. Address inputs to the PROMs comprising SHFAMTBL 23082 include FBA (0—4), SBA (0—4), and FLIPHALF (FLIPHALF) from MIC 20122. FBA (0—4) and SBA (0—4) have been described above. FLIPHALF is a control signal indicating that, as described above, that 16 bits of data

Petitioner Apple Inc. - Exhibit 1004, p. 5225

requested by IOS 10116 resides in the upper half of a 32 bit data word and causes those 16 bits to be transferred to the lower half of FIU 20120's output data word onto MIO Bus 10129. MIC 20122 also provides chip enable signals ATCS 10, 11, 12, 13, and 14. Upon receiving these control inputs from MIC 20122, SHFAMTDL 23082 generates an output shift amount (SHFAMT) (0—4) which, together with SBA (0—4) from MIC 20122, is transferred into SHFR 23090 by PIPELD at start of second clock cycle. SHFR 23090 then provides corresponding outputs SHFT (0—4) and SIGNSEL (0—4). As described above, SIGNSEL (0—4) are provided to SIGNSEL 23068 and MWG 23056 and MSK 23018. SHFT (0—4) is provided as SHFT (0—2) and SHFT (3—4) to, respectively, BYNL 23050 and BSL 23054 and DS 23016.

Referring to CPR 23084, as described above certain control signals are provided directly to FIU 20120 circuitry without being decoded by IDL 23074 or FDL 23076. Inputs to CPR 23084 include Sign Extension (SIGNEXT) and Lock (LOCK) indicating, respectively, that FIU 20120 is to perform a sign extension operation through MWG 23058 or a lock bit word operation through MWG 23056. CPR 23084 provides corresponding outputs SIGNEXT and LOCK to MSK 23018 to select these operations. Input Assembly Output Register (ASYMOR) and Blank Fill (BLANKFILL) are passed through CPR 23084 as ASYMOR and BLANKFILL to, respectively, MWG 23064 and MWG 23062 to select the output of ASYMR 23066 as a mask or to indicate that MSK 23018 is to generate a blank filled word through MWG 23062. Inputs OUTMSK and MSKENBL to CPR 23084 are provided, as discussed above, as enable signals OUTMSK and MSKENBL to, respectively, EXOR/ENOR 23096 and RMSKDL 23092 and LMSKBL 23094 and generating RMSK (0—31), LMSK (0—31), and DMSK (0—31) as described above.

Referring finally to ESPR 23098 and ESDL 23099, ESPR 23098 and PPLR 23072 together comprise a pipeline register and ESDL 23099 decoding logic for providing enable signals to FIU 20120 and other MEM 10112 circuitry. ESPR 23098 receives inputs Drive MOD Bus (DRVMOD) (0—1), Drive MIO Bus (DRVMIO) (0—1), and Enable Register (ENREG) (0—1) from MIC 20122 as previously described. DRVMOD (0—1), DRVMIO (0—1), and ENREG (0—1) are transferred into ESPR 23098 by PIPELD as previously described with reference to PPLR 23072. ESPR 23098 provides corresponding outputs to ESDL 23099, which in turn decodes DRVMOD (0—1), DRVMIO (0—1), and ENREG (0—1) to provide enable signals to FIU 20120 and other MEM 10112 circuitry. Outputs DRVSHFMOD, DRVASYMOD, DRVSHFMIO, and DRVASYMIO are provided to DSMOD 23036, DSMIO 23038, ASYMOD 23040, ASYMIO 23042, and FIUIO 23014 to control transfer of FIU 20120 manipulated data words onto MOD Bus 10144 and MIO Bus 10129. Outputs IARMEO, JWDEO, IWDEO, and RIDEO are provided as output enable signals to IARMR 23044, JWDR 23022, IWDR 23024, and RIDR 23026 to transfer the contents of these registers onto IB Bus 23030 as previously described. Outputs DRVCAMOD, DRVAMIO, DRVBYMOD, and DRVBYMIO are provided to MC 20116 for use in controlling transfer of information onto MOD Bus 10144 and MIO Bus 10129.

Having described the structure and operation of MEM 10112 above, the structure and operation of FU 10120 will be described next below.

B. Fetch Unit 10120 (Figs. 202, 206, 101, 103, 104, 238)

As has been previously described, FU 10120 is an independently operating, microcode controlled machine comprising, together with EU 10122, CS 10110's micromachine for executing user's programs. Principal functions of FU 10120 include: (1) Fetching and interpreting instructions, that is SINs comprising SOPs and Names, and data from MEM 10112 for use by FU 10120 and EU 10122; (2) Organizing and controlling flow and execution of user programs; (3) Initiating EU 10122 operations; (4) Performing arithmetic and logic operations on data; (5) Controlling transfer of data from FU 10120 and EU 10122 to MEM 10112; and, (6) Maintaining certain stack register mechanisms. Among these stack and register mechanisms are Name Cache (NC) 10226, Address Translation Cache (ATC) 10228, Protection Cache (PC) 10234, Architectural Base Registers (ABRs) 10364, Micro-Control Registers (mCRs) 10366, Micro-Stack (MIS) 10368, Monitor Stack (MOS) 10370 of General Register File (GRF) 10354, Micro-Stack Pointer Register Mechanism (MISPR) 10356, and Return Control Word Stack (RCWS) 10358. In addition to maintaining these FU 10120 resident stack and register mechanisms, FU 10120 generates and maintains, in whole or part, certain MEM 10112 resident data structures. Among these MEM 10112 resident data structures are Memory Hash Table (MHT) 10716 and Memory Frame Table (MFT) 10718, Working Set Matrix (WSM) 10720, Virtual Memory Management Request Queue (VMMRQ) 10721, Active Object Table (AOT) 10712, Active Subject Table (AST) 10914, and Virtual processor State Blocks (VPSBs) 10218. In addition, a primary function of FU 10120 is the generation and manipulation of logical descriptors which, as previously described, are the basis of CS 10110's internal addressing structure. As will be described further below, while FU 10120's internal structure and operation allows FU 10120 to execute arithmetic and logic operations, FU 10120's structure includes certain features to expedite generation and manipulation of logical descriptors.

Referring to Fig. 202, a partial block diagram of FU 10120 is shown. To enhance clarity of presentation, certain interconnections within FU 10120, and between FU 10120 and EU 10122 and MEM 10112 are not shown by line connections but, as described further below, are otherwise indicated, such as by common signal names. Major functional elements of FU 10120 include Descriptor Processor (DESP) 20210, MEM 10112 Interface Logic (MEMINT) 20212, and Fetch Unit Control Logic (FUCTL) 20214. DSP 20210 is, in general, an arithmetic and logic unit for generating and manipulating entries for MEM 10112 and FU 10120 resident stack mechanisms and caches, as described above, and, in particular, for generation and manipulation of logical descriptors. In addition, as stated above, DSP 20210 is a general purpose Central

Petitioner Apple Inc. - Exhibit 1004, p. 5226

Processor Unit (CPU) capable of performing certain arithmetic and logic functions.

DESP 20210 includes AON Processor (AONP) 20216, Offset Processor (OFFP) 20218, Length Processor (LENP) 20220. OFFP 20218 comprises a general, 32 bit CPU with additional structure to optimize generation and manipulation of offset fields of logical descriptors. AONP 20216 and LENP 20220 comprise, respectively, processors for generation and manipulation of AON and length fields of logical descriptors and may be used in conjuction with OFFP 20218 for execution of certain arithmetic and logical operations. DESP 20210 includes GRF 10354, which in turn include Global Registers (GRs) 10360 and Stack Registers (SRs) 10362. As previously described, GR's 10360 includes ABRs 10364 and mCRs 10366 while SRs 10362 includes MIS 10368 and MOS 10370.

MEMINT 20212 comprises FU 10120's interface to MEM 10112 for providing Physical Descriptors (physical addresses) to MEM 10112 to read SINs and data from and write data to MEM 10112. MEMINT 20212 includes, among other logic circuitry, MC 10226, ATC 10228, and PC 10234.

FUCTL 20214 controls fetching of SINs and data from MEM 10112 and provides sequences of microinstructions for control of FU 10120 and EU 10122 in response to SOPs. FUCTL 20214 provides Name inputs to MC 10226 for subsequent fetching of corresponding data from MEM 10112. FUCTL 20214 includes, in part, MISPR 10356, RCWS 10358, Fetch Unit S-Interpreter Dispatch Table (FUSDT) 11010, and Fetch Unit S-Interpreter Table (FUSITT) 11012.

Having described the overall structure of FU 10120, in particular with regard to previous descriptions in Chapter 1 of this description, DESP 20210, MEMINT 20212, and FUCTL 20214 will be described in further detail below, and in that order.

1. Description Processor 20210 (Figs. 202, 101, 103, 104, 238, 239)

As described above, DESP 20210 comprises a 32 bit CPU for performing all usual arithmetic and logic operations on data. In addition, a primary function of DESP 20210 is generation and manipulation of entries for, for example, Name Tables (NTs) 10350, ATC 10228, and PC 10234, and generation and manipulation of logical descriptors. As previously described, with reference to CS 10110 addressing structure, logical descriptors are logical addresses, or pointers, to data stored in MEM 10112. Logical descriptors are used, for example, as architectural base pointers or microcontrol pointers in ABRs 10364 and mCRs 10366 as shown in Fig. 103, or as linkage and local pointers of Procedure Frames 10412 as shown in Fig. 104. In a further example, logical descriptors generated by DESP 20210 and corresponding to certain operand Names are stored in MC 10226, where they are subsequently accessed by those Names appearing in SINs fetched from MEM 10112 to provide rapid translation between operand Names and corresponding logical descriptors.

As has been previously discussed with reference to CS 10110 addressing structure, logical descriptors provided to ATU 10228, from DESP 20210 or NC 10226, are translated by ATU 10228 to physical descriptors which are actual physical addresses of corresponding data stored in MEM 10112. That data subsequently is provided to JP 10114, and in particular to FU 10120 or EU 10122, through MOD Bus 10144.

As has been previously discussed with reference to MEM 10112, each data read to JP 10114 from MEM 10112 may contain up to 32 bits of information. If a particular data item referenced by a logical descriptor contains more than 32 bits of data, DESP 20210 will, as described further below, generate successive logical descriptors, each logical descriptor referring to 32 bits or less of information, until the entire data item has been read from MEM 10112. In this regard, it should be noted that NC 10226 may contain logical descriptors only for data items of 255 bits or less in length. All requests to MEM 10112 for data items greater than 32 bits in length are generated by DESP 20210. Most of data items operated on by CS 10110 will, however, be 32 bits or less in length so that NC 10226 is capable of handling most operand Names to logical descriptor translations.

As described above, DESP 20210 includes AONP 20216, OFFP 20218, and LENP 20220. OFFP 20218 comprises a general purpose 32 bit CPU with additional logic circuitry for generating and manipulating table and cache entries, as described above, and for generating and manipulating offset fields of AON pointers and logical descriptors. AONP 20216 and LENP 20220 comprise logic circuitry for generating and manipulating, respectively, AON and length fields of AON pointers and logical descriptors. As indicated in Fig. 202, GRF 10354 is vertically divided in three parts. A first part resides in ANOP 20216 and, in additon to random data, contains AON fields of logical descriptors. Second and third parts reside, respectively, in OFFP 20218 and LENP 20220 and, in addition to containing random data, respectively contain offset and length fields of logical descriptors. AON, Offset, and length portions of GRF 10354 residing respectively in AONP 20216, OFFP 20218, and LENP 20220 are designated, respectively, as AONGRF, OFFGRF, and LENGRF. AONGRF portion of GRF 10354 is 28 bits wide while OFFGRF and LENGRF portions of GRF 10354 are 32 bits in width. Although shown as divided vertically into three parts, GRF 10354 is addressed and operates as a unitary structure. That is, a particular address provided to GRF 10354 will address corresponding horizontal segments of each of GRF 10354's three sections residing in AONP 20216, OFFP 20218, and LENP 20220.

a. Offset Processor 20218 Structure

Referring first to OFFP 20218, in addition to being a 32 bit CPU and generating and manipulating table and cache entries and offset fields of AON pointers and logical descriptors, OFFP 20218 is DESP 20210's

Petitioner Apple Inc. - Exhibit 1004, p. 5227

primary path for receiving data from and transferring data to MEM 10112. OFFP 20218 includes Offset Input Select Multiplexer (OFFSEL) 20238, OFFGRF 20234, Offset Multiplexer Logic (OFFMUX) 20240, Offset ALU (OFFALU) 20242, and Offset ALU A Inputs Multiplexer (OFFALUSA) 20244.

OFFSEL 20238 has first and second 32 bit data inputs connected from, respectively, MOD Bus 10144 and JPD Bus 10142. OFFSEL 20238 has a third 32 bit data input connected from a first output of OFFALU 20242, a fourth 28 bit data input connected from a first output of AONGRF 20232, and a fifth 32 bit data input connected from OFFSET Bus 20228. OFFSEL 20238 has a first 32 bit output connected to input of OFFGRF 20234 and a second 32 bit output connected to a first input of OFFMUX 20240. OFFMUX 20240 has second and third 32 bit data inputs connected from, respectively, MOD Bus 10144 and JPD Bus 10142. OFFMUX 20240 also has a fourth 5 bit data input connected from Bias Logic (BIAS) 20246 and LENP 20220, described further below, and fifth 16 bit data input connected from NAME Bus 20224. Thirty-two bit data output of OFFGRF 20234 and first 32 bit data output of OFFMUX 20240 are connected to, respectively, first and second data inputs of OFFALUSA 20244. A first 32 bit data output of OFFALUSA 20244 and a second 32 bit data output of OFFMUX 20240 are connected, respectively, to first and second data inputs of OFFALU 20242. A second 32 bit data output of OFFALUSA 20244 is connected to OFFSET Bus 20228. A first 32 bit data output of OFFALU 20242 is connected to JPD Bus 10142, to a first input of AON Input Select Multiplexer (AONSEL) 20248 and AONP 20216, and, as described above, to a third input of OFFSEL 20238. A second 32 bit data output of OFFALU 20242 is connected to OFFSET Bus 20228 and third 16 bit output is connected to NAME Bus 20224.

b. AON Processor 20216 Structure

Referring to AONP 20216, a primary function of AONP 20216 is that of containing AON fields of AON pointers and logical descriptors. In addition, those portions of AONGRF 20232 not otherwise occupied by AON pointers and logical descriptors may be used as a 28 bit wide general register area by JP 10114. These portions of AONGRF 20232 may be so used either alone or in conjunction with corresponding portions of OFFGRF 20234 and LENGRF 20236. AONP 20216 includes AONSEL 20248 and AONGRF 20232. As previously described, a first 32 bit data input AONSEL 20248 is connected from a first data output of OFFALU 20242. A second 28 bit data input of AONSEL 20248 is connected from 28 bit output of AONGRF 20232 and from AON Bus 20230. A third 28 bit data input of AONSEL 20248 is connected from logic zero, that is a 28 bit input wherein each input bit is set to logic zero. Twenty-eight bit data output of AONSEL 20248 is connected to data input of AONGRF 20232. As just described, 28 bit data output of AONGRF 20232 is connected to second data input of AONSEL 20248, and is connected to AON Bus 20230.

c. Length Processor 20220 Structure

Referring finally to LENP 20220, a primary function of LENP 20220 is the generation manipulation of length fields of AON pointers and physical descriptors. In addition, LENGRF 20236 may be used, in part, either alone or in conjunction with corresponding address spaces of AONGRF 20232 and OFFGRF 20234, as general registers for storage of data. LENP 20220 includes Length Input Select Multiplexer (LENSEL) 20250, LENGRF 20236, BIAS 20246, and Length ALU (LENALU) 20252. LENSEL 20250 has first and second data inputs connected from, respectively, LENGTH Bus 20226 and OFFSET Bus 20228. LENGTH Bus 20226 is eight data bits, zero filled while OFFSET Bus 20228 is 32 data bits. LENSEL 20250 has a third 32 bit data input connected from data output of LENALU 20252. Thirty-two bit data output of LENSEL 20250 is connected to data input of LENGRF 20236 and to a first data input of BIAS 20246. Second and third 32 bit data inputs of BIAS 20246 are connected from, respectively, Constant (C) and Literal (L) outputs of FUSITT 11012 as will be described further below. Thirty-two bits data output of LENGRF 20236 is connected to JPD Bus 10142, to Write Length Input (WL) input of NC 10226, and to a first input of LENALU 20252. Five bit output of BIAS 20246 is connected to a second input of LENALU 20252, to LENGTH Bus 20226, and, as previously described, to a fourth input of OFFMUX 20240. Thirty-two bit output of LENALU 20252 is connected, as stated above, to third input of LENSEL 20250.

Having described the overall operation and the structure of DESP 20210, operation of DESP 20210 will be described next below in further detail.

d. Descriptor Processor 20210 Operation

a.a. Offset Selector 20238

Referring to OFFP 20218, GRF 10354 includes GR's 10360 and SR's 10362. GR's 10360 in turn contain ABR's 10364, mCR's 10366, and a set of general registers. SR's 10362 include MIS 10368 and MOS 10370. GRF 10354 is vertically divided into three parts. AONGRF 20232 is 28 bits wide and resides in AONP 20216, LENGRF 10354 is 32 bits wide and resides in LENP 20220, and OFFGRF 20234 is 32 bits wide and resides in OFFP 20218. AONGRF 20232, OFFGRF 20234, and LENGRF 20236 may be comprised of Fairchild 93422s.

In addition to storing offset fields of AON pointers and logical descriptors, those portions of OFFGRF 20234 not reserved for ABR's 10365, mCR's 10366, and SR's 10362 may be used as general registers, alone or in conjunction with corresponding portions AONGRF 20232 and LENGRF 20236, when OFFP 20218 is being utilized as a general purpose, 32 bit CPU. OFFGRF 20234 as will be described further below, is addressed in parallel with AONGRF 20232 and LENGRF 20236 by address inputs provided from FUCTL 20214.

Petitioner Apple Inc. - Exhibit 1004, p. 5228

OFFSEL 20238 is a multiplexer, comprised for example of SN74S244s and SN74S257s, for selecting data inputs to be written into selected address locations of OFFGRF 20234. OFFSEL 20238's first data input is from MOD Bus 10144 and is the primary path for data transfer between MEM 10112 and DESP 20210. As previously described, each data read from MEM 10112 to JP 10114 is a single 32 bit word where between one and 32 bits may contain actual data. If a data item to be read from MEM 10112 contains more than 32 bits of data, successive read operations are performed until the entire data item has been transferred.

OFFSEL 20238's second data input is from JPD Bus 10142. As will be described further below, JPD Bus 10142 is a data transfer path by which data outputs of FU 10120 and EU 10122 are written into MEM 10112. OFFSEL 20238's input of JPD Bus 10142 thereby provides a wrap around path by which data present at outputs of FU 10120 or EU 10122 may be transferred back into DESP 20210 for further use. For example, as previously stated a first output of OFFALU 20242 is connected to JPD Bus 10142, thereby allowing data output of OFFP 20218 to be returned to OFFP 20218 for further processing, or to be transferred to AONP 20216 or LENP 20220 as will be described further below. In addition, output of LENGRF 20236 is also connected to JPD Bus 10142 so that length fields of AON pointers or physical descriptors, or data, may be read from LENGRF 20236 to OFFP 20218. This path may be used, for example, when LENGRF 20236 is being used as a general purpose register for storing data or intermediate results of arithmetic or logical operations.

OFFSEL 20238's third input is provided from OFFALU 20242's output. This data path thereby provides a wrap around path whereby offset fields or data residing in OFFGRF 20234 may be operated on and returned to OFFGRF 20234, either in the same address location as originally read from or to a different address location. OFFP 20218 wrap around path from OFFALU 20242's output to OFFSEL 20238's third input, and thus to OFFGRF 20234, may be utilized, for example, in reading from MEM 10112 a data item containing more than 32 bits of data. As previously described, each read operation from MEM 10112 to JP 10114 is of a 32 bit word wherein between one and 32 bits may contain actual data. Transfer of a data word containing more than 32 bits is accomplished by performing a succession of read operations from MEM 10112 to JP 10114. For example, if a requested data item contains 70 bits of data, that data item will be transferred in three consecutive read operations. First and second read operations will each transfer 32 bits of data, and final read operation will transfer the remaining 6 bits of data. To read a data item of greater than 32 bits from MEM 10112 therefore, DESP 20210 must generate a sequence of logical descriptors, each defining a successive 32 bit segment of that data item. Final logical descriptor of the sequence may define a segment of less than 32 bits, for example, six bits as in the example just stated. In each successive physical descriptor, offset field must be incremented by value of length field of the preceding physical descriptor to define starting addresses of successive data items segments to be transferred. Length field of succeeding physical descriptors will, in general, remain constant at 32 bits except for final transfer which may be less than 32 bits. Offset field will thereby usually be incremented by 32 bits at each transfer until final transfer. OFFP 20218's wrap around data path from OFFALU 20242's output to their input of OFFSEL 20238 may, as stated above, be utilized in such sequential data transfer operations to write incremented or decremented offset field of a current physical descriptor back into OFFGRF 20234 to be offset field of a next succeeding physical descriptor.

In a further example, OFFP 20218's wrap around path from OFFALU 20242's output to third input of OFFSEL 20238 may be used in resolving Entries in Name Tables 10350, that is Name resolutions. In Name resolutions, as previously described, offset fields of AON pointers, for example Linkage Pointers 10416, are successively added and subtracted to provide a final AON pointer to a desired data item.

OFFSEL 20238's fourth input, from AONGRF 20232's output, may be used to transfer data or AON fields from AONGRF 20232 to OFFGRF 20234 or OFFMUX 20240. This data path may be used, for example, when OFFP 20218 is used to generate AON fields of AON pointers or physical descriptors or when performing Name evaluations.

Finally, OFFSEL 20238's fifth data input from OFFSET Bus 20228 allows offset fields on OFFSET Bus 20228 to be written into OFFGRF 20234 or transferred into OFFMUX 20240. This data path may be used, for example, to copy offset fields to OFFGRF 20234 when JP 10114 is performing a Name evaluation.

Referring now to OFFMUX 20240, OFFMUX 20240 includes logic circuitry for manipulating individual bits of 32 bit words. OFFMUX 20240 may be used, for example, to increment and decrement offset fields by length fields when performing string transfers, and to generate entries for, for example, MHT 10716 and MFT 10718. OFFMUX 20240 may also be used to aid in generating and manipulating AON, OFFSET, and LENGTH fields of physical descriptors and AON pointers.

b.b. Offset Multiplexer 20240 Detailed Structure (Fig. 238)

Referring to Fig. 238, a more detailed, partial block diagram of OFFMUX 20240 is shown. OFFMUX 20240 includes Offset Multiplexer Input Selector (OFFMUXIS) 23810, which for example may be comprised of SN74S373s and SN74S244s and Offset Multiplexer Register (OFFMUXR) 23812, which for example may be comprised of SN74S374s. OFFMUX 20240 also includes Field Extraction Circuit (FEXT) 23814, which may for example be comprised of SN74S257s, and Offset Multiplexer Field Selector (OFFMUXFS) 23816, which for example may be comprised of SN74S257s and SN74S374s. Finally, OFFMUX 20240 includes Offset Scaler (OFFSCALE) 23818, which may for example be comprised of AMD 25S10s, Offset Inter-element Spacing Encoder (OFFIESENC) 23820, which may for example be comprised of Fairchild 93427s

Petitioner Apple Inc. - Exhibit 1004, p. 5229

and Offset Multiplexer Output Selector (OFFMUXOS) 23822, which may for example be comprised of AMD 25Ss, Fairchild 93427s, and SN74S244s.

Referring first to OFFMUX 20240's connections to other portions of OFFP 20218, OFFMUX 20240's first data input, from OFFSEL 20238, is connected to a first input of OFFMUXIS 23810. OFFMUX 20240's second input, from MOD Bus 10144, is connected to a second input of OFFMUXIS 23810. OFFMUX 20240's third input, from JPD Bus 10142, is connected to a first input of OFFMUXFS 23816 while OFFMUX 20240's fourth input, from BIAS 20246, is connected to a first input of OFFMUXOS 23822. OFFMUX 20240's fifth input, from NAME Bus 20224, is connected to a second input of OFFMUXFS 23816. OFFMUX 20240's first output, to OFFALUSA 20244, is connected from output of OFFMUXR 23812 while OFFMUX 20240's second output, to OFFALU 20242, is connected from output of OFFMUXOS 23822.

Referring to OFFMUX 20240's internal connections, 32 bit output of OFFMUXIS 23810 is connected to input OFFMUXR 23812 and 32 bit output of OFFMUXR 23812 is connected, as described above, as first output of OFFMUX 20240, and as a third input of OFFMUXFS 23816. Thirty-two bit output of OFFMUXR 23812 is also connected to input of FEXT 23814. OFFMUXFS 23816's first, second and third inputs are connected as described above. A fourth input of OFFMUXFS 23816 is a 32 bit input wherein 31 bits are set to logic zero and 1 bit to logic 1. A fifth input is a 32 bit input wherein 31 bits are set to logic 1 and 1 to logic 0. A sixth input of OFFMUXFS 23816 is a 32 bit literal (L) input provided from FUSITT 11012 and is a 32 bit binary number comprising a part of a microinstruction FUCTL 20214, described below. OFFMUXFS 23816's seventh and eighth input are connected from FEXT 23814. Input 7 comprises FIU and TYPE fields of Name Table Entries which have been read into OFFMUXR 23812. Input 8 is a general purpose input conveying bits extracted from a 32 bit word captured in OFFMUXR 23812. As indicated in Fig. 238, OFFMUXFS 23816's first, third, fourth, fifth, and sixth inputs are each 32 bit inputs which are divided to provide two 16 bit inputs each. That is, each of these 32 bit inputs is divided into a first input comprising bit 0 to 15 of that 32 bit input, and a second input comprising bits 16 to 31.

Thirty-two bit output of OFFMUXFS 23816 is connected to inputs of OFFSCALE 23818 and OFFIESENC 23820. As indicated in Fig. 238, Field Select Output (FSO) of OFFMUXFS 23816 is a 32 bit word divided into a first word including 0 to 15 and a second word including bits 16 to 31. Output FSO of OFFMUXFS 23816, as will be described further below, thereby reflects the divided structure of OFFMUXFS 23816's first, third, fourth, fifth, and sixth inputs.

Logical functions performed by OFFMUXFS 23816 in generating output FSO, and which will be described in further detail in following descriptions, include:

(1) Passing the contents of OFFMUXR 23812 directly through OFFMUXFS 23816;

(2) Passing a 32 bit word on JPD Bus 10142 directly through OFFMUXFS 23816;

(3) Passing a literal value comprising a part of a microinstruction from FUCTL 20214 directly through OFFMUXFS 23816;

(4) Forcing FSO to be literal values 0000 0000;

(5) Forcing FSO to be literal value 0000 001;

(6) Extracting Name Table Entry fields;

(7) Accepting a 32 bit word from OFFMUXR 23812 or JPD Bus 10142, or 32 bits of a microinstruction from FUCTL 20214, and passing the lower 16 bits while forcing the upper 16 bits to logic 0;

(8) Accepting a 32 bit word from OFFMUXR 23812 or JPD Bus 10142, or 32 bits of miroinstruction from FUCTL 20214, and passing the higher 16 bits while forcing the lower 16 bits to logic 0;

(9) Accepting a 32 bit word from OFFMUXR 23812, or JPD Bus 10142, or Name Bus 20224, or 32 bits of a microinstruction from FUCTL 20214, and passing the lower 16 bits while sign extending bit 16 to the upper 16 bits; and,

(10) Accepting a 32 bit word from Name Bus 20224 and passing the lowest 8 bits while sign extending bit 24 to the highest 24 bits.

Thirty-two bit output of OFFSCALE 23818 and 3 bit output of OFFIESENC 23820 are connected, respectively, to second and third inputs of OFFMUXOS 23822. OFFMUXOS 23822's first input is, as described above, OFFMUX 20240's fourth input and is connected from output BIAS 20246. Finally, OFFMUXOS 23822's 32 bit output, OFFMUX (0—31) is OFFMUX 20240's second output and as previously described as connected to a second input of OFFALU 20242.

c.c. Offset Multiplexer 20240 Detailed Operation

a.a.a. Internal Operation

Having described the structure of OFFMUX 20240 as shown in Fig. 238, operation of OFFMUX 20240 will be described below. Internal operation of OFFMUX 20240, as shown in Fig. 238, will be described first, followed by description of OFFMUX 20240's operation with regard to DESP 20210.

Referring first to OFFMUXR 23812, OFFMUXR 23812 is a 32 bit register receiving either a 32 bit word from MOD Bus 10144, MOD (0—31), or a 32 bit word received from OFFSEL 20238, OFFSEL (0—31), and is selected by OFFMUXIS 23810. OFFMUXR 23812 in turn provides those selected 32 bit words from MOD Bus 10144 or OFFSEL 20238 as OFFMUX 20240's first data output to OFFALUSA 20244, as FEXT 23814's input, and as OFFMUXFS 23816's third input. OFFMUXR 23812's 32 bit output to OFFMUXFS 23816 is provided as two parallel 16 bit words designated as OFFMUXR output (OFFMUXRO) (0—15) and (16—31). As described above, OFFMUXFS 23816's output to OFFALUSA 20244 from OFFMUXR 23812 may be right shifted 16

Petitioner Apple Inc. - Exhibit 1004, p. 5230

places and the highest 16 bits zero filled.

FEXT 23814 receives OFFMUXRO (0—15) and (16—31) from OFFMUXR 23812 and extracts certain fields from those 16 bit words. In particular, FEXT 23814 extracts FIU and TYPE fields from NT 10350 Entries which have been transferred into OFFMUXR 23812. FEXT 23814 may then provide those FIU and TYPE fields as OFFMUXFS 23816's seventh input. FEXT 23814 may, selectively, extract certain other fields from 32 bit words residing in OFFMUXR 23812 and provide those fields as OFFMUXFS 23816's eighth input.

OFFMUXFS 23816 operates as a multiplexer to select certain fields from OFFMUXFS 23816's eight inputs and provide corresponding 32 bit output words, Field Select Output (FSO), comprised of those selected fields from OFFMUXFS 23816's inputs. As previously described, FSO is comprised of 2, parallel 16 bit words, FSO (0—15) and FSO (16—31). Correspondingly, OFFMUX 20240's third input, from JPD Bus 10142, is a 32 bit input presented as two 16 bit words, JPD (0—15) and JPD (16—31). Similarly, OFFMUXFS 23816's fourth, fifth, and sixth inputs are each presented as 32 bit words comprised of 2, parallel 16 bit words, respectively, "0" (0—15) and (16—31), "1" (0—15) and (16—31), and L (0—15) and (16—31). OFFMUXFS 23816's second input, from NAME Bus 20224, is presented as a single 16 bit word, NAME (16—31), while OFFMUXFS 23816's inputs from FEXT 23814 are each less than 16 bits in width. OFFMUXFS 23816 may, for a single 32 bit output word, select FSO (0—15) to contain one of corresponding 16 bit inputs JPD (0—15), "0" (0—15), "1" (0—15), or L (0—15). Similarly, FSO (16—31) of that 32 bit output word may be selected to contain one of NAME (16—31), JPD (16—31), 0 (16—31), 1 (16—31), L (16—31), or inputs 7 and 8 from FEXT 23814. OFFMUXFS 23816 therefore allows 32 bit words, comprised of two 16 bit fields, to be generated from selected portions of OFFMUXFS 23816's inputs.

OFFMUXFS 23816 32 bit output is provided as inputs to OFFSCALE 23818 and OFFIESENC 23820. Referring first to OFFIESENC 23820, OFFIESENC 23820 is used, in particular, in resolving, or evaluating, NT 10350 Entries (NTEs) referring to arrays of data words. As indicted in Fig. 108, word D of an NTE contains certain information relating to inter-element spacing (IES) of data words of an array. Word D of an NTE may be read from MEM 10112 to MOD Bus 10144 and through OFFMUX 20240 to input of OFFIESENC 23820. OFFIESENC 23820 then examines word D's IES field to determine whether inter-element spacing of that array is a binary multiple, that is 1, 2, 4, 8, 16, 32, or 64 bits. In particular, OFFIESENC 23820 determines whether 32 bit word D contains logic zeros in the most significant 25 bits and a single logic one in the least significant 7 bits. If inter-element spacing is such a binary multiple, starting addresses of data words of that array may be determined by left shifting of index (IES) to obtain offset fields of physical addresses of words in the array and a slower and more complex multiplication operation is not required. In such cases, OFFIESENC generates a first output, IES Encodeable (IESENC) to FUCTL 20214 to indicate that inter-element spacing may be determined by simple left shifting. OFFIESENC 23820 then generates encoded output, Encoded IES (ENCIES), to OFFMUXOS 23822. ENCIES is then a coded value specifying the amount of left shift necessary to translate index (IES) value into offsets of words in that array. As indicated in Fig. 238, ENCIES is OFFMUXOS 23822's third input.

OFFSCALE 23818 is a left shift shift network with zero fill of least significant bits, as bits are left shifted. Amount of shift by OFFSCALE 23818 is selectable between zero and 7 bits. Thirty-two bit words transferred into OFFSCALE 23818 from OFFSCALE 23818 from OFFMUXFS 23816 may therefore be left shifted, bit by bit, to selectively reposition bits within that 32 bit input word. In conjunction with OFFMUXFS 23816, and a wrap around connection provided by OFFALU 20242's output to OFFSEL 20238, OFFSCALE 23818 may be used to generate and manipulate, for example, entries for MHT 10716, MFT 10718, AOT 10712, and AST 10914, and other CS 10110 data structures.

OFFMUXOS 23822 is a multiplexer having first, second, and third inputs from, respectively, BIAS 20246, OFFSCALE 23818, OFFIESENC 23820. OFFMUXOS 23822 may select any one of these inputs as OFFMUX 20240's second output, OFFMUX (0—31). As previously described, OFFMUX 20240's second output is connected to a second input of OFFALU 20242.

Having described internal of OFFMUX 20240, operation of OFFMUX 20240 with regard to overall operation of DESP 20210 will be described next below.

b.b.b. Operation Relative to Descriptor Processor 20210

OFFMUX 20240's first input, from OFFSEL 20238, allows inputs to OFFSEL to be transferred through OFFMUXIS 23810 and into OFFMUXR 23812. This input allows OFFMUXR 23812 to be loaded, under control of FUCTL 20214 microinstructions, with any input of OFFSEL 20238. In a particular example, OFFALU 20242's output may be fed back through OFFSEL 20238's third input and OFFMUX 20240's first input to allow OFFMUX 20240 and OFFALU 20242 to perform reiterative operations on a single 32 bit word.

OFFMUX 20240's second input, from MOD Bus 10144, allows OFFMUXR 23812 to be loaded directly from MOD Bus 10144. For example, NTEs from a currently active procedure may be loaded into OFFMUXR 23812 to be operated upon as described above. In addition, OFFMUX 20240's second input may be used in conjunction with OFFSEL 20238's first input, from MOD Bus 10144, as parallel input paths to OFFP 20218. These parallel input paths allow pipelining of OFFP 20218 operations by allowing OFFSEL 20238 and OFFGRF 20234 to operate independently from OFFMUX 20240. For example, FU 10120 may initiate a read operation from MEM 10112 to OFFMUXR 23812 during a first microinstruction. The data so requested will appear on MOD Bus 10144 during a second microinstruction and may be loaded into OFFMUXR 23812 through OFFMUX 20240's second input from MOD Bus 10144. Concurrently, FU 10120 may initiate, at start

Petitioner Apple Inc. - Exhibit 1004, p. 5231

of second microinstruction, an independent operation to be performed by OFFSEL 20238 and OFFGRF 20234, for example loading output of OFFALU 20242 into OFFGRF 20234. Therefore, by providing an independent path into OFFMUX 20240 from MOD Bus 10144, OFFSEL 20238 is free to perform other, concurrent data transfer operations while a data transfer from MOD Bus 10144 to OFFMUX 20240 is being performed.

OFFMUX 20240's third input, from JPD Bus 10142, is a general purpose data transfer path. For example, data from LENGRF 20236 or OFFALU 20242 may be transferred into OFFMUX 20240 through JPD Bus 10142 and OFFMUX 20240's third input.

OFFMUX 20240's fourth input is connected from BIAS 20246 and primarily used during string transfers as described above. That is, length fields of physical descriptors generated for a string transfer may be transferred into OFFMUX 20240 through OFFMUX 20240's fourth input to increment or decrement, offset fields of those physical descriptors in OFFALU 20242.

OFFMUX 20240's fifth input is connected from NAME Bus 20224. As will be described further below, Names are provided to NC 10226 by FUCTL 20214 to call, from MC 10226, logical descriptors corresponding to Names appearing on MOD Bus 10144 as part of sequences of SINs.

As each Name is presented to NC 10226, that Name is transferred into and captured in Name Trap (NT) 20254. Upon occurrence of an NC 10226 miss, that is NC 10226 does not contain an entry corresponding to a particular Name, that Name is subsequently transferred from NT 20254 to OFFMUX 20240 through NAME Bus 20224 and OFFMUX 20240's fifth input. That Name, which is previously described as an 8, 12, or 16 bit binary number, may then be scaled, that is multiplied by a NTE size. That scaled Name may then be added to Name Table Pointer (NTP) from mCRs 10366 to obtain the address of a corresponding NTE in an NT 10350. In addition, a Name resulting in a NC 10226 miss, or a page fault in the corresponding NT 10350, or requiring a sequence of Name resolves, may be transferred into OFFGRF 20234 from OFFMUX 20240, through OFFALU 20242 and OFFSEL 20238 third input. That Name may subsequently be read, or restored, from OFFGRF 20234 as required.

Referring now to outputs of OFFMUX 20240, OFFMUX 20240's first output, from OFFMUXR 23812, allows contents of OFFMUXR 23812 to be transferred to first input of OFFALU 20242 through OFFALUSA 20244. OFFMUX 20240's second output, from OFFMUXOS 23822, is provided directly to second input of OFFALU 20242. OFFALU 20242 may be concurrently provided with a first input from OFFMUXR 23812 and a second input, for example a manipulated offset field, from OFFMUXOS 23822.

Referring to OFFALUSA 20244, OFFALUSA 20244 is a multiplexer. OFFALUSA 20244 may select either output of OFFGRF 20234 or first output of OFFMUX 20240 to be either first input of OFFALU 20242 or to be OFFP 20218's output to OFFSET Bus 20228. For example, an offset field from OFFGRF 20234 may be read to OFFSET Bus 20228 to comprise offset field of a current logical descriptor, and concurrently read into OFFALU 20242 to be incremented or decremented to generate offset field of a subsequent logical descriptor in a string transfer.

OFFALU 20242 is a general purpose, 32 bit arithmetic and logic unit capable of performing all usual ALU operations. For example, OFFALU 20242 may add, subtract, increment, or decrement offset fields of logical descriptors. In addition, OFFALU 20242 may serve as a transfer path for data, that is OFFALU 20242 may transfer input data to OFFALU 20242's outputs without operating upon that data. OFFALU 20242's first output, as described above, is connected to JPD Bus 10142, to third input of OFFSEL 20238, and to first input of AONSEL 20248. Data transferred or manipulated by OFFALU 20242 may therefore be transferred on to JPD Bus 10142, or wrapped around into OFFP 20218 through OFFSEL 20238 for subsequent or reiterative operations. OFFALU 20242's output to AONSEL 20248 may be used, for example, to load AON fields of AON pointers or physical descriptors generated by OFFP 20218 into AONGRF 20232. In addition, this data path allows FU 10120 to utilize AONGRF 20232 as, for example, a buffer or temporary memory space for intermediate or final results of FU 10120 operations.

OFFALU 20242's output to OFFSET Bus 20228 allows logical descriptor offset fields to be transferred onto OFFSET Bus 20228 directly from OFFALU 20242. For example, a logical descriptor offset field may be generated by OFFALU 20242 during a first clock cycle, and transferred immediately onto OFFSET Bus 20228 during a second clock cycle.

OFFALU 20242's third output is to NAME Bus 20224. As will be described further below, NAME Bus 20224 is address input (ADR) to NC 10226. OFFALU 20242's output to NAME Bus 20224 thereby allows OFFP 20218 to generate or provide addresses, that is Names, to NC 10226.

Having described operation of OFFP 20218, operation of LENP 20220 will be described next below.

e. Length Processor 20220 (Fig. 239)

Referring to Fig. 202, a primary function of LENP 20220 is generation and manipulation of logical descriptor length fields, including length fields of logical descriptors generated in string transfers. LENP 20220 includes LENGRF 20236, LENSEL 20250, BIAS 20246, and LENALU 20252. LENGRF 20236 may be comprised, for example, of Fairchild 93422s. LENSEL 20250 may be comprised of, for example, SN74S257s, SN74S157s, and SN74S244s, and LENALU 20252 may be comprised of, for example, SN74S381s.

As previously described, LENGRF 20236 is a 32 bit wide vertical section of GRF 10354. LENGRF 20236 operates in parallel with OFFGRF 20234 and AONGRF 20232 and contains, in part, length fields of logical descriptors. In addition, also as previously described, LENGRF 20236 may contain data.

Petitioner Apple Inc. - Exhibit 1004, p. 5232

LENSEL 20250 is a multiplexer having three inputs and providing outputs to LENGRF 20236 and first input of BIAS 20246. LENSEL 20250's first input is from Length Bus 20226 and may be used to write physical descriptor or length fields from LENGTH Bus 20226 into LENGRF 20236 or into BIAS 20246. Such length fields may be written from LENGTH Bus 20226 to LENGRF 20236, for example, during Name evaluation or resolve operations. LENSEL 20250's second input is from OFFSET Bus 20228. LENSEL 20250's second input may be used, for example, to load length fields generated by OFFP 20218 into LENGRF 20236. In addition, data operated upon by OFFP 20218 may be read into LENGRF 20236 for storage through LENSEL 20250's second input.

LENSEL 20250's third input is from output of LENALU 20252 and is a wrap around path to return output of LENALU 20252 to LENGRF 20256. LENSEL 20250's third input may, for example, be used during string transfers when length fields of a particular logical descriptor is incremented or decremented by LENALU 20252 and returned to LENGRF 20236. This data path may also, for example, be used in moving a 32 bit word from one location in LENGRF 20236 to another location in LENGRF 20236. As stated above, LENSEL 20250's output is also provided to first input BIAS and allows data appearing at first, second, or third inputs of LENSEL 20250 to be provided to first input of BIAS 20246.

BIAS 20246, as will be described in further detail below, generates logical descriptor length fields during string transfers. As described above, no more than 32 bits of data may be read from MEM 10112 during a single read operation. A data item of greater than 32 bits in length must therefore be transferred in a series, or string, of read operations, each read operation transferring 32 bits or less of data. String transfer logical descriptor length fields generated BIAS 20246 are provided to LENGTH Bus 20226, to LENALU 20252 second input, and to OFFMUX 20240's fourth input, as previously described. These string transfer logical descriptor length fields, referred to as bias fields are provided to LENGTH Bus 20226 by BIAS 20246 to be length fields of the series of logical descriptors generated by DESP 20210 to execute a string transfer. These bias fields are provided to fourth input OFFMUX 20240 to increment or decrement offset fields of those logical descriptors, as previously described. These bias fields are provided to second input of LENALU 20252, during string transfers, to correspondingly decrement the length field of a data item being read to MEM 10112 in a string transfer. BIAS 20246 will be described in greater detail below, after LENALU 20252 is first briefly described.

a.a. Length ALU 20252

LENALU 20252 is a general purpose, 32 bit arithmetic and logic unit capable of executing all customary arithmetic and logic operations. In particular, during a string transfer of a particular data item LENALU 20252 receives that data items length field from LENGRF 20236 and successive bias fields from BIAS 20246. LENALU 20252 then decrements that logical descriptor's current length field to generate length field to be used during next read operation of the string transfer, and transfers new length field back into LENGRF 20236 through LENSEL 20250's third input.

b.b. BIAS 20246 (Fig. 239)

Referring to Fig. 239, a partial block diagram of BIAS 20246 is shown. BIAS 20246 includes Bias Memory (BIASM) 23910, Length Detector (LDET) 23912, Next Zero Detector (NXTZRO) 23914, and Select Bias (SBIAS) 23916. Input of LDET 23912 is first input of BIAS 20246 and connected from output of LENSEL 20250. Output of LDET 23912 is connected to data input of BIASM 23910, and data output of BIASM 23910 is connected to input of NXTZRO 23914. Output of NXTZRO 23914 is connected to a first input of SBIAS 23916. A second input of SBIAS 23916 is BIAS 20246's second input, L8, and is connected from an output of FUCTL 20214. A third input of SBIAS 23916 is BIAS 20246's third input, L, and is connected from yet another output of FUCTL 20214. Output of SBIAS 23916 is output of BIAS 20246 and, as described above, is connected to LENGTH Bus 20226, to a second input of LENALU 20252, and to fourth input of OFFMUX 20240.

BIASM 23910 is a 7 bit wide random access memory having a length equal to, and operating and addressed in parallel with, SR's 10362 of GRF 10354. BIASM 23910 has an address location corresponding to each address location of SR's 10362 and is addressed concurrently with those address locations in SR's 10362. BIASM 23910 may be comprised, for example, of AMD 27SO3As.

BIASM 23910 contains a bias value of each logical descriptor residing in SR's 10362. As described above, a bias value is a number representing number of bits to be read from MEM 10112 in a particular read operation when a data item having a corresponding logical descriptor, with a length field stored LENGRF 20236, is to be read from MEM 10112. Initially, bias values are written into BIASM 23910, in a manner described below, when their corresponding length fields are written into LENGRF 20236. If a particular data item has a length of less than 32 bits, that data item's initial bias value will represent that data items actual length. For example, if a data item has a length of 24 bits the associated bias value will be a 6 bit binary number representing 24. That data item's length field in LENGRF 20236 will similarly contain a length value of 24. If a particular item has a length of greater than 32 bits for example, 70 bits as described in a previous example, that data item must be read from MEM 10112 in a string transfer operation. As previously described, a string transfer is a series of read operations transferring 32 bits at a time from MEM 10112, with a final transfer of 32 bits or less completing transfer of that data item. Such a data item's initial length field entry in LENGRF 20236 will contain, using the same example as previously described, a value of 70.

Petitioner Apple Inc. - Exhibit 1004, p. 5233

That data item's initial bias entry written into a corresponding address space of BIASM 23910 will contain a bias value of 32. That initial bias value of 32 indicates that at least the first read operation required to transfer that data item from MEM 10112 will transfer 32 bits of data.

When a data item having a length of less than 32 bits, for example 24 bits, is to be read from MEM 10112, that data item's bias value of 24 is read from BIASM 23910 and provided to LENGTH Bus 20226 as length field of logical descriptor for that read operation. Concurrently, that bias value of 24 is subtracted from that data items length field read from LENGRF 20236. Subtracting that bias value from that length value will yield a result of zero, indicating that no further read operations are required to complete transfer of that data item.

If a data item having, for example, a length of 70 bits is to be read from MEM 10112, that data item's initial bias value of 32 is read from BIASM 23910 to LENGTH Bus 20226 as length field of first logical descriptor of a string transfer. Concurrently, that data item's initial length field is read from LENGRF 20236. That data item's initial bias value, 32, is subtracted from that data item's initial length value, 70, and LENALU 20252. The result of that subtraction operation is the remaining length of data item to be transferred in one or more subsequent read operations. In this example, subtracting initial bias value from initial length value indicates that 38 bits of that data item remain to be transferred. LENALU 20252's output representing results of this subtraction, for example 38, are transferred to LENSEL 20250's third input to LENGRF 20236 and written into address location from which that data item's initial length value was read. This new length field entry then represents remaining length of that data item. Concurrently, LDET 23912 examines that residual length value being written into LENGRF 20236 to determine whether remaining length of that data item is greater than 32 bits or is equal to or less than 32 bits. If remaining length is greater than 32 bits, LDET 23912 generates a next bias value of 32, which is written into BIASM 23910 and same address location that held initial bias value. If remaining data item length is less than 32 bits, LDET 23912 generates a 6 bit binary number representing actual remaining length of data item to be transferred. Actual remaining length would then, again, be written into BIASM 23910 address location originally containing initial bias value. These operations are also performed by LDET 23912 in examining initial length field and generating a corresponding initial bias value. These read operations are continued as described above until LDET 23912 detects that remaining length field is 32 bits or less, and thus that transfer of that data item will be completed upon next read operation. When this event is detected, LDET 23912 generates a seventh bit input into BIASM 23910, which is written into BIASM 23910 together with last bias value of that string transfer, indicating that remaining length will be zero after next read operation. When a final bias value is read from BIASM 23910 at start of next read operation of that string transfer, that seventh bit is examined by NXTZRO 23914 which subsequently generates a test condition output, Last Read (LSTRD) to FUCTL 20214. FUCTL 20214 may then terminate execution of that string transfer after that last read operation, if the transfer has been successfully completed.

As previously described, the basic unit of length of a data item in CS 10110 is 32 bits. Accordingly, data items of 32 or fewer bits may be transferred directly while data items of more than 32 bits require a string transfer. In addition, transfer of a data item through a string transfer requires tracking of the transferred length, and remaining length to be transferred, of both the data item itself and the data storage space of the location the data item is being transferred to. As such, BIAS 20246 will store, and operate with, in the manner described above, length and bias fields of the logical descriptors of both the data item and the location the data item is being transferred to. FUCTL 20214 will receive an LSTRD test condition if bias field of source descriptor becomes zero before or concurrently with that of the destination, that is a completed transfer, or if bias field of destination becomes zero before that of the source, and may provide an appropriate microcode control response. It should be noted that if source bias field becomes zero before that of the destination, the remainder of the location that this data item is being transferred to will be filled and padded with zeros. If the data item is larger than the destination storage capacity, the destination location will be filled to capacity and FUCTL 20214 notified to initiate appropriate action.

In addition to allowing data item transfers which are insensitive to data item length, BIAS 20246 allows string transfers to be accomplished by short, tight microcode loops which are insensitive to data item length. A string transfer, for example, from location A to location B is encoded as:

(1) Fetch from A, subtract length from bias A, and update offset and length of a; and,

(2) Store to B, subtract length from bias B, and branch to (1) if length of B does not go to zero or fall through (end transfer) if length of B goes to zero. Source (A) length need not be texted as the microcode loop continues until length of B goes to zero; as described above, B will be filled and padded with zeros if length of A is less than length of B, or B will be filled and the string transfer ended if length of A is greater than or equal to length of B.

LDET 23912 and NXTZRO 23914 thereby allow FUCTL 20214 to automatically initiate a string transfer upon occurrence of a single microinstruction from FUCTL 20214 initiating a read operation by DESP 20210. That microinstruction initiating a read operation will then be automatically repeated until LSTRD to FUCTL 20214 from NXTZRO 23914 indicates that the string transfer is completed. LDET 23912 and NXTZRO 23914 may, respectively, be comprised for example of S74S260s, SN74S133s, SN74S51s, SN74S00s, SN74S00s, SN74S04s, SN74S02s, and SN74S32s.

Referring finally to SBIAS 23916, SBIAS 23916 is a multiplexer comprised, for example, of SN74S288s, SN74S374s, and SN74S244s. SBIAS 23916, under microinstruction control from FUCTL 20214, selects BIAS

Petitioner Apple Inc. - Exhibit 1004, p. 5234

20246's output to be one of a bias value from BIASM 23910, L8, or L. SBIAS 23916's first input, from BIASM 23910, has been described above. SBIAS 23916's second input, L8, is provided from FUCTL 20214 and is 8 bits of a microinstruction provided from FUSITT 11012. SBIAS 23916's second input allows microcode selection of bias values to be used in manipulation of length and offset fields of logical descriptors by LENALU 20252 and OFFALU 20242, and for generating entries to MC 10226. SBIAS 23916's third input, L, is similarly provided from FUCTL 20214 and is a decoded length value derived from portions of microinstructions in FUSITT 11012. These microcode length values represent certain commonly occurring data item lengths, for example length of 1, 2, 4, 8, 16, 32, and 64 bits. An L input representing a length of 8 bits, may be used for example in reading data from MEM 10112 on a byte by byte basis.

Having described operation of LENP 20220, operation of AONP 20216 will be described next below.

f. AON Processor 20216

a.a. AONGRF 20232

As described above, AONP 20216 includes AONSEL 20248 and AONGRF 20232. AONGRF 20232 is a 28 bit wide vertical section of GRF 10354 and stores AON fields of AON pointers and logical descriptors. AONSEL 20248 is a multiplexer for selecting inputs to be written into AONGRF 20232. AONSEL 20248 may be comprised, for example of SN74S257s. AONGRF 20232 may be comprised of, for example, Fairchild 93422s.

As previously described, AONGRF 20232's output is connected onto AON Bus 20230 to allow AON fields of AON pointers and logical descriptors to be transferred onto AON Bus 20230 from AONGRF 20232. AONGRF 20232's output, together with a bi-directional input from AON Bus 20230, is connected to a second input of AONSEL 20248 and to a fourth input of AONSEL 20238. This data path allows AON fields, either from AONGRF 20232 or from AON Bus 20230, to be written into AONGRF 20232 or AONGRF 20234, or provided as an input to OFFMUX 20240.

b.b. AON Selector 20248

AONSEL 20248's first input is, as previously described, connected from output of OFFALU 20242 and is used, for example, to allow AON fields generated or manipulated by OFFP 20218 to be written into AONGRF 20232. AONSEL 20248's third input is a 28 bit word wherein each bit is a logical zero. AONSEL 20248's third input allows AON fields of all zeros to be written into AONGRF 20232. An AON field of all zeros is reserved to indicate that corresponding entries in OFFGRF 20234 and LENGRF 20236 are neither AON pointers nor logical descriptors. AON fields of all zeros are thereby reserved to indicate that corresponding entries in OFFGRF 20234 and LENGRF 20236 contain data.

In summary, as described above, DESP 20210 includes AONP 20216, OFFP 20218, and LENP 20220. OFFP 20218 contains a vertical section of GRF 10354, OFFGRF 20234, for storing offset fields of AON pointers and logical descriptors, and for containing data to be operated upon by DESP 20210. OFFP 20218 is principal path for transfer of data from MEM 10112 to JP 10114 and is a general purpose 32 bit arithmetic and logic unit for performing all usual arithmetic and logic operations. In addition, OFFP 20218 includes circuitry, for example OFFMUX 20240, for generation and manipulation of AON, OFFSET, and LENGTH fields of logical descriptors and AON pointers. OFFP 20218 may also generate and manipulate entries for, for example, NC 10226, ATU 10228, PC 10234, AOT 10712, MHT 10716, MFT 10718, and other data and address structures residing in MEM 10112. LENP 20220 includes a vertical section of GRF 10354, LENGRF 20236, for storing length fields of logical descriptors, and for storing data. LENP 20220 further includes BIAS 20246, used in conjunction with LENGRF 20236 and LENALU 20252, for providing length fields of logical descriptors for MEM 10112 read operations and in particular automatically performing string transfers. AONP 20216 similarly includes a vertical section of GRF 10354, AONGRF 20232. A primary function AONGRF 20232 is storing and providing AON fields of AON pointers and logical descriptors.

Having described structure and operation of DESP 20210, structure and operation of Memory Interface (MEMINT) 20212 will be described next below.

2. Memory Interface 20212 (Figs. 106, 240)

MEMINT 20212 comrises FU 10120's interface to MEM 10112. As described above, MEMINT 20212 includes Name Cache (NC) 10226, Address Translation Unit (ATU) 10228, and Protection Cache (PC) 10234, all of which have been previously briefly described. MEMINT 20212 further includes Descriptor Trap (DEST) 20256 and Data Trap (DAT) 20258. Functions performed by MEMINT 20212 includes (1) resolution of Names to logical descriptors, by NC 10226; (2) translation of logical descriptors to physical descriptors, by ATU 10228; and (3) confirmation of access writes to objects, by PC 10234.

As shown in Fig. 202, NC 10226 adress input (ADR) is connected from NAME Bus 20224. NC 10226 Write Length Field Input (WL) is connected from LENGRF 20236's output. NC 10226's Write Offset Field Input (WO) and Write AON Field Input (WA) are connected, respectively, from OFFSET Bus 20228 and AON Bus 20230. NC 10226 Read AON Field (RA), Read Offset Field (RO), and Read Length Field (RL) outputs are connected, respectively, to AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226.

DEST 20256's bi-directional AON (AON), Offset (OFF), and Length (LEN) ports are connected by bi-directional buses to and from, respectively, AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226.

PC 10234 has AON (AON) and Offset (OFF) inputs connected from, respectively, AON Bus 20230 and

Petitioner Apple Inc. - Exhibit 1004, p. 5235

OFFSET Bus 20228. PC 10234 has a Write Entry (WEN) input connected from JPD Bus 10142. ATU 10228 has AON (AON), Offset (OFF), and Length (LEN) inputs connected from, respectively, AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226. ATU 10228's output is connected to physical Descriptor (PD) Bus 10146.

Finally, DAT 20258 has a bi-directional port connected to and from JPD Bus 10142.

a.a. Description Trap 20256 and Data Trap 20258

Referring first to DST 20256 and DAT 20258, DST 20256 is a register for receiving and capturing logical descriptors appearing on AON Bus 20230, OFFSET Bus 20228, and Length Bus 20226. Similarly, DAT 20258 is a register for receiving and capturing data words appearing on JPD Bus 10142. DST 20256 and DAT 20258 may subsequently return captured logical descriptors or data words to, respectively, AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226, and to JPD Bus 10142.

As previously described, many CS 10110 operations, in particular MEM 10112 and JP 10114 operations, are pipelined. That is, operations are overlapped with certain sets within two or more operations being executed concurrently. For example, FU 10120 may submit read request to MEM 10112 and, while MEM 10112 is accepting and servicing that request, submit a second read request. DEST 20256 and DAT 20258 assist in execution of overplapping operations by providing a temporary record of these operations. For example, a part of a read or write request to MEM 10112 by FU 10120 is a logical descriptor provided to ATU 10228. If, for example the first red request just referred to results in a ATU 10228 cache miss or a protection violation, the logical descriptor of that first request must be recovered for subsequent action by CS 10110 as previously described. That logical descriptor will have been captured and stored in DEST 20256 and thus is immediately available, so that DESP 20210 is not required to regenerate that descriptor. DAT 20258 serves a similar purpose with regard to data being written into MEM 10112 from JP 10114. That is, DAT 20258 receives and captures a copy of each 32 bit word transferred onto JPD Bus 10142 by JP 10114. In event of MEM 10112 being unable to accept a write request, that data may be subsequently reprovided from DAT 20258.

b.b. Name Cache 10226, Address Translation Unit 10228, and Protection Cache 10234 (Fig. 106)

Referring to NC 10226, ATU 10228, and PC 10234, these elements of MEMINT 20212 are primarily cache mechanisms to enhance the speed of FU 10120's interface to MEM 10112, and consequently of CS 10110's operation. As described previously, NC 10226 contains a set of logical descriptors corresponding to certain operand names currently appearing in a process being executed by CS 10110. NC 10226 thus effectively provides high speed resolution of certain operand names to corresponding logical descriptors. As described above with reference to string transfers, NC 10226 will generally contain logical descriptors only for data items of less than 256 bits length. NC 10226 read and write addresses are names provided on NAME Bus 20224. Name read and write addresses may be provided from DESP 20210, and in particular from OFFP 20218 as previously described, or from FUCTL 20214 as will be described in a following description of FUCTL 20214. Logical descriptors comprising NC 10226 entries, each entry comprising an AON field, an Offset field, a Length field, are written into NC 10226 through NC 10226 inputs WA, WO, and WL from, respectively, AON Bus 20230, OFFSET Bus 20228, and LENGRF 20236's output. Logical descriptors read from NC 10226 in response to names provided to NC 10226 ADR input are provided to AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226 from, respectively, NC 10226 outputs RA, RO, and RL.

ATU 10228 is similarly a cache mechanism for providing high speed translation of logical to physical descriptors. In general, ATU 10228 will contain, at any given time, a set of logical to physical page number mappings for MEM 10112 read and write requests which are currently being made, or anticipated to be made, to MEM 10112 by JP 10114. As previously described, each physical descriptor is comprised of a Frame Number (FN) field, and Offset Within Frame (O) fields, and a Length field. As discussed with reference to string transfers, a physical descriptor length field, as in a logical descriptor length field, specify a data item of less than or equal to 32 bits length. Referring to Fig. 106C, as previously discussed a logical descriptor comprised of a 14 bit AON field, a 32 bit Offset field, and Length field, wherein 32 bit logical descriptor Offset field is divided into a 18 bit Page Number (P) field and a 14 bit Offset within Page (O) field. In translating a logicl into a physical descriptor, logical descriptor Length and O fields are used directly, as respectively, physical descriptor length and O fields. Logical descriptor AON and P fields are translated into physical descriptor FN field. Because no actual translation is required, ATU 10228 may provide logical descriptor L field and corresponding O field directly, that is without delay, to MEM 10112 as corresponding physical descriptor O and Length fields. ATU 10228 cache entries are thereby comprised of physical descriptor FN fields corresponding to AON and P fields of those logical descriptors for which ATU 10228 has corresponding entries. Because physical descriptor FN fields are provided from ATU 10228's cache, rather than directly as in physical descriptor O and Length fields, a physical descriptor's FN field will be provided to MEM 10112, for example, one clock cycle later than that physical descriptors O and Length fields, as has been previously discussed.

Referring to Fig. 202, physical descriptor FN fields to be written into ATU 10228 are, in general, generated by DESP 20210. FN fields to be written into ATU 10228 are provided to ATU 10228 Data Input (DI) through JPD Bus 10142. ATU 10228 read and write addresses are comprised of AON and P fields of logical

Petitioner Apple Inc. - Exhibit 1004, p. 5236

descriptors and are provided to ATU 10228's AON and OFF inputs from, respectively, AON Bus 20230 and OFFSET Bus 20228. ATU 10228 read and write addresses may be provided from DESP 20210 or, as described further below, from FUCTL 20214. ATU 10228 FN outputs, together with O and Length fields comprising a physical descriptor, are provided to PD Bus 10146.

PC 10234 is a cache mechanism for confirming active procedure's access rights to objects identified by logical descriptors generated as a part of JP 10114 read or write requests to MEM 10112. As previously described access rights to objects are arbitrated on the basis of subjects. A subject has been defined as a particular combination of a principal, process, and domain. A principal, process, and domain are each identified by corresponding UIDs. Each subject having access rights to an object is assigned an Active Subject Number (ASN) as described in a previous description of CS 10110's Protection Mechanism. The ASN of a subject currently active in CS 10110 is stored in ASN Register 10916 in FU 10120. Access rights of a currently active subject to currently active objects are read from those objects Access Control Lists (ACL) 10918 and stored in PC 10234. If the current ASN changes, PC 10234 is flushed of corresponding access right entries and new entries, corresponding to the new ASN, are written into PC 10234. The access rights of a particular current ASN to a particular object may be determined by indexing, or addressing, PC 10234 with the AON identifying that object. Addresses to write entries into or read entries from PC 10234 are provided to PC 10234 AON input from AON Bus 20230. Entries to be written into PC 10234 are provided to PC 10234's WEN input from JPD Bus 10142. PC 10234 is also provided with inputs, not shown in Fig. 202 for purposes of clarity, from FUCTL 20214 indicating the current operation to be performed by JP 10114 with respect to an object being presently addressed by FU 10120. Whenever FU 10120 submits a read or write request concerning a particular object to MEM 10112, AON field of that request is provided as an address to PC 10234. Access rights of the current active subject to that object are read from corresponding PC 10234 entry and compared to FUCTL 20214 inputs indicating the particular operation to be performed by JP 10114 with respect to that object. The operation to be performed by JP 10114 is then compared to that active subject's access rights to that object and PC 10234 provides an output indicating whether that active subject possesses the rights required to perform the intended operation. Indexing of PC 10234 and comparison of access rights to intended operation is performed concurrently with translation of the memory request logical descriptor to a corresponding physical descriptor by ATU 10228. If PC 10234 indicates that that active subject has the required access rights, the intended operation is executed by JP 10114. If PC 10234 indicates that that active subject does not have the required access rights, PC 10234 indicates that a protection mechanism violation has occurred and interrupts execution of the intended operation.

c.c Structure and Operation of a Generalized Cache and NC 10226 (Fig. 240)

Having described overall structure and operation of NC 10226, ATU 10228, and PC 10234, structure and operation of these caches will be described in further detail below. Structure and operation of NC 10226, ATU 10228, and PC 10234 are similar, except that NC 10226 is a four-way set associative cache, ATU 10228 is a threeway set associative cache and PC 10234 is a two-way set associative cache.

As such, the structure and operation of NC 10226, ATU 10228, and PC 10234 will be described by reference to and description of a generalized cache similar but not necessarily identical to each of NC 10226, ATU 10228, and PC 10234. Reference will be made to NC 10226 in the description of a generalized cache next below, both to further illustrate structure and operation of the generalized cache, and to describe differences between the generalized cache and NC 10226. ATU 10228 and PC 10234 will then be described by description of differences between ATU 10228 and PC 10234 and the generalized cache.

Referring to Fig. 240, a partial block diagram of a generalized four-way, set associative cache is shown. Tag Store (TS) 24010 is comprised of Tag Store A (TSA) 24012, Tag Store B (TSB) 24014, Tag Store C (TSC) 24016, and Tag Store D (TSD) 24018. Each of the cache's sets, represented by TSA 24012 to TSD 24018, may contain, for example as in NC 10226, up to 16 entries, so that TSA 24012 to TSD 24018 are each 16 words long.

Address inputs to a cache are divided into a tag field and an index field. Tag fields are stored in the cache's tag store and indexed, that is addressed to be read or written from or to tag store by index field of the address. A tag read from tag store in response to index field of an address is then compared to tag field of that address to indicate whether the cache contains an entry corresponding to that address, that is, whether a cache hit occurs. In, for example, NC 10226, a Name syllable may be comprised of an 8, 12, or 16 bit binary word, as previously described. The four least significant bits of these words, or Names, comprise NC 10226's index field while the remaining 4, 8, or 12 most significant bits comprise NC 10226's tag field. TSA 24012 to TDS 24018 may each, therefore, be 12 entry wide memories to store the 12 bit tag fields of 16 bit names. Index (IND) or address inputs of TSA 24012 to TSD 24018, would in NC 10226, be connected from four least significant bits of NAME Bus 20224 while Tag Inputs (TAGI) of TSA 24012 to TSD 24018 would be connected from the 12 most significant bits of NAME Bus 20224.

As described above, tag outputs of TS 24010 are compared to tag fields of addresses presented to the cache to determine whether the cache contains an entry corresponding to that address. Using NC 10226 as an example 12 bit Tag Outputs (TAGOs) of TSA 24012 to TSD 24018 are connected to first inputs of Tag Store Comparators (TSC) 24019, respectively to inputs of Tag Store Comparitor A (TSCA) 24020, Tag Store Comparitor B (TSCB) 24022, Tag Store Comparitor D (TSCD) 24024, and Tag Store Comparitor E (TSCE) 24026. Second 12 bit inputs of TSCA 24020 to TSCE 24026 may be connected from the 12 most significant

Petitioner Apple Inc. - Exhibit 1004, p. 5237

bits of NAME Bus 20224 to receive tag fields of NC 10226 addresses. TAS 24020 to TSCE 24026 compare tag field of an address to tag outputs read from TSA 24012 to TSE 24018 in response to index field of that address, and provide four bit outputs indicating which, if any, of the possible 16 entries and their associated tag store correspond to that address tag field. TSCA 24020 to TSCE 24026 may be comprised, for example, of Fairchild 93S46s.

Four bit outputs of TSCA 24012 to TSCE 24026 are connected in the generalized cache to inputs of Tag Store Pipeline Registers (TSPR) 24027; respectively to inputs of Tag Store Pipeline Register A (TSPRA) 24028, Tag Store Pipeline Register B (TSPRB) 24030, Tag Store Pipeline Register C (TSPRC) 24032, and Tag Store Pipeline Register D (TSPRD) 24034. ATU 10228 and PC 10234 is pipelined with a single cache access operation being executed in two clock cycles. During first clock cycle tag store is addressed and tags store therein compared to tag field of address to provide indication of whether a cache hit has occurred, that is whether cache contains an entry corresponding to a particular address. During second clock cycle, as will be described below, a detected cache hit is encoded to obtain access to a corresponding entry in cache data store. Pipeline operation over two clock cycles is provided by cache pipeline registers which include, in part, TSPRA 24028 to TSPRD 24034. NC 10226 is not pipelined and does not include TSPRA 24028 to TSPRD 24034. In NC 10226, outputs of TSCA 24012 to TSCD 24024 are connected directly to inputs of TSHEA 24036 to TSHED 24042, described below.

Outputs of TSPRA 24028 to TSPRD 24034 are connected to inputs of Tag Store Hit Encoders (TSHE) 24035, respectively to Tag Store Hit Encoder A (TSHEA) 24036, Tag Store Hit Encoder B (TSHEB) 24038, Tag Store Hit Encoder C (TSHEC) 24040, and Tag Store Hit Encoder D (TSHED) 24042. TSHEA 24036 to TSHED 24042 encode, respectively, bit inputs from TSPRA 24028 to TSPRD 24034 to provide single bit outputs indicating which, if any, set of the cache's four sets includes an entry corresponding to the address input.

Single bit outputs of TSHEA 24036 to TSHED 24042 are connected to inputs of Hit Encoder (HE) 24044. HE 24044 encodes single bit inputs from TSHEA 24036 to TSHED 24042 to provide two sets of ouputs. First outputs of HE 24044 are provided to Cache Usage Store (CUS) 24046 and indicate in which of the cache's four sets, corresponding to TSA 24012 to TSD 24018, a cache hit has occurred. As described previously with reference to MC 20116, and will be described further below, CUS 24046 is a memory containing information for tracking usage of cache entries. That is, CUS 24046 contains entries indicating whether, for a particular Index, Set A, Set B, Set C or Set D of the cache's four sets has been most recently used and which has been least recently used. CUS 24046 entries regarding Sets A, B, C, and D are stored in, respectively, memories CUSA 24088, CUSB 24090, CUSC 24092, and CUSD 24094. Second output of HE 24044, as described further below, is connected to selection input of Data Store Selection Multiplexer (DSSMUX) 24048 to select an output from Data Store (DS) 24050 to be provided as output of the cache when a cache hit occurs.

Referring to DS 24050, as previously described a cache's data store contains the information, or entries, stored in that cache. For example, each entry in NC 10226's DS 24050 is a logical descriptor comprised of an AON, and Offset, and Length. A cache's data store parallels, in structure and organization, that cache's tag store and entries therein are identified and located through that cache's tag store and associated tag store comparison and decoding logic. In NC 10226, for example, for each Name having an entry in NC 10226 there will be an entry, the tag field of that name, stored in TS 24010 and a corresponding entry, a logical descriptor corresponding to that Name, in DS 24050. As described above, NC 10226 is a four-way, set associative cache so that TS 24010 and DS 24050 will each contain four sets of data. Each set was previously described as containing up to 16 entries. DS 24050 is therefore comprised of four 16 word memories. Each memory is 65 bits wide, accommodating 28 bits of AON, 32 bits of offset, and 5 bits of length. These four component data store memories of DS 24050 are indicated in Fig. 240 as Data Store A (DSA) 24052, Data Store B (DSB) 24054, Data Store C (DSC) 24056, and Data Store D (DSD) 24058. DSA 24052, DSB 24054, DSC 24056 and DSD 24058 correspond, respectively, in structure, contents, and operation to TSA 24012, TSB 24014, TSC 24016 and TSD 24018.

Data Inputs (DIs) of DSA 24052 to DSD 24058 are, in NC 10226 for example, connected from AON Bus 20230, OFFSET Bus 20228, LENGTH Bus 20226 and comprise inputs WA, WO, WL respectively of NC 10226. DSA 24052 to DSD 24058 DIs are, in NC 10226 as previously described, utilized in writing NC 10226 entries into DSA 24052 to DSD 24058. Address inputs of DSA 24052 to DSD 24058 are connected from address outputs of Address Pipeline Register (ADRPR) 24060. As will be described momentarily, except during cache flush operations, DSA 24052 to DSD 24058 address inputs are comprised of the same index fields of cache addresses as are provided as address inputs to TS 24010, but are delayed by one clock cycle and ADRPR 24060 for pipelining purposes. As described above, NC 10226 is not pipelined and does not have the one clock cycle delay. An address input to the cache will thereby result in corresponding entries, selected by index field of that address, being read from TSA 24012 to TSD 24018 and DSA 24052 to DSD 24058. The four outputs of DSA 24052 to DSD 24058 selected by a particular index field of a particular address are provided as inputs to DSSMUX 24048. DSSMUX 24048 is concurrently provided with selection control input from HE 24044. As previously described, this selection input to DSSMUX 24048 is derived from TS 24010 tag entries and indicates which of DSA 24052 to DSD 24058 entries corresponds to an address provided to the cache. In response to that selection control input, DSSMUX 24048 selects one of DS 24050's four logical descriptor outputs as the cache's output corresponding to that address. DSSMUX 24048's output is then provided, through Buffer Driver (BD) 24062 as the cache's output, for example in NC 10226 to AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226.

Petitioner Apple Inc. - Exhibit 1004, p. 5238

Referring to ADRMUX 24062, ADRMUX 24062 selects one of two sources to provide address inputs to DS 24050, that is to index to DS 24050. As described above, a first ADRMUX 24062 input is comprised of the cache's address index fields and, for example in NC 10226, is connected from the four least significant bits of NAME Bus 20224. During cache flush operations, DS 24050 address inputs are provided from Flush Counter (FLUSHCTR) 24066, which in the example is a four bit counter. During cache flush operations, FLUSHCTR 24066 generates sequential bit addresses which are used to sequentially address DSA 24052 to DSD 24058. Selection between ADRMUX 24062 first and second inputs, respectively the address index fields and from FLUSHCTR 24066, is controlled by Address Multiplexer Select (ADRMUXS) from FUCTL 20214.

Validity Store (VALS) 24068 and Dirty Store (DIRTYS) 24070 are memories operating in parallel with, and addressed in parallel with TS 24010. VALS 24068 contains entries indicating validity of corresponding TS 24010 and DS 24050 entries. That is, VALS 24068 entries indicate whether corresponding entries have been written into corresponding locations in TS 24010 and DS 24050. In the example, VALS 24068 may thereby be a 16 word by 4 bit wide memory. Each bit of a VALS 24068 word indicates validity of a corresponding location in TSA 24012 and DSA 24052, TSB 24014 and DSB 24054, TSC 24016 and DSC 24056, and TSD 24018 and DSD 24058. DIRTYS 24070 similarly indicates whether corresponding entries in corresponding locations of TS 24010 and DS 24050 have been written over, or modified. Again, DIRTYS 24070 will be a sixteen word by four bit wide memory.

Address inputs of VALS 24068 and DIRTYS 24070 are, for example in NC 10226, connected from least significant bits of NAME Bus 20224 and are thus addressed by index fields of NC 10226 addresses in parallel with TS 24010. Outputs of VALS 24068 are provided to TSCA 24020 to TSEE 24026 to inhibit outputs of TSCA 24020 through TSCE 24026 upon occurrence of an invalid concurrence between a TS 24010 entry and a NC 10226 address input. Similar outputs of DIRTYS 24070 are provided to FUCTL 20214 for use in cache flush operations to indicate which NC 10226 entries are dirty and must be written back into an MT 10350 rather than disgarded.

Outputs of VALS 24068 and DIRTYS 24070 are also connected, respectively, to inputs of Validity Pipeline Register (VALPR) 24072 and Dirty Pipeline Register (DIRTYPR) 24074. VALPR 24072 and DIRTYPR 24074 are pipeline registers similar to TSPRA 24028 to TSPRD 24034 and are provided for timing purposes as will be described momentarily. Outputs of VALPR 24072 and DIRTYPR 24074 are connected to inputs of, respectively, Validity Write Logic (VWL) 24076 and Dirty Write Logic (DWL) 24078. As described above, NC 10226 is not a pipelined cache and does not include VALPR 24072 and DIRTYPR 24074; outputs of VALS 24068 and DIRTYS 24070 are connected directly to inputs of VWL 24076 and DWL 24078. Outputs of VWL 24076 and DWL 24078 are connected, respectively, to data inputs of VALS 24068 and DIRTYS 24070. Upon occurrence of a write operation to TS 24010 and DS 24050, that is writing in or modifying a cache entry, corresponding validity and dirty word entries are read from VALS 24068 and DIRTYS 24070 by index field of the caches input address. Outputs to VALS 24068 DIRTYS 24070 are received and stored in, respectively, VALPR 24070 and DIRTYPR 24074. At start of next clock cycle, validity and dirty words in VALPR 24072 and DIRTYPR 24074 are read into, respectively, VWL 24076 and DWL 24078. VWL 24076 and DWL 24078 respectively modify those validity or dirty word entries from VALS 24068 and DIRTYS 24070 in accordance to whether the corresponding entries in TS 24010 and DS 24050 are written into or modified. These modified validity and dirty words are then written, during second clock cycle, from VWL 24076 and DWL 24078 into, respectively, VALS 24068 and DIRTYS 24070. Control inputs of VWL 24076 and DWL 24078 are provided from FUCTL 20214.

Referring finally to Least Recent Used Logic (LRUL) 24080, LRUL 24080 tracks usage of cache entries. As previously described, the generalized cache of Fig. 240 is a four way, set associative cache with, for example, up to 16 entries in each of NC 10226's sets. Entries within a particular set are identified, as described above, by indexing the cache's TS 24010 and DS 24050 may contain, concurrently, up to four individual entries identified by the same index but distiguished by having different tags. In this case, one entry would reside in Set A, comprising TSA 24012 and DSA 24052, one in Set B, comprising TSB 24014 and DSB 24054, and so on. Since the possible number of individual entries having a common tag is greater than the number of cache sets, it may be necessary to delete a particular cache entry when another entry having the same tag is to be written into the cache. In general, the cache's least recently used entry would be deleted to provide a location in TS 24010 and DS 24050 for writing in the new entry. LRUL 24080 assists in determining which cache entries are to be deleted when necessary in writing in a new entry by tracking and indicating relative usage of the cache's entries. LRUL 24080 is primarily comprised of a memory, LRU Memory (MLRU) 24081, containing a word for each cache set. As described above, NC 10226, for example, includes 16 sets of 4 frames each, so that LRUL 24080's memory may correspondingly be, for example, 16 words long. Each word indicates relative usage of the 4 frames in a set and is a 6 bit word.

Words are generated and written into LRUL 24080's MLRU 24081, through Input Register A, B, C, D (RABCD) 24083, according to a write only algorithm executed by HE 24044, as described momentarily. Each bit of each six word pertains to a pair of frames within a particular cache set and indicates which of those two frames was more recently used than the other. For example, Bit 0 will contain logic 1 if Frame A was used more recently than Frame B and a logic zero if Frame B was used more recently than Frame A. Similarly, Bit 1 pertains to Frames A and C, Bit 2 to Frames A and D, Bit 3 to Frames B and C, Bit 4 to Frames B and D, and Bit 5 to Frames C and D. Initially, all bits of a particular LRUL 24080 word are set to zero.

Petitioner Apple Inc. - Exhibit 1004, p. 5239

Assuming, for example, that the frames of a particular set are used in the sequence Frame A, Frame D, Frame B; Bits 0 to 5 of that LRUL 24080 word will initially contain all zeros. Upon a reference to Frame A, Bits 0, 1, and 2, referring respectively to Frames A and B, Frames A and C, and Frames A and D, will be written as logic 1's. Bits 3, 4, and 5, referring respectively to Frames B and C, Frames B an D, and Frames C and D, will remain logic 0. Upon reference to Frame D, Bits 0 and 1, referring respectively to Frames A and B and Frames A and C, will remain logic 1's. Bit 2, referring to Frames A and D, will be changed from logic 1 to logic 0 to indicate that Frame D has been referred to more recently than Frame A. Bit 3, referring to Frames B and C, will remain logic 0. Bits 4 and 5, referring respectively to Frames B and D and Frames C and D, will be written as logic 0, although they are already logic zeros, to indicate respectively that Frame D has been used more recently than Frame B or Frame C. Upon reference to Frame B, Bit 0, referring to Frames A and B, will be written to logic 0 to indicate that Frame B has been used more recently than Frame A. Bits 1 and 2, referring resectively to Frames A and C and Frames A and D, will remain respectively as logic 1 and logic 0. Bits three and four, referring respectively to Frames B and C and Frames B and D, will be written as logics 1's to indicate respectively that Frame B has been used more recently than Frame C or Frame D. Bit five will remain logic 0.

When it is necessary to replace a cache entry in a particular frame, the LRUL 24080 word referring to the cache set containing that frame will be read from LRUL 24080's MLRL 24081 through LRU Register (RLRU) 24085 and decoded by LRU Decode Logic (LRUD) 24087 to indicate which is least recently used frame. This decoding is executed by means of a Read Only Memory operating as a set of decoding gating.

Having described the structure and operation of a generalized cache as shown in Fig. 240, with references to NC 10226 for illustration and to point out differences between the generalized cache and NC 10226, structure and operation of ATU 10228 and PC 10234 will be described next below. ATU 10228 and PC 10234 will be described by describing the differences between ATU 10228 and PC 10234 and the generalized cache and NC 10226. ATU 10228 will be described first, followed by PC 10234.

d.d. Address Translation Unit 10228 and Protection Cache 10234

ATU 10228 is a three-way, set associative cache of 16 sets, that is contains 3 frames for each set. Structure and operation of ATU 10228 is similar to the generalized cache described above. Having 3 rather than 4 frames per set, ATU 10228 does not include a STD 24018, ATSCE 24026, ATSPRD 24034, ATSHED 24042, or ADSD 24058. As previously described ATU 10228 address inputs comprise AON and O fields of logical descriptors. AON fields are each 28 bits and O fields comprise the 18 most significant bits of logical descriptor offset fields, so that ATU 10228 address inputs are 48 bits wide. Four least significant bits of O fields are used as index. AON fields and the 14 most significant bits of O field comprise ATU 10228's tags. ATU 10228 tags are thereby each 42 bits in width. Accordingly, TSA 24012, TSB 24014, and TSC 24016 of ATU 10228's TS 24010 are each 16 words long by 42 bits wide.

DSA 24052, DSB 24054, and DSC 24056 of ATU 10228 are each 16 bits long. ATU 10228 outputs are, as previously described, physical descriptor Frame Number (FN) fields of 13 bits each. ATU 10228's DSA 24052, DSB 24054, DSC 24056 are thereby each 13 bits wide.

ATU 10228's LRUL 24080 is similar in structure and operation to that of the generalized cache. ATU 12028's LRUL 24080 words, each corresponding to an ATU 10228 set, are each 3 bits in width as 3 bits are sufficient to indicate relative usage of frames within a 3 frame set. In ATU 10228, Bit 1 of an LRUL 24080 word indicates whether Frame A was used more recently than Frame B, Bit 2 whether Frame A was used more recently than Frame C, and Bit 3 whether Frame B was used more recently than Frame C. In all other respects, other than as stated above, ATU 10228 is similar in structure and operation to the generalized cache.

Referring to PC 10234, PC 10234 is a two-way, set associative cache of 8 sets, that is has two frames per set. Having 2 rather than 4 frames, PC 10234 will not include a TSC 24016, a TSD 24018, a TSCC 24024, a TSCD 24026, a TSPRC 24032, a TSPRD 24034, a TSHEC 24040, a TSHED 24042, a DSC 24056, or a DSD 24058.

Address inputs of PC 10234 are the 28 bit AON fields of logical descriptors. The 3 least significant bits of those AON fields are utilized as indexes for addressing PC 10234's TS 24010 and DS 24050. The 25 most significant bits of those AON field address inputs are utilized as PC 10234's tags, so that PC 10234's TSA 24012 and TSB 24014 are each 8 word by 25 bit memories.

Referring to PC 10234's LRUL 24080, a single bit is sufficient to indicate which of the two frames in each of PC 10234's sets was most recently accessed. PC 10234's LRUL 24080's memory is thereby 8 words, or sets long, one bit wide.

As previously described, PC 10234 entries comprise information regarding access rights of certain active subjects to certain active objects. Each PC 10234 entry contains 35 bits of information. Three bits of this information indicate whether a particular subject was read, write, or execute rights relative to a particular object. The remaining 32 bits effectively comprise a length field indicating the volume or portion, that is the number of data bits, of that object to which those access rights pertain.

Referring again to Fig. 240, PC 10234 differs from the generalized cache and from NC 10226 and ATU 10228 in further including Extent Check Logic (EXTCHK) 24082 and Operation Check Logic (OPRCHK) 24084. PC 10234 entries include, as described above, 3 bits identifying type of access rights a particular subject has to a particular object. These 3 bits, representing a Read (R), Write (W), or Execute (E) right, are provided to a

Petitioner Apple Inc. - Exhibit 1004, p. 5240

first input of OPRCHK 24084. A second input of OPRCHK 24084 is provided from FUCTL 20214 and specifies whether JP 10114 intends to perform a Read (RI), a Write (WI), or Execute (EI), operation with respect to that object. OPRCHK 24084 compares OPRCHK 24084 access right inputs from DS 24050 to OPRCHK 24084's intended operation input from FUCTL 20214. If that subject does not possess the rights to that object which are required to perform the operation intended by JP 10114, OPRCHK 24084 generates an Operation Violation (OPRV) indicating that a protection violation has occurred.

Similarly, the 32 bits of a PC 10234 entry regarding extent rights is provided as an input (EXTENT) to EXTCHK 24082. As stated above, EXTENT field of PC 10234 entry indicates the length or number of data bits, within an obect, to which those access rights pertain. EXTENT field from PC 10234 entry is compared, by EXTCHK 24082, to offset field of the logical descriptor of the current JP 10114 request to MEM 10112 for which a current protection mechanism check is being made. If comparison of extent rights and offset field indicate that the current memory request goes beyond the object length to which the corresponding rights read from DS 24050 apply, EXTCHK 24082 generates an Extent Violation (EXTV) output. EXTV indicates that a current memory request by JP 10114 refers to a portion of an object to which the PC 10234 entry read from BS 24050 does not apply. As described previously, each read from or write to MEM 10112, even as part of a string transfer, is a 32 bit word. As such, EXTCHK 24082 will generate an EXTV output when OFFSET field of a current logical descriptor describes a segment of an object less than 32 bits from the limit defined by EXTENT field of the PC 10234 entry provided in response to that logical descriptor. EXTV and OPRV are gated together, by Protection Violation Gate (PVG) 24086 to generate Protection Violation (PROTV) output indicating that either an extent or an operation violation has occurred.

Having described the structure and operation of MEMINT 20212, and previously the structure and operation of DESP 20210, structure and operation of FUCTL 20214 will be described next below.

3. Fetch Unit Control Logic 20214 (Fig. 202)

The following descriptions will provide a detailed description of FU 10120's structure and operation. Overall operation of FU 10120 will be described first, followed by description of FU 10120's structure, and finally by a detailed description of FU 10120 operation.

As previously described, FUCTL 20214 directs operation of JP 10114 in executing procedures of user's processes. Among the functions performed by FUCTL 20214 are, first, maintenance and operation of CS 10110's Name Space, UID, and AON based addressing system, previously described; second, interpretation of SOPs of user's processes to provide corresponding sequences of microinstructions to FU 10120 and EU 10122 to control operation of JP 10114 in execution of user's processes, previously described; and, third, control of operation of CS 10110's internal mechanisms, for example CS 10110's stack mechanisms.

As will be described in further detail below, FUCTL 20214 includes prefetcher (PREF) 20260 which generates a sequence of logical addresses, each logical address comprising an AON and an offset field, for reading S-Instructions (SINs) of a user's program from MEM 10112. As previously described, each SIN may be comprised of an S-Operation (SOP) and one or more operand Names and may occupy one or more 32 bit words. SINs are read from MEM 10112 as a sequence of single 32 bit words, so that PREF 20260 need not specify a length field in a MEM 10112 read request for an SIN. SINs are read from MEM 10112 through MOD Bus 10144 and are captured and stored in Instruction Buffer (INSTB) 20262. PARSER 20264 extracts, or parses, SOPs and operand Names from INSTB 20262. PARSER 20264 provides operand Names to NC 10226 and SOPs to FUS Intrepreter Dispatch Table (FUSDT) 11010 and to EU Dispatch Table (EUSDT) 20266 through Op-Code Register (OPCODEREG) 20268. Operation of INSTB 20262 and PARSER 20264 is controlled by Current program Counter (CPC) 20270, Initial Program Counter (IPC) 20272, and Executed program Counter (EPC) 20274.

As previously described, FUSDT 11010 provides, for each SOP received from OPCODEREG 20268, a corresponding S-Interpreter Dispatch (SD) Pointer, or address, to FUSITT 11012 to select a corresponding sequence of microinstructions to direct operation of JP 10114, in particular FU 10120. As previously described, FUSITT 11012 also contains sequences of microinstructions for controlling and directing operation of CS 10110's internal mechanisms, for example those mechanisms such as RCWS 10358 which are involved in swapping of processes. EUSDT 20266 performs an analogous function with respect to EU 10122 and provides SD Pointers to EU S-Interpreter Tables (EUSITTs) residing in EU 10122.

Micro-Program Counter (mPC) 20276 provides sequential addresses to FUSITT 11012 to select individual microinstructions of sequences of microinstructions. Branch and Case Logic (BRCASE) 20278 provides addresses to FUSITT 11012 to select microinstructions sequences for microinstructions branches and and cases. Repeat Counter (REPCTR) 20280 and Page Number Register (PNREG) 20282 provide addresses to FUSITT 11012 during FUSITT 11012 load operations.

As previously described, FUSITT 11012 is a writable microinstruction control store which is loaded with selected S-Interpreters (SINTs) from MEM 10112.

FUSITT 11012 addresses are also provided by Event Logic (EVENT) 20284 and by JAM input from NC 10226. As will be described further below, EVENT 20284 is part of FUCTL 20214's circuitry primarily concerned with operation of CS 10110's internal mechanisms. Input JAM from NC 10226 initiates certain FUCTL 20214 control functions for CS 10110's Name Space addressing mechanisms, and in particular NC 10226. Selection between the above discussed address inputs to FUSITT 11012 is controlled by S-

Petitioner Apple Inc. - Exhibit 1004, p. 5241

Interpreter Table Next Address Generator Logic (SITTNAG) 20286.

Other portions of FUCTL 20214's circuitry are concerned with operation of CS 10110's internal mechanisms. For example, FUCTL 20214 includes Return Control Word Stack (RCWS) 10358, previously described with reference to CS 10110's Stack Mechanisms. Register Address Generator (RAG) 20288 provides pointers for addressing of GRF 10354 and RCWS 10358 and includes Micro-Stack Pointer Registers (MISPR) 10356.

As previously described, MISPR 10356 mechanism provides pointers for addressing Micro-Stack (MIS) 10368. As will be described further below, actual MIS 10368 Pointers pointing to current, previous, and bottom frames of MIS 10368 reside in Micro-Control Word Register 1 (MCW1) 20290. MCW1 20290 and Micro-Control Word Zero Register (MCWO) 20292 together contain certain information indicating the current execution environment of a microinstruction sequence currently being executed by FU 10120. This execution information is used in aide of execution of these microinstruction sequences. State Registers (STATE) 20294 capture and store certain information regarding state of operation of FU 10120. As described further below, this information, referred to as state vectors, is used to enable and direct operation of FU 10120.

Timers (TIMERS) 20296 monitor elapsed time since occurrence of the events requiring servicing by FU 10120. If waiting time for these events exceeds certain limits, TIMERS 20296 indicate that these limits have been exceeded so that service of those events may be initiated.

Finally, Fetch Unit to E Unit Interface Logic (FUEUINT) 20298 comprises the FU 10120 portion of the interface between FU 10120 an EU 10122. FUEUINT 20298 is primary path through which operation of FU 10120 and EU 10122 is coordinated.

Having described overall operation of FU 10120, structure of FU 10120 will be described next below with aide of Fig. 202, description of FU 10120's structure will be followed by a detailed description of FU 10120 wherein further, more detailed, diagrams of certain portions of FU 10120 will be introduced as required to enhance clarity of presentation.

a.a. Fetch Unit Control Logic 20214 Overall Structure

Referring again to Fig. 202, as previously described Fig. 202 includes a partial block diagram of FUCTL 20214. Following the same sequence of description as above, PREF 20260 has a 28 bit bi-directional port connected to AON Bus 20230 and 32 bit bi-directional port directed from OFFSET Bus 20228. A control input of PREF 20260 is connected from control output of INSTB 20262.

INSTB 20262 32 bit data input (DI) is connected from MOD Bus 10144. INSTB 20262's 16 bit output (DO) is connected to 16 bit bi-directional input of OPCODEREG 20268 and to 16 bit NAME Bus 20224. OPCODEREG 20268's input comprises 8 bits of SINT and 3 bits of dialect selection. As previously described, NAME Bus 20224 is connected to 16 bit bi-directional port of Name Trap (NT) 20254, to address input ADR of NC 10226, and to inputs and outputs of OFFP 20228. Control inputs of INSTB 20262 and PARSER 20264 are connected from a control output of CPC 20270.

Thirty-two bit input of CPC 20270 is connected from JPD Bus 10142 and CPC 20270's 32 bit output is connected to 32 bit input of IPC 20272. Thirty-two bit output of IPC 20272 is connected to 32 bit input of EPC 20274 and to JPD Bus 10142. EPC 20274's 32 bit output is similarly connected to JPD Bus 10142.

Eleven bit outputs of OPCODEREG 20268 are connected to 11 bit address inputs of FUSDT 11010 and EUSDT 20266. These 11 bit address inputs to FUSDT 11010 and EUSDT 20266 each comprise 3 bits of dialect selection code and 8 bits of SINT code. Twelve bit SDT outputs of EUSDT 20266 is connected to inputs of Microinstruction Control Store in EU 10122, as will be described in a following description of EU 10122. FUSDT 11010 has, as described further below, two outputs connected to address (ADR) Bus 20298. First output of FUSDT 11010 are six bit SDT pointers, or addresses, corresponding to generic SINTs as will be described further below. Second output of FUSDT 11010 are 16 bit SDT pointers, or addresses, for algorithm microinstruction sequences, again as will be described further below.

Referring to RCWS 10358, RCWS 10358 has a first bidirectional port connected from JPD Bus 10142. Second, third, and fourth bi-directional ports of RCWS 10358 are connected from, respectively, a bi-directional port of MCW1 20290, a first bi-directional port EVENT 20284, and a bi-directional port of mPC 20276. An output of RCWS 10358 is connected to ADR Bus 20298.

An input of mPC 20276 is connected from ADR Bus 20298 and first and second outputs of mPC 20276 are connected to, respectively, an input of BRCASE 20278 and to ADR Bus 20298. An output of BRCASE 20278 is connected to ADR Bus 20298.

As described above, a first bi-directional port of EVENT 20284 is connected to RCWS 10358. A second bidirectional port of EVENT 20284 is connected from MCWO 20292. An output of EVENT 20284 is connected to ADR Bus 20298.

Inputs of RPCTR 20280 and PNREG 20282 are connected from JPD Bus 10142. Outputs of RPCTR 20280 and PNREG 20282 are connected to ADR Bus 20298.

ADR Bus 20298, and an input from a first output of FUSITT 11012, are connected to inputs of SITTNAG 20286.

Output of SITTNAG 20286 is connected, through Control Store Address (CSADR) Bus 20299, to address input of FUSITT 11012. Data input of FUSITT 11012 is connected from JPD Bus 10142. Control outputs of FUSITT 11012 are connected to almost all elements of JP 10114 and thus, for clarity of presentation, are not

Petitioner Apple Inc. - Exhibit 1004, p. 5242

shown in detail by drawn physical connections but are described in following descriptions.

As described above, MCW0 20292 and MCW1 20290 have bi-directional ports connected to, respectively, bidirectional ports of EVENT 20284 and to a second bidirectional port of RCWS 10358. Outputs of MCW0 20292 and MCW1 20290 are connected to JPD Bus 10142. Other inputs of MCW0 20292 and MCW1 20290, as will be described further below, are connected from several other elements of JP 10114 and, for clarity of presentation, are not shown herein in detail but are described in the following text. STATE 20294 similarly has a large number of inputs and outputs connected from and to other elements of JP 10114, and in particular FU 10120. Inputs and outputs of STATE 20294 are not indicated here for clarity of presentation and will be described in detail below.

RAG 20288 has an input connected from JPD Bus 10142 and other inputs connected, for example, from MCW1 20290. RAG 20288, including MISPR 10356, provides outputs, for example, as address inputs to RCWS 10358 and GRF 10354. Again, for clarity of presentation, inputs and outputs of RAG 20288 are not shown in detail in Fig. 202 but will be described in detail further below.

TIMERS 20296 receive inputs from EVENT 20284 and FUSITT 11012 and provide outputs to EVENT 20284. For clarity of presentation, these indications are not shown in detail in Fig. 202 but will be described further below.

FUINT 20298 receives control inputs from FUSITT 11012 and EU 10122. FUINT 20298 provides outputs to EU 10122 and to other elements of FUCTL 20214. For clarity of presentation, connections to and from FUINT 20298 are not shown in detail in Fig. 202 but will be described in further detail below.

Having described the overall operation, and structure, of FUCTL 20214, operation of FUCTL 20214 will be described next below. During the following descriptions further diagrams of certain portions of FUCTL 20214 will be introduced as required to disclose structure and operation of FUCTL 20214 to one of ordinary skill in the art. FUCTL 20214's operation with regard to fetching and interpretation of SINs, that is SOPs and operand Names, will be described first, followed by description of FUCTL 20214's operation with regard to CS 10110's internal mechanisms.

b.b. Fetch Unit Control Logic 20214 Operations

Referring first to those elements of FUCTL 20214 directly concerned with control of JP 10114 in response to SOPs and Name syllables, those elements include: (1) PREF 20260; (2) INSTB 20262; (3) PARSER 20264; (4) CPC 20270, IPC 20272, and EPC 20274; (5) OPCODEREG 20268; (6) FUSDT 11010 and EUSDT 20266; (7) mPC 20276; (8) BRCASE 20278; (9) REPCTR 20280 and PNREG 20282; (10) a part of RCWS 10358; (11) SITTNAG 20286; (12) FUSITT 11012; and, (13) NT 20254. These FUCTL 20214 elements will be described below in the order named.

a.a.a. Prefetcher 20260, Instruction Buffer 20262, Parser 20264, Operation Code Register 20268, CPC 20270, IPC 20272, and EPC 20274 (Fig. 241)

As described above, PREF 20260 generates a series of addresses to MEM 10112 to read SINs of user's programs from MEM 10112 to FUCTL 20214, and in particular to INSTB 20262. Each PREF 20260 read request transfers one 32 bit word from MEM 10112. Each SIN may be comprised of an SOP and one or more Name syllables. Each SOP may comprise, for example, 8 bits of information while each Name syllable may comprise, for example, 8, 12, or 16 bits of data. In general, and as will be described in further detail in a following description of STATE 20294, PREF 20260 obtains access to MEM 10112 on alternate 110 nanosecond system clock cycles. PREF 20260's access to MEM 10112 is conditional upon INSTB 20262 indicating that INSTB 20262 is ready to receive an SIN read from MEM 10112. In particular, INSTB 20262 generates control output Quiry Prefetch (QPF) to PREF 20260 to enable PREF 20260 to submit a request to MEM 10112 when, as described further below, INSTB 20262 is ready to receive an SIN read from MEM 10112.

PREF 20260 is a counter register comprised, for example of SN74S163s.

Bi-directional inputs and outputs of PREF 20260 are connected to AON Bus 20230 and OFFSET Bus 20228. As PREF 20260 reads only single 32 bit words, PREF 20260 is not required to specify a LENGTH field as part of an SIN read request, that is an AON and an OFFSET field are sufficient to define a single 32 bit word. At start of read of a sequence of SINs from MEM 10112, address (AON and OFFSET fields) of first 32 bit word of that SIN sequence are provided to MEM 10112 by DESP 20210 and concurrently loaded, from AON Bus 20230 and OFFSET Bus 20228, into PREF 20260. Thereafter, as each successive thirty-two bit word of the SIN's sequence is read from MEM 10112, the address residing in PREF 20260 is incremented to specify successive 32 bit words of that SIN's sequence. The successive single word addresses are, for all words after first word of a sequence, provided to MEM 10112 from PREF 20260.

As described above, INSTB 20262 receives SINs from MEM 10112 through MOD Bus 10144 and, with PARSER 20264 and operating under control of CPC 20270, provides Name syllables to NAME Bus 20224 and SINs to OPCODEREG 20268. INSTB 20262 is provided, together with PREF 20260 to increase execution speed of SINS.

Referring to Fig. 241, a more detailed block diagram of INSTB 20262, PARSER 20264, CPC 20270, IPC 20272, EPC 20274 as shown. INSTB 20262 is shown as comprising two 32 bit registers having parallel 32 bit inputs from MOD Bus 10144. INSTB 20262 also receives two Write Clock (WC) inputs, one for each 32 bit register of INSTB 20262, from Instruction Buffer Write Control (INSTBWC) 24110. INSTB 20262's outputs

Petitioner Apple Inc. - Exhibit 1004, p. 5243

are structured as eight, eight bit Basic Syllables (BSs), indicated as BS0 to BS7. BS0, BS2, BS4, and BS6 are ORed to comprise eight bit Basic Syllable, Even (BSE) of INSTB 20262 while BS0, BS3, BS5, and BS7 are similarly ORed to comprise Basic Syllable, Odd (BSO) of INSTB 20262. BSO and BSE are provided as inputs of PARSER 20264.

PARSER 20264 receives a first control input from Current Syllable Size Register (CSSR) 24112, associated with CPC 20270. A second control input of PARSER 20264 is provided from Instruction Buffer Syllable Decode Register (IBSDECR) 24114, also associated with CPC 20270. PARSER 20264 provides an eight bit output to NAME Bus 20224 and to input of OPCODEREG 20268.

Referring to INSTBWC 24110, INSTBWC 24110 provides, as described further below, control signals pertaining to writing of SINs into INSTB 20262 from MOD Bus 10144. INSTBWC 24110 also provides control signals pertaining to operation of PREF 20260. In addition to WC outputs to INSTB 20262, INSTBWC 24110 provides control output QPF to PREF 20260, control output Instruction Buffer Hung (IBHUNG) to EVENT 20284, and control signal Instruction Buffer Wait (IBWAIT) to STATE 20294. INSTBWC 24110 also receives a control input BRANCH from BRCASE 20278 and an error input from TIMERS 20296.

Referring to CPC 20270, IPC 20272, and EPC 20274, IPC 20272 and EPC 20274 are represented in Fig. 241 as in Fig. 202. Further FUCTL 20214 circuitry is shown as associated with CPC 20270. CPC 20270 is a twenty-nine bit register receiving bits one to twety-five (CPC(1—25)) from bits one to twenty-five of JPD Bus 10142. CPC 20270 Bit 0 (CPC0) is provided from CPC0 CPC0 Select (CPC0S) 24116. Inputs of CPC0S 24116 are Bit 1 output from CPC 20270 (CPC1) and Bit 0 from JPD Bus 10142. Bits twentysix, twenty-seven, and twenty-eight of CPC 20270 (CPC(2628)) are provided from CPC Multiplexer (CPCMUX) 24118. CPCMUX 24118 also provides an input to IBSDECR 24114. Inputs of CPCMUX 24118 are bits twenty-five, twenty-six, and twenty-eight from JPD Bus 10142 and a three bit output of CPC Arithmetic and Logic Unit (CPCALU) 24120. A first input of CPCALU 24120 is connected from output bits 26, 27, and 28 of CPC 20270. Second input of CPCALU 24120 is connected from CSSR 24112. CSSR 24112's input is connected from JPD Bus 10142.

As described above, INSTB 20262 is implemented as a sixty-four bit wide register. INSTB 20262 is organized as two thirty-two bit words, referred to as Instruction Buffer Word 0 (IB0) and Instruction Buffer Word 1 (IB1), and operates as a two word, first-in-first-out buffer memory. PREF 20260 loads one of IB0 or IB1 on each memory reference by PREF 20260. Only PREF 20260 may load INSTB 20262, and INSTB 20262 may be loaded only from MOD Bus 10144. Separate clocks, respectively Instruction Buffer Write Clock 0 (IBWC0) and Instruction Buffer Write Clock 1 (IBWC1), are provided from INSTBWC 24110 to load, respectively, IBW0 and IBW1 into INSTB 20262. IBWC0 and IBWC1 are each a gated 110 nano-second clock. An IBW0 or an IBW1 is written into INSTB 20262 when, respectively, IBWC0 or IBWC1 is enabled by INSTBWC 24110. IBWC0 and IBWC1 will be enabled only when MEM 10112 indicates that data for INSTB 20262 is availabe by asserting interface control signal DAVI as previously discussed.

INSTBWC 24110 is primarily concerned with control of FU 10120 with respect to writing of SINs into INSTB 20262. As described above, INSTBWC 24110 provides IBWC0 and IBWC1 to INSTB 20262. IBWC0 and IBWC1 are enabled by INSTBWC 24110's input DAVI from MEM 10112. Selection between IBWC0 and IBWC1 is controlled by INSTBWC 24110's input from CPC 20270. In particular, and as will be described further below, Bit 26 (CPC 26) of CPC 20270's twenty-nine bit word indicates whether IBW0 or IBW1 is written into INSTB 20262.

In addition to controlling writing of IBW0 and IBW1 into INSTB 20262, INSTBWC 24110 provides control signals to elements of FU 10120 to control reading of SINs from MEM 10112 to INSTB 20262. In this regard, INSTBWC 24110 detects certain conditions regarding status of SIN words in INSTB 20262 and provides corresponding control signals, described momentarily, to other elements of FU 10120 so that INSTB 20262 would generally always contain at least one valid SOP or Name syllable. First, if INSTB 20262 is not full, that is either IBW0 or IBW1 or both is invalid, for example because IBW0 has been read from INSTB 20262 and executed, INSTBWC 24110 detects this condition and provides control signal QPF to PREF 20262 to initiate a read from MEM 10112. INSTBWC 24110 currently enables either IBW0 or IBW1 portion of INSTB 20262 to receive the word read from MEM 10112 in response to PREF 20260's request. As stated above, this operation will be initiated when INSTBWC 24110 detects and indicates, by generating a validity flag, that either IBW0 or IBW1 is invalid. In this case, IBW0 or IBW1 will be indicated as invalid when read from INSTB 20262 by PARSER 20264. As will be described further below, INSTBWC 24110 validity flags for IBW0 and IBW1 are generated by INSTBWC 24110 control inputs comprising Bits 26 to 28 (CPC 26—28) from CPC 20270 and by current syllable size or value, flag (K) input from CSSR 24112. Secondly, INSTBWC 24110 will detect when INSTB 20262 is empty, that is when both IBW0 and IBW1 are invalid, as just described, or when only a half of a sixteen bit Name syllable is present in INSTB 20262. In response to either condition, INSTBWC 24110 will generate control signal IBWAIT to STATE 20294. As will be described further below, IBWAIT will result in suspension of execution of microinstructions referencing INSTB 20262. PREF 20260 requests to MEM 10112 will already have been initiated, as described above unless certain other corditions, described momentarily, occur. Thirdly, INSTBWC 24110 will detect when INSTB 20262 is empty and PREF 20262 is hung, that is unable to submit requests to MEM 10112, and a current microinstruction is attempting to parse a syllable from INSTB 20262. In this case, INSTBWC 24110 will generate control signal Instruction Buffer Hung (IBHUNG) to EVENT 20284. As will be described further below, IBHUNG will result in initiation of a microinstruction sequence to restore flow of words to INSTB 20262. Fourthly, INSTBWC 24110 will detect, through microinstruction control signals provided from FUSITT 11012, when a branch in

Petitioner Apple Inc. - Exhibit 1004, p. 5244

a microinstruction sequence provided by FUSITT 11012 in response to an SOP occurs. In this case, both IBW0 and IBW1 will be flaged as invalid. INSTBWC 24110 will then ignore SIN words being read from MEM 10112 in response to a previously submitted PREF 20260 request, but not yet received at the time the branch occurs. This prevents INSTB 20260 from receiving invalid SIN words; PREF 20260 and INSTB 20262 will then proceed to request and receive valid SIN words of the branch.

As described above, PARSER 20264, operating under control of CPC 20270 and CPC 20270 associated circuitry, reads Name syllables and SOPs from INSTB 20262 to, respectively, NAME Bus 20224 and OPCODEREG 20268. PARSER 20264 operates as a multiplexer with associated control logic.

As previously described, INSTB 20262 is internally structured as eight, eight bit words, BS0 to BS7. IBW0 comprises BS0 to B3 while IBW1 comprises BS4 to BS7. Each SOP is comprised of eight bits of data and thus comprises one Basic Syllable while each Name syllable comprises 8, 12, or 16 bits of data and thus comprises either one or two Basic Syllables. Name syllable size, as previously stated, is indicated by Current Syllable Size Value K stored in CSSR 24112.

BS0 and BS4 are loaded into INSTB 20262 from MOD Bus 10144 bits zero to seven while BS2 and BS6 are loaded from MOD Bus 10144 bits sixteen to twenty-three. BS1 and BS5 are loaded from MOD Bus 10144 bits eight to fifteen while BS3 and BS7 are loaded from MOD Bus 10144 bits twenty-four to thirty-one. Odd numbered Basic Syllable outputs BS1, BS3, BS5, and BS7 are ORed to comprise eight bit Basic Syllable, Odd output BS0 of INSTB 20262. Even numbered Basic Syllable outputs BSo, BS2, BS4 and BS6 of INSTB 20262 are similarly ORed to comprise eight bit Basic Syllable, Even output BSE. At any time, one odd numbered Basic Syllable output and one even numbered Basic Syllable output of INSTB 20262 are selected as inputs to PARSER 20264 by Instruction Buffer Read Enable (IBORE) enable and selection signals provided to INSTB 20262 by IBSDECR 24114. IBSDECR 24114 includes decoding circuitry. Input to IBSDECR 24114's decoding logic is comprised of three bits (RCPC(26—28)) provided from CPCMUX 24118. As indicated in Fig. 241, CPC (26—28) may be provided from JPD Bus 10142 bits 25 to 28 or from output of CPCALU 24120. One input CPCALU 24120 is CPC (26—28) from CPC 20270. Operation of CPC 20270 and CPC 20270's associated circuitry will be described further below. RCPC (26—28) is decoded by IBSDECR 24114 to generate IBORE (0—7) to INSTB 20262. RCPC 26 and RCPC 27 are decoded to select one of the four odd numbered Basic Syllable outputs (that is BS1, BS3, BS5 or BS7) of INSTB 20262 as the odd numbered basic syllable input to PARSER 20264. RCPC 28 selects either the preceding or the following even numbered Basic Syllable output of INSTB 20262 as the even numbered Basic Syllable input to PARSER 20264. The eight decoded bits of IBORE (0—7) generated by IBDECR 24114 decoding logic are loaded into IBSDECR 24114 eight bit register and subsequently provided to INSTB 20262 as IBORE (0—7).

PARSER 20264 selects BS0, or BSE, or both BSO and BSE, as PARSER 20264s output to NAME Bus 20224 or to OPCODEREG 20268. In the case of an SOP or an eight bit Name syllable, either BSO or BSE will be selected as PARSER 20264's output. In the case of a twelve or sixteen bit Name syllable, both BS0 and BSE may be selected as PARSER 20264's output. PARSER 20264 operation is controlled by microinstruction control outputs from FUSITT 11012.

Program counters IPC 20272, EPC 20274, and CPC 20270 are associated with control of fetching and parsing of SINs. In general, IPC 20272, EPC 20274, and CPC 20270 operate under microinstruction control from FUSITT 11012.

CPC 20270 is Current Program Counter and contains 28 bits pointing to the current syllable in INSTB 20272. Bits 29 to 31 of CPC 20270 are not provided, so the bits 29 to 31 of CPC 20270's output are zero, which guarantees byte boundaries for SOPs. Contents of CPC 20270 are thereby also a pointer which is a byte align offset into a current procedure object. Initial Program Counter (IPC) 20272 is a buffer register connected from output of CPC 20270 and provided for timing overlap. IPC 20272 may be loaded only from CPC 20270 which, as previously described, is 29 bits wide, that does not contain bits 29, 30, and 31 which are forced to zero in IPC 20272. IPC 20272 may be read onto JPD Bus 10142 as a start value in an unconditional branch.

EPC 20274 is a thirty-two bit register usually containing a pointer to the current SOP being executed. Upon occurrence of an SOP branch, the pointer in EPC 20274 will point to the SOP from which the branch was executed. The pointer residing in EPC 20274 is an offset into a current procedure object. EPC 20274 may be loaded only from IPC 20272, and may be read onto JPD Bus 10142.

Referring again to CPC 20270, as described above CPC 20270 is a current syllable counter. CPC 20270 contains a pointer to the next SOP syllable, or Base Syllable, to be parsed by PARSER 20264. As SOPs are always on byte boundaries, CPC 20270 pointer is 29 bits wide, CPC (0—28). The three low order bits of CPC 20270's pointer, that is CPC (29—31), do not physically exist and are assumed to be always zero. CPC 20270's pointer to next instruction syllable to be parsed thereby always points to byte boundaries.

CPC 20270 bits 26 to 28, CPC (26—28), indicate, as described above, a particular Base Syllable in INSTB 20262. Bits 0—25 (CPC(0—25)) of CPC 20270 indicate 32 bit words, read into INSTB 20262 as IBW0 and IBW1, of a sequence of SINs. CPC 20270 pointer is updated each time a parse operation reading a Base Syllable from INSTB 20262 is executed. As previously described, these parsing operations are performed under microinstruction control from FUSITT 11012.

Conceptually, CPC 20270 is organized as a twenty-six bit counter, containing CPC (0—25), with a three bit register appended on the low order side, as CPC (26—28). This organization is used because CPC (26—28) counts INSTB 20262 Base Syllables parsed and must be incremented dependant upon current

Petitioner Apple Inc. - Exhibit 1004, p. 5245

Name Syllable Size K stored CSSR 24112. CPC (0—25), however, counts successive thirty-two bit words of a sequence of SINs and may thereby be implemented as a binary counter. As shown in Fig. 241, CPC (26—28) is loaded from output of CPCMUX 24118. A first input of CPCMUX 24118 is connected from bits 29 to 31 of JPD Bus 10142. This input to CPC (26—28) from JPD Bus 10142 is provided to allow CPC 20270 to be loaded from JPD Bus 10142, for example when loading CPC 20270 with an initial pointer value. Second input of CPCMUX 24118 is from output of CPCALU 24120 and is the path by which CPC (26—28) is incremented as successive Base Syllables are parsed from INSTB 20262. A first input of CPCALU 24120 is CPC (26—28) from CPC 20270. Second input of CPALU 24120 is a dual input from CSSR 24112. First input from CSSR 24112 is logic 1 in the least significant bit position, that is in position corresponding to CPC (28). This input is used when single Base Syllables are parsed from INSTB 20262, for example in an eight bit SOP or an eight bit Name syllable. CSSR 24112's first input to CPCALU 24120 increments CPC (0—32) by eight, that is one to CPC (26—28), each time a single Base Syllable is parsed from INSTB 20262. Second input to CPCALU 24120 from CSSR 24112 is K, that is current Name Syllable size. As previously described, K may be eight, twelve, or sixteen. CPC (26—28) is thereby incremented by one when K equals eight and is incremented by two when K equals twelve or sixteen. As shown in Fig. 241, K is loaded into CSSR 24112 from JPD Bus 10142.

CPC (0—25), as described above, operates as a twenty-six bit counter which is incremented each time CPC (26—28) overflows. CPC (0—25) is incremented by carry output of CPCALU 24120. In actual implementation, CPC 20270 is organized to reduce the number of integrated circuits required. CPC (1—25) is constructed as a counter and inputs of CPC (1—25) counter are connected from bits 1 to 24 of JPD Bus 10142 to allow loading of an initial value of CPC 20270 pointer. CPC (0) and CPC (26—28) are implemented as a four bit register. Operation of CPC (26—28) portions of this register have been described above. Input of CPC (0) portion of this register is connected from output of CPCOS 24116. CPCOS 24116 is a multiplexer having a first input connected from bit 0 of JPD Bus 10142. This input from JPD Bus 10142 is used, for example, when loading CPC 20272 with an initial pointer value. Second input of CPCOS 24116 is overflow output of CPC (1—25) counter and allows CPC (0) portion of the four bit register and CPC (1—25) counter to operate as a twenty-six bit counter.

Finally, as shown in Fig 241, output of CPC 20270 may be loaded into IPC 20272. An initial CPC 20270 pointer value may therefore be written into CPC 20270 from JPD Bus 10142 and subsequently copied into IPC 20272.

Referring again to PARSER 20264, as described above PARSER 20264 reads, or parses, basic syllables from INSTB 20262 to NAME Bus 20224. Input of PARSER 20264 is a sixteen bit word comprised of an eight bit odd numbered Base Syllable, BS0, and an eight bit even numbered Base Syllable, BSE. Depending upon whether PARSER 20264 is parsing an eight bit SOP, an eight bit Name syllable, a twelve bit Name syllable, or sixteen bit Name syllable, PARSER 20264 may select BS0, BSE, or both BS0 and BSE, as output onto NAME Bus 20224.

If PARSER 20264 is parsing Name syllables and K is not equal to eight, that is equal to twelve or sixteen, PARSER 20264 transfers both BSO and BSE onto NAME Bus 20224 and determines which of BSO or BSE is most significant. The decision as to whether BSO or BSE is most significant is determined by CPC (28). If CPC (28) indicates BS0 is most significant, BSO is transferred onto NAME Bus 20224 bits 0 to 7 (NAME(0—7)) and BSE onto NAME Bus 20224 bits eight to fifteen (NAME(8—15)). If CPC (28) indicates BSE is most significant, BSE is transferred onto NAME (0—7) and BSO onto NAME (8—15). This operation insures that Name syllables are parsed onto NAME Bus 20224 in the order in which occur in the SIN stream.

If PARSER 20264 is parsing Name syllables of Syllable Size K = 8, PARSER 20264 will select either BSO or BSE, as indicated by CPC (28), as output to NAME (0—7). PARSER 20264 will place 0's on NAME (8—15).

If PARSER 20264 is parsing SOPs of eight bits, PARSER 20264 will select BSO or BSE as output to NAME (0—7) as selected by CPC (28). PARSER 20264 will place 0's onto NAME (8—15). Concurrently, PARSER 20264 will generate OPREGE to OPCODEREG 20268 to enable transfer of NAME (0—7) into OPCODEREG 20268. OPCODEREG 20268 is not loaded when PARSER 20264 is parsing Name syllables. The microinstruction input from FUSITT 11012 which controls PARSER 20264 operation also determines whether PARSER 20264 is parsing an SOP or a Name syllable and controls generation of OPREGE.

Operation of NC 10226, which receives Name syllables as address inputs from NAME Bus 20224, has been discussed previously with reference to MEMINT 20212. Name Trap (NT) 20254 is connected from NAME Bus 20224 to receive and capture Name syllables parsed onto NAME Bus 20224 by PARSER 20264. Operation of NT 20254 has been also previously discussed with reference to MEMINT.

b.b.b. Fetch Unit Dispatch Table 11010, Execute Unit Dispatch Table 20266 and Operation Code Register 20268 (Fig. 242)

As previously described, CS 10110 is a multiple language machine. Each program written in a high level user language is compiled into a corresponding S-Language program containing S-Language Instructions referred to as SOPs. CS 10110 provides a set or dialect, of microcode instructions, referred to as S-Interpreters (SINTs) for each S-Language. SINTs interpret SOPs to provide corresponding sequences of microinstructions for detailed control of CS 10110 operations. CS 10110's SINTs for FU 10120 and EU 10122 operations are stored, respectively, in FUSITT 11012 and in a corresponding control store memory in EU 10122, described in a following description of EU 10122. Each SINT comprises one or more sequences

Petitioner Apple Inc. - Exhibit 1004, p. 5246

of microinstructions, each sequence of microinstructions corresponding to a particular SOP in a particular S-Language dialect. Fetch Unit S-Interpreter Dispatch Table (FUSDT) 11010 and Execute Unit S-Interpreter Dispatch Table (EUSDT) 20266 contain an S-Interpreter Dispatcher (SD) for each S-Language dialect. Each SD is comprised of a set of SD Pointers (SDPs) wherein each SDP in a particular SD corresponds to a particular SOP of that SD dialect. Each SDP is an address pointing to a location, in FUSITT 11012 or EUSITT, of the start of the corresponding sequence of microinstructions for interpreting the SOP corresponding to that SDP. As will be described further below, SOPs received and stored in OPCODEREG 20268 are used to generate addresses into FUSDT 11010 and EUSDT 20266 to select corresponding SDPs. Those SDPs are then provided to FUSITT 11012 through ADR 20202, or to EUSITT through EUDIS Bus 20206, to select corresponding sequences of microinstructions from FUSITT 11012 and EUSITT.

Referring to Fig. 242, a more detailed block diagram of OPCODEREG 20268, FUSDT 11010, and EUSDT 20266 is shown. As shown therein, OPCODEREG 20268 is comprised of OP-Code Latch (LOPCODE) 24210, Dialect Register (RDIAL) 24212, Load Address Register (LADDR) 24214, and Fetch Unit Dispatch Encoder (FUDISENC) 24216. Data inputs of LOPCODE 24010 are connected from NAME Bus 20224 to receive SOPs parsed from INSTB 20262. Load inputs of RDIAL 24212 are connected from Bits 28 to 31 of JPD Bus 10142. Outputs of LOPCODE 24210, RDIAL 24212 and LADDR 24214 are connected to inputs of FUDISENC 24216. Outputs of FUDISENC 24216 are connected to address inputs of FUSDT 11010 and EUSDT 20266.

FUSDT 11010 is comprised of Fetch Unit Dispatch File (FUDISF) 24218 and Algorithm File (AF) 24220. Address inputs of FUDISF 24218 and AF 24220 are connected, as previously described, from address outputs of FUDISENC 24216. Data load inputs of FUDISF 24218 and AF 24220 are connected from, respectively, Bits 10 to 15 and Bits 16 to 31 of JPD Bus 10142. SDP outputs of FUDISF 24218 and AF 24220 are connected to ADR Buses 20202.

EUSDT 20266 is comprised of Execute Unit Dispatch File (EUDISF) 24222 and Execute Unit Dispatch Selector (EUDISS) 24224. Address inputs of EUDISF 24222 are, as described above, connected from outputs of FUDISENC 24216. Data load inputs of EUDISF 24222 are connected from Bits 20 to 31 of JPD Bus 10142. Inputs of EUDISS 24224 are connected from SDP output of EUDISF 24222, from Bits 20 to 31 of JPD Bus 10142, and from Microcode Literal (mLIT) output of FUSITT 11012. SDP outputs of EUDISS 24224 are connected to EUDIS Bus 20206.

As previously described, OPCODEREG 20268 provides addresses, generated from SOPs loaded into OPCODEREG 20268, to FUSDT 11010 and EUSDT 20266 to select SDPs to be provided as address inputs to FUSITT 11012 and EUSITT. LOPCODE 24210 receives and stores eight bit SOPs parsed from INSTB 20262 as described above. OPCODEREG 20268 also provides addresses to FUSDT 11010 and EUSDT 20266 to load FUSDT 11010 and EUSDT 20266 with SDs for S-Language dialects currently being utilized by CS 10110. LOPCODE 24210 and RDIAL 24212, as described below, provide addresses to FUSDT 11010 and EUSDT 20266 when translating SOPs to SDPs and ADDR 24214 provides addresses when FUSDT 11010 and EUSDT 20266 are being loaded with SDs.

Referring first to LADDR 24214, LADDR 24214 has an eight bit counter. Addresses are provided to FUSDT 11010 and EUSDT 20266 from LADDR 24214 only when FUSDT 11010 and EUSDT 20266 are being loaded with SDs, that is groups of SDPs for S-Language dialects currently being utilized by CS 10110. During this operation, output of LADDR 24214 is enabled to FUSDT 11010 and EUSDT 20266 by microcode control signals (not shown for clarity of presentation) from FUSITT 11012. Selection between FUDISF 24218, AF 24220, and EUDISF 24222 to receive addresses is similarly provided by microinstruction enable signals (also not shown for clarity of presentation) provided from FUSITT 11012. These FUSDT 11010 and EUSDT 20266 address enable inputs may select, at any time, any or all of FUDISF 24218, AF 24220, or EUDSF 24222 to receive address inputs. SDPs to be loaded into FUDISF 24218, AF 24220, and EUDISF 24222 are provided, respectively, from Bits 10 to 15 (JPD(10—15)), Bits 16 to 31 (JPD(16—31)), and Bits 20 to 31 (JPD(20—31)) of JPD Bus 10142. Address contents of LADDR 24214 are successively incremented by one as successive SDPs are loaded into FUSDT 11010 and EUSDT 20266. Incrementing of LADDR 24214 is, again, controlled by microinstruction control inputs from FUSITT 11012.

Address inputs to FUSDT 11010 and EUSDT 20266 during interpretation of SOPs are provided from LOPCODE 24210 and RDIAL 24212. LOPCODE 24210 is a register counter having, as described above, data inputs connected from NAME Bus 20224 to receive SOPs from PARSER 20264. In a first mode, LOPCODE 24210 may operate as a latch, loaded with one SOP at a time from output of PARSER 20264. In a second mode, LOPCODE 24210 operates as a clock register to receive successive eight bit inputs from low order eight bits of NAME Bus 20224 (NAME(8—15)). Loading of LOPCODE 24210 is controlled by microinstruction control outputs (not shown for clarity of presentation) from FUSITT 11012.

As will be described further below, eight bit SOPs stored in LOPCODE 24210 are concatenated with the output of RDIAL 24212 to provide addresses to FUSDT 11010 and EUSDT 20266 to select SDPs corresponding to particular SOPs. That portion of these addresses provided from LOPCODE 24210, that is the eight bit SOPs, selects particular SDPs within a particular SD. Particular SDs are selected by that portion of these addresses which is provided from the contents of RDIAL 24212.

RDIAL 24212 receives and stores four bit Dialect Codes indicating the particular S-Language dialect currently being used by CS 10110 and executing the SOPs of a user's program. These four bit Dialect Codes are provided from JPD Bus 10142, as JPD (28—31). Loading of RDIAL 24212 with four bit Dialect Codes is controlled by microinstruction control signals provided form FUSITT 11012 (not shown for clarity of

Petitioner Apple Inc. - Exhibit 1004, p. 5247

presentation).

Four bit Dialect Codes in RDIAL 24212 define partitions in FUDISF 24218, AF 24220 and EUDISF 24222. Each partition contains SDPs for a different S-Language dialect, that is contains a different SD. FUDISF 24218, AF 24220 and EUDISF 24222 may contain, for example, eight 128 word partitions or four 256 word partitions. A single bit of Dialect Code, for example Bit 3, defines whether FUDISF 24218, AF 24220, and EUDISF 24222 contain four or eight partitions. If FUSDT 11010 and EUSDT 20266 contain four partitions, the two most significant bits of address into FUSDT 11010 and EUSDT 20266 are provided from Dilect Code Bits 1 and 2 and determine which partition is addressed. The lower order eight bits of address are provided from LOPCODE 24210 and determine which word in a selected partition is addressed. If FUSDT 11010 and EUSDT 20266 contain eight partitions, the three most significant bits of address into FUSDT 11010 and EUSDT 20266 are provided from Bits 0 to 2 of Dialect Code, to select a particular partition, and the lower seven bits of address are provided from LOPCODE 24210 to select a particular word in the selected partition.

As described above, LOPCODE 24210 eight bit output and RDIAL 24212's four bit output are concatenated together, through FUDISENC 24216, to provide a ten bit address input to FUSDT 11010 and EUSDT 20266. FUDISENC 24216 is an encoding circuit and will be described further below with reference to FUDISF 24218. As previously described, selection of FUDISF 24218, AF 24220, and EUDISF 24222 to receive address inputs from RDIAL 24212 and LOPCODE 24210 is controlled by microinstruction control enable inputs provided from FUSITT 11012 (not shown for clarity of presentation).

Referring to FUSDT 11010, both FUDISF 24218 and AF 24220 provide SDPs to FUSITT 11012, but do so for differing purposes. In general, microinstruction control operations may be regarded as falling into two classes. First, there are those microinstruction operations which are generic, that is general in nature and used by or applying to a broad variety of SOPs of a particular dialect or even of many dialects. An example of this class of microinstruction operation is fetches of operand values. FUDISF 24218 provides SDPs for this class of microinstruction operations. As described below, FUDISF 24218 is a fast access memory allowing a single microinstruction control output of FUSITT 11012 to parse an SOP from INSTB 20262 into LOPCODE 24210, and a corresponding SDP to be provided from FUDISF 24218. That is, an SOP of this generic class may be parsed from INSTB 20262 and a corresponding SDP provided from FUDISF 24218 during a single system clock cycle. Operation of FUDISF 24218 thereby enhances speed of operation of JP 10114, in particular at the beginning of execution of new SOPs.

The second class of microinstruction operations are those specific to particular SINTs or to particular groups of SINTs. These groups of SINTs may reside entirely within a particular dialect, for example FORTRAN, or may exist within one or more dialects. SDPs for this class of microinstruction operation are provided by AF 24220. As described further below, AF 24220 is slower than FUDISF 24218, but is larger. In general, AF 24220 contains SDPs of microinstruction sequences specific to particular SINTs. In general, generic microinstruction operations are performed before those operations specific to particular SINTs, so that SDPs are required from AF 24220 at a later time than those from FUDISF 24218. SDPs for specific SINT operations may therefore be provided from lower speed AF 24220 without a penalty in speed of execution of SOPs.

Referring again to FUDISF 24218, FUDISF 24218 is a 1,024 word by 6 bit fast access by polar memory. Each word contained therein, as described above, is an SDP, or address to start of a corresponding sequence of microinstructions in FUSITT 11012. As will be described further below, FUSITT is an 8K (8192) word memory. SDPs provided by FUDISF 24218 are each, as described above, 6 bits wide and may thus address a limited, 32 word area of FUSITT 11012's address space. FUDISF 24218 is enabled to provide SDPs to FUSITT 11012 by microinstruction control signals (not shown for clarity of presentation) from FUSITT 11012. FUDISF 24218 six bit SDPs are encoded by FUDISENC 24219 to address FUSITT 11012 address space in increments of 4 microinstructions, that is in increments of 4 address locations. FUDISF 24218 SDPs thereby address 4 microinstructions at a time from FUSITT 11012's microinstruction sequences. As will be described further below, mPC 20276 generates successive microinstruction addresses to FUSITT 11012 to select successive microinstructions of a sequence following an initial microinstruction selected by an SDP from FUSDT 11010. An FUDISF 24218 SDP will thereby select the first microinstruction of a 4 microinstruction block, and mPC 20276 will select the following 3 microinstructions of that 4 microinstruction sequence. A 4 microinstruction sequence may therefore be executed in line, or sequentially, for each FUDISF 24218 SDP provided in response to a generic SOP. FUDISENC 24219 encodes FUDISF 24218 six bit SDPs to select these 4 microinstruction sequences so that the least significant bit of these SDPs occupies the 24 bit of FUSITT 11012 address inputs, and so on. The two least significant bits of an FUSITT 11012 address, or SDP, provided from FUDISF 24218 are forced to 0 while the ninth and higher bits may be hard-wired to define any particular block of 128 addresses in FUSITT 11012. This hard-wiring of the most significant bits of FUSITT 11012 addresses from FUDISF 24218 allows a set of generic microinstruction sequences selected by FUDISF 24218 to be located as desired within FUSITT 11012's address space. FUDISENC 24219 is comprised of a set of driver gates.

As previously described, SDPs for generic microinstructions currently being utilized by CS 10110 in executing user's programs are written into FUDISF 24218 from Bits 10 to 15 of JPD Bus 10142 (JPD(10—15)). Addresses for loading SDPs into FUDISF 24218 are provided, as previously described, from LADDR 24214. LADDR 24214 is enabled to provide load addresses, and FUDISF 24218 is enabled to be

Petitioner Apple Inc. - Exhibit 1004, p. 5248

written into, by microinstruction control signals (not shown for clarity of presentation) provided from FUSITT 11012.

Referring to AF 24220, as previously described AF 24220 is of larger capacity than FUDISF 24218, but has slower access time. AF 24220 is a 1,024 word by 15 bit memory. In general, 2 clock cycles are required to obtain a DSP from AF 24220. During first clock cycle, an SOP is loaded into LOPCODE 24210 and, during second clock cycle, AF 24220 is addressed to provide a corresponding SDP. SDPs provided by AF 24220 are each 15 bits in width and thus capable of addressing a larger address space than that of FUSITT 11012. As previously described, FUSITT 11012 is an 8K word memory. If FUSITT 11012 is addressed by an AF 24220 SDP referring to an address location outside of FUSITT 11012's address space, FUSITT 11012 will generate a microinstruction Not In Control Store output to EVENT 20284 as described further below. An AF 24220 SDP resulting in this event will then be used to address certain microinstruction sequences stored in MEM 10112. These microinstructions will then be executed from MEM 10112, rather than from FUSDT 11010. This operation allows certain microinstruction sequences, for example rarely used microinstruction sequences, to remain in MEM 10112, thus freeing AF 24220 and FUSITT 11012's address spaces from more frequently used SOPs.

As previously described AF 24220 is loaded, with SDPs, for SINTs currently being used by CS 10110 in executing user's programs, from Bits 16—31 of JPD Bus 10142 (JPD(16—31)). Also as previously discussed, addresses to load SDPs into AF 24220 are provided from LADDR 24214. LADDR 24214 is enabled to provide load addresses and AF 24220 to receive SDPs, by microinstruction control signals (not shown for clarity of presentation) provided from FUSITT 11012.

Referring finally to EUSDT 20266, SDPs may be provided to EU 10122 from 3 sources. EU 10122 SDPs may be provided from EUDISF 24222, from JPD Bus 10142 or from literal fields of microinstructions provided from FUSITT 11012. EUDISF 24222's SDPs are each 12 bits in width and comprise 9 bits of address into EUSITT and 3 bits of operand format information.

EUDISF 24222 is 1,024 word by 12 bit memory. As previously described addresses to read SDPs from EUDISF 24222 are provided from OPCODEREG 20268 by concatenating a 4 bit Dialect Code from RDIAL 24212 and an 8 bit SOP from LOPCODE 24210. SDPs provided by EUDISF 24222 are provided as a first input to EUDISS 24224.

EUDISS 24224 is a multiplexer. As just described, a first input of EUDISS 24224 are SDPs from EUDISF 24222. A second 12 bit input of EUDISS 24224 is provided from Bits 20 to 31 of JPD Bus 10142 (JPD(20—31)). A third input of EUDISS 24224 is a 12 bit input provided from a literal field of an FUSITT 11012 microinstruction output. EUDISS 20224 selects one of these 3 inputs to be transferred on EUDIS Bus 20206 to be provided as an execute unit SDP to EUSITT. Selection between EUDISS 20224's inputs is provided by microinstruction control signals (not shown for clarity of presentation) provided from FUSITT 11012.

As previously described, EUDISF 24222 is loaded, with SDPs for S-Language dialects currently being used by CS 10110, from Bits 20 to 31 of JPD Bus 10142 (JPD(20—31)). Addresses to load SDPs into EUDISF 24222 are provided, as previously described, from LADDR 20214. FUSITT 11012 provides enable signals (not shown for clarity of presentation) to LADDR 24214 and EUDISF 24222 to enable writing of SDPs into EUDISF 24222.

The structure and operation of FUCTL 20214 circuitry for fetching and parsing SINs from MEM 10112 to provide Name syllables and SOPs, and for interpreting SOP to provide SDPs to FUSITT 11012 and EUSITT from FUSDT 11010 and EUSDT 20266, have been described above. As described above, SDPs provided by FUSDT 11010 and EUSDT 20266 are initial, or starting, addresses pointing to first microinstructions of sequences of microinstructions. Addresses for microinstructions following those initial microinstructions are provided by FUCTL 20214's next address generator circuitry which may include mPC 20276, BRCASE 20278, REPCTR 20280 and PNREG 20282, EVENT 20284 and SITTNAG 20286. mPC 20276, BRCASE 20278, REPCTR 20280 and PNREG 20282, and SITTNAG 20286 are primarily concerned with generation of next addresses during execution of microinstruction sequences in response to SOPs and will be described next below. EVENT 20284 and other portions of FUCTL 20214's circuitry are more concerned with generation of microinstruction sequences with regard to CS 10110's internal mechanisms operations and will be described in a later description. EU 10122 also includes next address generation circuitry and this circuitry will be described in a following description of EU 10122.

c.c.c. Next Address Generator 24310 (Fig. 243)

As stated above, in FU 10120 first, or initial, microinstuctions of microinstruction sequences for interpreting SOPs are provided by FUSDT 11010. Subsequent addresses of microinstructions within these sequences are, in general, provided by mPC 20276 and BRCASE 20278. mPC 20276, as described further below, provides sequential addresses for selecting sequential microinstructions of microinstruction sequences. BRCASE 20278 provides addresses for selecting microinstructions when a microinstruction Branch or microinstruction Case operation is required. REPCTR 20280 and PNREG 20282 provide addresses for writing, or loading, of microinstruction sequences into FUSITT 11012. Other portions of FUCTL 20214 circuitry, for example EVENT 20284, provides microinstruction sequence selection addresses to select microinstruction sequences for controlling operation of CS 10110's internal mechanisms. SITTNAS 20286 selects between these microinstruction address sources to provide to FUSITT 11012 those addresses

Petitioner Apple Inc. - Exhibit 1004, p. 5249

required to select microinstructions of the operation to be currently executed by CS 10110.

Referring to Fig. 243, a partial block diagram of FU 10120's Next Address Generator (NAG) 24310 is shown. In addition to FUSDT 11010, NAG 24310 includes mPC 20276, BRCASE 20278, EVENT 20284, REPCTR 20280 and PNREG 20282, a part of RCWS 10358, and SITTNAS 20286. EVENT 20284 is, as described above, primarily concerned with execution of microinstruction sequences for controlling CS 10110 internal mechanisms. EVENT 20284 as shown herein only to illustrate its relationships to other portions of NAG 24310. EVENT 20284 will be described further in a following description of FUCTL 20214's circuitry' controlling CS 10110's internal mechanisms. Similarly, operation of RCWS 10358 will be described in part in the present description of NAG 24310, and in part in a following description of control of CS 10110's internal mechanisms.

Referring first to NAG 24310's structure, interconnections of FUSDT 11010, RCWS 10358, mPC 20276, BRCASE 20278, REPCTR 20280, PNREG 20282, EVENT 20284, and SITTNAS 20286 have been previously described with reference to Fig. 202. NAG 24310's structure will be described below only wherein Fig. 243 differs from Fig. 202.

Referring first to SITTNAS 20286, SITTNAS 20286 is shown as comprised of EVENT Gate (EVNTGT) 24310 and Next Address Select Multiplexer (NASMUX) 24312. NASMUX 24312 is comprised of NAS. Multiplexer A (NASMUXA) 24314, NASMUXB 24316, NASMUXC 24318, and NASMUXD 24320. Outputs of EVNTGT 24310 and NASMUXA 24314 to NASMUXD 24320 are ORed to CSADR 20204 to provide microinstruction selection addresses to FUSITT 11012.

ADR 20202 is shown in Fig. 243 as comprised of nine buses, Address A (ADRA) Bus 24322 to Address I (ADRI) Bus 24338. Output of EVENT 20284 is connected to input of EVNTGT 24310 by ADRA Bus 24322. Outputs of REPCTR 20280 and PNREG 20282 and output of AF 24220 are connected to inputs of NASMUXA 24314 by, respectively, ADRB Bus 24324 and ADRC Bus 24326. Outputs of RCWS 10358 and FUDISENC 24219 are connected to inputs of NASMUXB 24316 by, respectively, ADRD Bus 24328 and ADRE Bus 24330. Outputs of BRCASE 20278 and second output of mPC 20276 are connected to inputs of NASMUXC 24318 by, respectively, ADRF Bus 24332 and ADRG Bus 24334. Second output of mPC 20276 and JAM output of NC 10226 are connected to inputs of NASMUXD 24320 by, respectively, ADRH Bus 24336 and ADRI Bus 24338. ADR 20202 thus comprises a set of buses connecting microinstruction address sources to inputs of SITTNAS 20286.

Referring to mPC 20276, mPC 20276 is comprised of Micro-Program Counter Counter (mPCC) 24340 and Micro-Program Counter Arithmetic and Logic Unit (mPCALU) 24342. Data input of mPCC 24340 is connected from CSADR Bus 20204. Output of mPCC 24340 is connected to a first input of mPCALU 24342 and is mPC 20276's third output to BRCASE 20278. Second input of mPCALU 24342 is a fifteen binary number set, for example by hard-wiring, to be binary one. Output of mPCALU 24342 comprises mPC 20276's first output, to RCWS 10358, and mPC 20276's second output, to inputs of NASMUXC 24318 and NASMUXD 24320.

BRCASE 20278 is shown in Fig. 243 as comprising Mask and Shift Multiplexer (MSMUX) 24344, Case Mask and Shift Logic (CASEMS) 24346, Branch and Case Multiplexer (BCMUX) 24348 and Branch and Case Arithmetic and Logic Unit (BCALU) 24350. A first input of MSMUX 24344 (AONBC, not previously shown) is connected from output of AONGRF 20232. A second input of MSMUX 24344 (OFFMUXR, not previously shown) is connected from output of OFFMUXR 23812. Output of MSMUX 24344 is connected to input CASEMS 24346, and output of CASEMS 24346 is connected to a first input of BCMUX 24348. A second input of BCMUX 24348, BLIT is connected from a literal field output of FUSITT 11012's microinstruction output. Output of BCMUX 24348 and third output of mPC 20276, from output of mPCC 24340, are connected, respectively, to first and second inputs of BCALU 24350. Output of BCALU 24350 comprises BRCASE 20278 outputs to NASMUXC 24318.

An address to select a next microinstruction may be provided to FUSITT 11012 by SITTNAS 20286 from any of eight sources. First source is output of mPC 20276. Output of mPC 20276 is referred to as Micro-Program Count Plus 1 (mPC+1) and is fifteen bits of address. Second source is from EVENT 20284 and is comprised of five bits of address. Third source is output of FUDISP 24218 and FUDISENC 24219 and, as previously described, is comprised of six bits of address. Fourth source is output of AF 24220 and, as previously described, is comprised of fifteen bits of address. Fifth source is output of BRCASE 20278. Output of BRCASE 20278 is referred to as Branch and Case Address (BRCASEADR) and comprises fifteen bits of address. Sixth source is an output of RCWS 10358. Output of RCWS 10358 is referred to as RCWS Address (RCWSADR) and is comprised of fifteen bits of address. Seventh source is REPCTR 20280 and PNREG 20282 whose outputs (REPPN) together comprise fifteen bits of address. Finally, eighth source is JAM input from NC 10226, which comprises five bits of address. These address sources differ in number of bits of address that they provide, but a microinstruction address gated onto CSADR Bus 20202 by SITTNAS 20286 always comprises fifteen bits of address. If a particular source applies fewer than fifteen bits, that address is extended to fifteen bits by SITTNAS 20286. In general, extension of address bits may be performed by hard-wiring of additional address input bits to SITTNAS 20286 from each of these sources and will be described further below.

Referring to mPC 20276, mPCC 24340 is a fifteen bit register and mPCALU 24342 is a fifteen bit ALU. mPCC 24340 is, as described above, connected from CSADR Bus 20204 and is sequentially loaded with a microinstruction address currently being presented to FUSITT 11012. mPCC 24340 will thus contain the

Petitioner Apple Inc. - Exhibit 1004, p. 5250

address of the currently executing microinstruction. mPCALU 24342 is dedicated to incrementing the address contained in mPCC 24340 by one. mPC+1 output of mPCALU 24342 will thereby always be address of next sequential microinstruction. mPC+1 is, as described above, a fifteen bit address and is thus not extended in SITTNAS 20286.

Referring to BRCASE 20278, as described above BRCASE 20278 provides next microinstruction addresses for mPC 20276 Relative Branches and for Case Branches. Next microinstruction addresses for microprogram Relative Branches and for Case Branches are both generated as addresses relative to address of currently executing microinstruction as stored in mPCC 24340, but differ in the manner in which these relative addresses are generated. Considering first Case Branches, Case Branch addresses relative to a currently executing microinstruction address are generated, in part, by MSMUX 24344 and CASEMS 24346. As described above, MSMUX 24344 which is a multiplexer receives two inputs. First input is AONBC from output of AONGRF 20232 and second input is OFFMUXR from output of OFFMUXR 23812. Each of these inputs is eight bits, or one byte, in width. Acting under control of microinstruction output from FUSITT 11012, MSMUX 24344 selects either input AONBC or input OFFMUXR as an eight bit output to input of CASEMS 24346. CASEMS 24346 is a Mask and Shift circuit, similar in structure and operation to that of FIU 20116 but operating upon bytes rather than thirty-two bit words. CASEMS 24346, operating under microinstruction control from FUSITT 11012, manipulates eight bit input from MSMUX 24344 by masking and shifting to provide eight bit Case Value (CASEVAL) output to BCMUX 24348. CASEVAL represents a microinstruction address displacement relative to address of a currently executing microinstruction and, being an eight bit number, may express a displacement of 0 to 255 address locations in FUSITT 11012.

BCMUX 24348 is an eight bit multiplexer, similar in structure and operation to MSMUX 24344, and is controlled by microinstruction inputs provided from FUSITT 11012. In executing a case operation, BCMUX 24348 selects CASEVAL input to MCMUX 24348's output to first input of BCALU 24350. BCALU 24350 is a sixteen bit arithmetic and logic unit. Second input of BCALU 24350 is fifteen bit address of currently executing microinstruction from mPCC 24340. BCALU 24350 operates under microinstruction control provided from FUSITT 11012 and, in executing a Case operation, adds CASEVAL to the address of a currently executing microinstruction. During a Case operation, carry input of BSALU 24350 is forced, by microinstruction control from FUSITT 11012, to one so that BCALU 24350's second input is effectively mPC+1, or address of currently executing microinstruction plus 1. Output BRCASEADR of BCALU 24350 will thereby be fifteen bit Case address which is between one and 256 FUSITT 11012 address locations higher than the address location of the currently executing microinstruction. The actual case value address displacement from the address of the currently executing microinstruction is determined by either input AONBC or input OFFMUXR to MSMUX 24344, and these mask and shift operations are performed by CASEMS 24346.

Case operations as described above may be used, for example, in interpreting and manipulating CS 10110 table entries. For example, Name Table Entries of Name Tables 10350 contain flag fields carrying information regarding certain operations to be performed in resolving and evaluating those Name Table Entries. These operations may be implemented as Case Branches in microinstruction sequences for resolving and evaluating those Name Table Entries. In the present example, during resolve of a Name Table Entry the microinstruction sequence for performing that resolve may direct a byte of that Name Table Entry's flag field to be read from AONGRF 20232, or OFFMUXR 23812, and through MSMUX 24344 to CASEMS 24346. That microinstruction sequence will then direct CASEMS 24346 to shift and mask that flag field byte to provide a CASEVAL. That CASEVAL will have a value dependent upon the flags within that flag field byte and, when added to mPC+1, will provide a FUSITT 11012 microinstruction address for a microinstruction sequence for handling that Name Table Entry in accordance with those flag bits.

As described above, BRCASE 20278 may also generate microinstruction addresses for Branches occurring within execution of a given microinstruction sequence. In this case, microinstruction control signals from FUSITT 11012 direct BCMUX 24348 to select BCMUX 24348's second input as output to BCALU 24350. BCMUX 24348's second input is Branch Literal (BLIT). As described above, BLIT is provided from a literal field of a microinstruction word from FUSITT 11012's microinstruction output. BLIT output of BCMUX 24348 is added to address of currently executing microinstruction from mPCC 24340, and BCALU 24350, to provide fifteen bit BRCASEADR of a microinstruction address branched to from the address of the currently executing microinstruction. BRCASEADR may represent, for example, any of four Branch Operations. Possible Branch Operations are: first, a Conditional Short Branch; second, a Conditional Short Call; third, a Long Go To; and, fourth, a Long Call. In each of these possible Branch Operations, BLIT is treated as the twos complement of the desired branch value, that is the microinstruction address offset relative to the address of the currently executing microinstruction. BLIT field may therefore be, effectively, added to or subtracted from the address of the currently executing microinstruction, to provide a microinstruction address having a positive or negative displacement from the address of the currently executing microinstruction. In a Conditional Short Branch or a Conditional Short Call, the fourteen bit literal field is a sign extended eight bit number. Both Conditional Short Branch and Conditional Short Call microinstruction addresses may therefore point to an address within a range of +127 to −128 FUSITT 11012 address locations of the address of the currently executing microinstruction. In the case of a Long Go To or Long Call, the BLIT field is a fourteen bit number representing displacement relative to the address of the currently executing microinstruction. BRCASEADR may, in these cases, represent a FUSITT 11012

Petitioner Apple Inc. - Exhibit 1004, p. 5251

microinstruction address within a range of +8191 to −8192 FUSITT 11012 address locations of the address of the currently executing microinstruction. BRCASE 20278 thereby provides FU 10120 with capability of executing a full range of microinstruction sequence Case and Branch operations.

Referring to RCWS 10358, as previously described RCWS 10358 stores information regarding microinstruction sequences whose execution has been halted. RCWS 10358 allows execution of those microinstruction sequences to be resumed at a later time. A return control word (RCW) may be written onto RCWS 10358 during any microinstruction sequence that issues a Call to another microinstruction sequence. The calling microinstruction sequence may, for example, be aborted to service an event, as described further in a following description, or may result in a Jam. A Jam is a call for a microinstruction sequence which is forced by operation of CS 10110 hardware, rather than by a microinstruction sequence. RCWS 10358 operation with regard to CS 10110's internal mechanisms will be described in a following description of EVENT 20284, STATE 20294, and MCW1 20290 and MCWO 20292. For purposes of the present discussion, that portion of a RCW concerned with interpretation of SOPs contains, first, certain state information from FUSITT 11012 and, second, a return address into FUSITT 11012. State that FUSITT 11012 state is provided from STATE 20294, as described below, and that portion of a RCW containing FUSITT 11012 state information will be described in a following description. Microinstruction address portions of RCWs are provided from output of mPCALU 24342. This microinstruction address is the address of the microinstruction to which FU 10120 is to return upon return from a Call, Event, or Jam. Upon occurrence of a Call or Jam, the microinstruction return address is mPC+1, that is the address of the microinstruction after the microinstruction issuing the Call or Return. For aborted microinstruction sequences, the microinstruction return address is mPC, that is the address of the microinstruction executing at the time abort occurs.

Upon return from a call, service of an event, or service of a jam, FU 10120 state flag portion of RCW is loaded into STATE 20294. Microinstruction return address is provided by RCWS 10358 as fifteen bit RCWSADR to SITTNAS 20286 and is gated onto CSADR 20204. RCWSADR is provided to FUSITT 11012 to select the next microinstruction and is loaded into mPCC 24340 from CSADR 20204.

As previously described, RCWS 10358 is connected to JPD Bus 10142 by a bi-directional bus. RCWs may be written into RCWS 10358 from JPD Bus 10142, or read from RCWS 10358 to JPD Bus 10142. The fifteen bit next microinstruction address portion, and the single bit FUSITT 11012 state portion of RCW is written from or read to Bits 16 to 31 of JPD Bus 10142. FU 10120 may write Present Bottom RCW or Previous RCW into RCWS 10358 from JPD Bus 10142 and may read Present Bottom RCW, or Previous RCW, or another selected RCW, onto JPD Bus 10142. RCWS 10358 thereby provides a means for storing and returning microinstruction addresses of microinstruction sequences whose execution has been suspended, and a means for writing and reading microinstruction address, and FUSITT 11012 state flags, from and to JPD Bus 10142.

As previously described, REPCTR 20280 and PNREG 20282 provide microinstruction addresses for writing of microinstructions into FUSITT 11012. REPCTR 20280 is an eight bit counter and PNREG 20282 is a seven bit register. Eight bit output of REPCTR 20280 is left concatenated with seven bit output of PNREG 20282 to provide fifteen bit microinstruction addresses REPPN. That is, REPCTR 20280 provides the eight low order bits of microinstruction address while PNREG 20282 provides the seven most significant bits of address.

REPCTR may be loaded from Bits 24—31 of JPD Bus 10142, and may be read to Bits 24—31 of JPD Bus 10142. In addition, the eight bits of microinstruction address in REPCTR 20280 may be incremented or decremented as microinstructions are written into FUSITT 11012.

As described above, PNREG 20282 contains the seven most significant bits of microinstruction address. These address bits may be written into PNREG 20282 from Bits 17—23 of JPD Bus 10142. Contents of PNREG 20282 may not, in general, be read to JPD Bus 10142 and may not be incremented or decremented.

Referring to JAM input to SITTNAS 20286 from NC 10226, certain Name evaluate or resolve operations may result in jams. A Jam functions as a call to microinstruction sequences for servicing Jams and are forced by FU 10120 hardware circuitry involved in Name syllable evaluates and resolves.

JAM input to SITTNAS 20286 is comprised of six Jam address bits. Three bits are provided by NC 10226 and three bits are provided from FUSITT 11012's microinstruction output as part of microinstruction sequences for correcting Name syllable evaluates and resolves. The three bits of address from NC 10226 form the most significant three bits of JAM address. One of these bits gates JAM address onto CSADR Bus 20204 and is thus not a true address bit. Output of FUSITT 11012 provides the three least significant bits of JAM address and specifies the particular microinstruction sequence required to service the particular Jam which has occurred. Therefore, during Name evaluate or resolves, the microinstruction sequences provided by FUSITT 11012 to perform Name evaluates or resolves specifies what microinstruction sequences are to be initiated if a Jam occurs. The three bits of JAM address provided by NC 10226 determine, first, that a Jam has occurred and, second, provide two bits of address which, in combination with the three bits of address from FUSITT 11012, specify the particular microinstruction sequence for handling that Jam. JAM address inputs from NC 10226 and from FUSITT 11012 thereby provide six of the fifteen bits of JAM address. The remaining nine bits of JAM address are provided, for example, by hard-wired inputs to NASMUXD 24320. These hard-wired address bits force JAM address to address FUSITT

Petitioner Apple Inc. - Exhibit 1004, p. 5252

11012 in blocks of 4 microinstruction addresses, in a manner similar to address inputs to FUDISF 24218 and FUDISENC 24219.

Address inputs provided to SITTNAS 20286 from FUSDT 11010 have been previously described with respect to description of FUCTL 20214 fetch, parse, and dispatch operations. Address inputs provided by EVENT 20284 will be described in a following description of FUCTL 20214's operations with regard to CS 10110's internal mechanisms.

Referring finally to SITTNAS 20286, as previously described SITTNAS 20286 is comprised of EVNTGT 24310 and NASMUX 24312. Inputs are provided to NASMUX 24312, as described above, from FUSDT 11010, mPC 20276, BRCASE 20278, RCWS 10358, REPCTR 20280 and PNREG 20282, and by JAM input. These inputs are, in general, provided with regard to FUCTL 20214's operations in fetching, parsing, and interpreting SOPs and Name syllables. These operations are thereby primarily directly concerned with execution of user's programs, that is the execution of sequences of SINs. NASMUX 24312 selects between these inputs and transfers selected address inputs onto CSADR 20204 as microinstruction addresses to FUSITT 11012 under microinstruction control from microinstruction outputs of FUSITT 11012. Microinstruction address outputs are provided to SITTNAS 20286 from EVENT 20284 in response to Events, described further below, occurring in CS 10110's operations in executing user's programs. These microinstruction addresses from EVENT 20284 are gated onto CSADR 20204, to select appropriate microinstruction sequences, by EVNTGT 24310. EVNTGT 24310 is separated from NASMUX 24312 to allow EVNTGT 24310 to over-ride NASMUX 24312 and provide microinstruction address to EVENT 20284 while NASMUX 24312 is inhibited due to occurrence of certain Events. These Events are, in general, associated with operation of CS 10110's internal mechanisms and structure and operation of EVENT 20284, together with STATE 20294, MCW1 20290, and MCWO 20292, and other portions of RCWS 10358, will be described next below.

c.c. FUCTL 20214 Control Circuitry for CS 10110 Internal Mechanisms (Figs. 244—249)

Certain portions of FUCTL 20214's Control Circuitry are more directly concerned with operation of CS 10110's internal mechanisms, for example CS 10110 Stack Mechanisms. This circuitry may include STATE 20294, EVENT 20284, MCW1 20290 and MCWO 20292, portions of RCWS 10358, REG 20288, and Timers 20296. These FUCTL 20214 control elements will be described next below, beginning with STATE 20294.

a.a.a. State Logic 20294 (Figs. 244A—244Z)

In general, all CS 10110 operations, including execution of microinstructions, are controlled by CS 10110's Operating State. CS 10110 has a number of Operating States, hereafter referred to as States, each State being defined by certain operations which may be performed in that State. Each of these States will be described further below. Current State of CS 10110 is indicated by a set of State Flags stored in a set of registers in STATE 20294. Each State is entered from previous State and is exited to a following State. Next State of CS 10110 is detected by random logic gating distributed throughout CS 10110 to detect certain conditions indicating which State CS 10110 will enter next. Outputs of these Next State Detection gates are provided as inputs to STATE 20294's registers. A particular State register is set and provides a State Flag output when CS 10110 enters the State associated with that particular register. State Flag outputs of STATE 20294's state registers are provided as enable signals throughout CS 10110 to enable initiation of operations allowed within CS 10110's current State, and to inhibit initiation of operations which are not allowed within CS 10110's current State.

Certain of CS 10110's States, and associated STATE 20294 State Registers and State Flag outputs, are:

(1) MO: the initial State of any microinstruction.

State MO is always entered as first data cycle of every microinstruction. During MO, CS 10110's State may not be changed, thus allowing a microinstruction to be arbitrarily aborted and restarted from State MO. In normal execution of microinstructions, State MO is followed by State M1, described below, that is, State MO is exited to State M1. State M0 may be entered from State M0 and from State M1, State AB, State LR, State NR, or State MS, each of which will be described below.

(2) EP: Enable Pause State. State EP is entered when State MO is entered for the first time in a microinstruction. If that microinstruction requests a pause, that microinstruction will force State MO to be re-entered for one clock cycle. If State M0 lasts more than one clock cycle, State EP is entered on each extension of State M0 unless the extension is a result of a pause request.

(3) SR: Source GRF State. SR State is active for one clock cycle wherein SR State register enables loading of a GRF 10354 output register. State SR is re-entered on every State M0 cycle except a State M0 cycle generated by a microinstruction requesting extension of State M0. When all STATE 20294 State Registers are cleared, DP 20218 may set state SR register alone, for purposes of reading from GRF 10354.

(4) M1: Final state of normal microinstruction execution. State M1 is the exit State of normal microinstruction execution. FUSITT 11012 microinstruction register, described below, is loaded with a next microinstruction upon exit from State M1. In addition, State M1 Flag output of STATE 20294 enables all CS 10110 registers to receive data on their inputs, that is data on inputs of these registers are clocked to outputs of these registers. State M1 may be entered from State M1, or from State M0, State MW, State MWA, or State WB.

(5) LA: Load Accumulator Enable State. State LA is entered, upon exit from State M1, by

Petitioner Apple Inc. - Exhibit 1004, p. 5253

microinstructions which read data from MEM 10112 to OFFMUXR 23812. As previously described, OFFMUXR 23812 serves as a general purpose accumulator for DESP 20210. STATE LA overlaps into execution of next microinstruction, and persists until data is returned from MEM 10112 in response to a request to MEM 10112. When MEM 10112 signals data is available, by asserting DAVFA, LA State Flag enables loading of data into OFFMUXR 23812. If the next microinstruction references OFFMUXR 23812, that microinstruction execution is deferred until a read to OFFMUXR 23812 is completed, as indicated by CS 10110 exiting from State LA.

(6) RW: Load GRF 10354 Wait State. State RW is entered from State M1 of microinstructions which read data from MEM 10112 to GRF 10354. RW Fag Inhibits initiation of a next microinstruction, that is prevents entry to State M0, and persists through the CS 10110 clock cycle during which data is returned from MEM 10112 in response to a request. State RW initiates Load GRF Enable State, described below.

(7) LR: Load GRF Enable State. State LR is entered in parallel with State RW, on last clock cycle of RW, and persists for one CS 10110 clock cycle. LR Flag enables writing of MEM 10112 output data into GRF 10354.

(8) MR: Memory Reference Trailer State. State MR is entered on transition to State MO whenever a previous microinstruction makes a logical or physical address reference to MEM 10112. MR Flag enables recognition of any MEM 10112 reference Events, described below, which may occur. State MR persists for one clock cycle. If an MEM 10112 memory reference Event occurs, that Event forces exit from State MR to States AB and MA, otherwise State MR has no effect upon selection next state.

(9) SB: Store Back Enable State. State SB is entered during State M0 of a microinstruction following a microinstruction which generated a store back of a result of a EU 10122 operation. SB Flag gates that result to be written into MEM 10112 through JPD Bus 10142.

(10) AB: Microinstruction Abort State. State AB is entered from first MO State after an Event request is recognized, as described in a following description.

State AB may be entered from State MO or from State AB and suppresses an entry into State M1. If there has been an uncompleted reference to MEM 10112, that is, the reference has not been aborted and data has not returned from MEM 10112, JP 10114 remains in State AB until the MEM 10112 reference is completed. Should an abort have occurred due to a MEM 10112 reference Event, State AB lasts two clock cycles only. As will be described in a following description of EVENT 20284, State MO of a first microinstruction of a Handler for an Event causing an abort is entered from State AB. AB Flag gates the Handler address of the highest priority recognized Event onto CSADR Bus 20204 to select a corresponding Event Handler microinstruction sequence. EVENT 20284 is granted control of CSADR Bus 20204 during all State AB clock cycles.

(11) AR: Microinstruction Abort Reset State. State AR is entered in parallel with first clock cycle of State AB and persists for one clock cycle. AR Flag resets various STATE 20294 State Registers when an abort occurs. If there are no uncompleted MEM 10112 references, next State AB clock cycle is the last. On uncompleted MEM 10112 references, State AR is entered, but State AB remains active until reference is complete. Should a higher priority Event request service and be recognized while JP 10114 is in State AB, State AR is reentered. State AB will thereby be active for two clock cycles during all honored Event requests.

(12) MA: MEM 10112 Reference Abort. State MA is entered in parallel with State AB if a MEM 10112 reference is aborted, as indicated by asserted ABORT control signal output from MEM 10112. State MA persists for one clock cycle and State AB flag generates a MEM 10112 Reference Abort Flag which, as described below, results in a repeat of the MEM 10112 reference. AB Flag also resets MEM 10112 Trailer States, described below.

(13) NW: Nano-interrupt Wait State. State NW is entered from State M0 of a microinstruction which issues a Nano-interrupt Request to EU 10122 for an EU 10122 operation. FU 10120 remains in State NW until EU 10122 acknowledges that interrupt. Various EU 10122 Events may make requests at this time. State NW is exited into State AB or State M1.

(14) FM: First Microinstruction of a SIN. State FM is entered in parallel with State M0 on first microinstruction of each SIN and persists for one clock cycle. FM Flag inhibits premature use of AF 24220 and enables recognition of SIN Entry Events. State FM is re-entered upon return from all aborts taken during State M0 of the first microinstruction of an SIN.

(15) SOP: Original Entry to First SIN. State SOP is entered upon entry to State M0 of the first microinstruction of an SOP and is exited from upon any exit from that microinstruction. State SOP is entered only once for each SOP. SOP Flag may be used, for example, for monitoring performance of JP 10114.

(16) EU: EU 10122 Operand Buffer Unavailable. State EU is entered from State M0 of a microinstruction which attempts to read data to EU 10122 Operand Buffer, described in a following description, wherein EU 10122 Operand Buffer is full. When a new SOP is entered, three fetches of data from MEM 10112 may be performed before EU 10122 Operand Buffer is full; two fetches will fill EU 10122 Operand Buffer but EU 10122 may take one operand during a second fetch, thereby clearing EU 10122 Operand Buffer space for a third operand.

(17) NR: Long Pipeline Read. Entry into State NR disables overlap of MEM 10112 reads and disables execution of the next microinstruction. A following microinstruction does not enter State M0 until

Petitioner Apple Inc. - Exhibit 1004, p. 5254

requested data is returned from MEM 10112. State NR is entered from State NR or from State M1.

(18) NS: Nonpipeline Store Back. State NS is entered in parallel with State SB whenever a microinstruction requesting a pipeline store back, or a write to MEM 10112, occurs. State NS flag generates entry into State M0 of a following microinstruction upon exit from State SB.

(19) WA: Load Control Store State A. State WA is entered from State M0 of a microinstruction which directs loading of microinstruction into FUSITT 11012. WA State Flag controls selection of addresses to CSADR Bus 20204 for writing into FUSITT 11012, and generates a write enable pulse to FUSITT 11012 to write microinstructions into FUSITT 11012.

(20) WB: Load Control Store State B. State WB is entered from State WA and is used to generate an appropriate timing interval for writing into FUSITT 11012. State WB also extends State M1 to 2 clock cycles to ensure a valid address input to FUSITT 11012 when a next microinstruction is to be read from FUSITT 11012.

Having described certain CS 10110 states, and operations which may be performed within those states, state sequences for certain CS 10110 operations will be described next below with aid of Figs. 244A to 244Z. Fig. 244A to Fig. 244Z represent those state timing sequences necessary to indicate major features of CS 10110 state timing. All state timing shown in Figs. 244A to 244V assumes full pipelining of CS 10110 operations, for example pipelining of reads from and writes to MEM 10112 by JP 10114. Pipelining is not assumed in Figs. 244W to 244Z. Referring to Figs. 244A to 244Z, these figures are drawn in the form of timing diagrams, with time increasing from left to right. Successive horizontally positioned "boxes" represents successive CS 10110 states during successive CS 10110 110 nano-second clock cycles. Vertically aligned "boxes" represent alternate CS 10110 states which may occur during a particular clock cycle. Horizontally extended dotted lines connecting certain states represented in Fig. 244A to 244Z represent an indeterminate time interval which is an integral multiple of 110 nano-second CS 10110 clock cycles.

Referring to Fig. 244A to 244Z in sequence, State Timing Sequences shown therein represent:

(1) Fig. 244A; state timing for execution of a normal microinstruction with no Events occurring and no MEM 10112 references.

(2) Fig 244B execution of a normal microinstruction, with no Events occurring, no MEM 10112 references, and a hold in State M0 for one clock cycle.

(3) Fig. 244C; a microinstruction requests an extension of State M0 for one clock cycle, with no Events occurring and no MEM 10112 references.

(4) Fig. 244D; a write to MEM 10112 from DESP 20210, for example from GRF 10354 or from OFFALU 20242. MEM 10112 port is available and MEM 10112 reference is made during first sequential occurrence of States M0 and M1.

(5) Fig. 244E; a write to MEM 10112 from DESP 20210 as described above. MEM 10112 port is unavailable for an indeterminate number of clock cycles. A MEM 10112 reference is made during first sequential occurrence of States M0 and M1.

(6) Fig. 244F; writing of an EU 10122 result back into MEM 10112. MEM 10112 is available and a write operation is initiated during first sequential occurrence of States M0 and M1.

(7) Fig. 244G; writing back of an EU 10122 result to MEM 10112 as described above. MEM 10112 port is unavailable for an undetermined number of clock cycles, or EU 10122 does not have a result ready to be written into MEM 10112. Write operation is initiated during first sequential occurrence of States M0 and M1.

(8) Fig. 244H; a read of an EU 10122 result into FU 10120. EU 10122 result is not available for an undetermined number of clock cycles.

(9) Fig. 244I; a read from MEM 10112 to OFFMUXR 23812, with no delays. The microinstruction following the microinstruction initiating a read from MEM 10112 does not reference OFFMUXR 23812.

(10) Fig. 244J; a read from MEM 10112 to OFFMUXR 23812 with data from MEM 10112 being delayed by an indeterminate number of clock cycles. The next following microinstruction from that initiating the read from MEM 10112 does not reference OFFMUXR 23812.

(11) Fig. 244K; a read from MEM 10112 to OFFMUXR 23812. The next microinstruction following the microinstruction initiating the read from MEM 10112 references OFFMUXR 23812.

(12) Fig. 244L; a read from MEM 10112 to GRF 10354. The read to GRF 10354 is initiated by the first sequentially occurring States M0 and M1.

(13) Fig. 244M; a read from MEM 10112 to GRF 10354 and to OFFMUXR 23812. In this case, read operations may not be overlapped.

(14) Fig. 244N; JP 10114 honors an Event request and initiates a corresponding Event Handler microinstruction sequence, no MEM 10112 references occur.

(15) Fig. 2440; JP 10114 honors an Event request as stated above. MEM 10112 references are made during the first sequential occurrence of States M0 and M1 and a MEM 10112 reference Event occurs. In case of an MEM 10112 reference event, State MA is entered from one clock cycle. This occurs only if a MEM 10112 reference is made and aborted.

(16) Fig. 244P; an Event occurs in a MEM 10112 reference made during the first sequential occurrence of States M0 and M1. The MEM 10112 reference does not result in a memory reference Event. CS 10110 remains in State AB until the MEM 10112 reference is completed by return of data from MEM 10112.

(17) Fig. 244Q; a read of data from MEM 10112 or JPD Bus 10114 to EU 10122 Operand Queue. EU 10122 Operand Queue is not full.

Petitioner Apple Inc. - Exhibit 1004, p. 5255

(18) Fig. 244R; a read of MEM 10112 or JPD Bus 10142 data to EU 10122 Operand Queue. EU 10122 Operand Queue is full when the microinstruction initiating the read is issued.

(19) Fig. 244S; a request for a "nano-interrupt" to EU 10122 by FU 10120 with no Events occurring.

(20) Fig. 244T; FU 10120 submits a "nano-interrupt" request to EU 10122 and an EU 10122 State Overflow, described further in a following description, occurs. No other Events are recognized, as described in a following description of EVENT 20284.

(21) Fig. 244U; FU 10120 submits a "nano-interrupt" request to EU 10122. Another Event is recognized during State M0 and an abort results. First abort state is entered for the non-EU 10122 event. All aborts recognized in State M0 are taken or acknowledged, before entrance into State M0. Therefore, on retry at State M0 of the original microinstruction entered from State M0, next abort recognized is for EU 10122 Stack Overflow Event since EU 10122 Stack Overflow has higher priority.

(22) Fig. 244V; a load of a 27 bit microinstruction segment into FUSITT 11012.

In Figs. 244A to 244V, pipelining MEM 10112 reads and writes, and of JP 10114 operations, has been assumed. In Figs. 244W to 244Z, non-overlapping operation of JP 10114 is assumed.

(23) Fig. 244W; a read of data from MEM 10112 to OFFMUXR 23812.

(24) Fig. 244X; a read of data from MEM 10112 to EU 10122 Operand Queue.

(25) Fig. 244Y; a write of an EU 10122 result into MEM 10112.

(26) Fig. 244Z; a read of a 32 bit SIN word from MEM 10112 in response to a prefetch or conditional prefetch request.

Having described the general structure and operation of STATE 20294, and the operating states and operations of CS 10110, structure and operation of EVENT 20284 will be described next below.

b.b.b. Event Logic 20284 (Figs. 245, 246, 247, 248)

An Event is a request for a change in sequence of execution of microinstructions which is generated by CS 10110 circuitry, rather than by currently executing microinstructions. Occurrence of an Event will result in provision of a microinstruction sequence, referred to as an Event Handler, by FUSITT 11012 which modifies CS 10110's operations in accordance with the needs of that Event. Event request signals may be generated by CS 10110 circuitry internal to JP 10114, that is from FU 10120 or EU 10122 or CS 10110 circuitry external to JP 10114, for example from IOP 10116 or from MEM 10112. Event request signals are provided as inputs to EVENT 20284. As will be described further below, EVENT 20284 masks Event Requests to determine which Events will be recognized during a particular CS 10110 Operating State, assigns priorities for servicing multiple Event Requests, and fabricates Handler addresses to FUSITT 11012 for microinstruction sequences for servicing requests. EVENT 20284 then provides those Handler microinstruction addresses to FUSITT 11012 through EVNTGT 24310, to initiate execution of selected Event Handler microinstruction sequences.

Certain terms and expressions are used throughout the following description. The following paragraphs define these usages and provide examples illustrating these terms. An Event "makes a request" when a condition in CS 10110 hardware operation results in a Event Request signal being provided to EVENT 20284. As will be described further below, these Event Request signals are provided to EVENT 20284 combinatorial logic which determines the validity of those "requests".

An Event Request "is recognized" if it is not masked, that is inhibited from being acted upon. Masking may be explicit, using masks generated by FUSITT 11012, or may be implicit, resulting from an improper CS 10110 State or invalid due to other considerations. That is, certain Events are recognized only during certain CS 10110 States even though those requests may be recognized during certain other states. Any number of requests, for example up to 31, may be simultaneously recognized.

An Event Request is "honored" if it is the highest priority Event Request occurring. When a request is honored, a corresponding address, of a corresponding microinstruction sequence in FUSITT 11012, for its Handler microinstruction sequence is gated onto CSADR Bus 20204 by EVENT 20284. A request is honored when CS 10110 enters State AB. State AB gates the selected Event Handler microinstruction address on CSADR Bus 20284.

To summarize, a number of Events may request service by JP 10114. Of these Events, all, some, or none, may be recognized. Only one Event Request, the highest priority Event Request, will be honored when JP 10114 enters State AB. Microinstruction control of CS 10110 will then transfer to that Event's Handler microinstruction sequence. A necessary condition for entering State AB is that an Event Request has been made and recognized.

A microinstruction sequence "completes", "is completed", or reaches "completion" when CS 10110 exits State M1 while that microinstruction sequence is active. A microinstruction sequence may, as described above, be aborted in State M0 an indefinite number of times before, if ever, reaching completion.

A MEM 10112 reference "completes", "is completed", or reaches "completion" when requested data is returned to the specified destination, that is read from MEM 10112 to the requestor, or MEM 10112 accepts data to be written into MEM 10112.

"Trace Traps" are an inherent feature of microinstructions being executed. Trace Traps occur on every microinstruction of a given type (if not masked), for example during a sequence of microinstructions to perform a Name evaluate or resolve, and occur on each microinstruction of the sequence. In general, a Trace Trap Event must be serviced before execution of the next microinstruction. Trace Traps are distinct

Petitioner Apple Inc. - Exhibit 1004, p. 5256

from Interrupts in that an Interrupt, described below, does not occur on execution of each microinstruction of a microinstruction sequence, but only on those microinstructions where certain other conditions must be considered.

"Interrupts" are the largest class of events in JP 10114. Occurrence of an Interrupt may not, in general, be predicted for a particular execution of a particular microinstruction in a particular instance. Interrupts may require service before execution of the next microinstruction, before execution of the current microinstruction can complete, or before beginning of the next SIN. An Interrupt may be unrelated to execution of any microinstruction, and is serviced before beginning of the next microinstruction.

A "Machine Check" is an Event that JP 10114 may not handle alone, or whose occurrence makes further actions by JP 10114 suspect. These events are captured in EVENT 20284 Registers and result in a request to DP 10118 to stop operation of JP 10114 for subsequent handling.

In summary, three major classes of Events in CS 10110 are Trace Traps, Interrupts, and Machine Checks. Each of these class of events will be described in further detail below, beginning with Trace Traps.

The State of all possible Trace Trap Event Requests, whether requesting or not requesting, is loaded into EVENT 20284 Registers at completion of State M1 and at completion of State AB. That is, since Trap Requests are a function of the currently executing microinstruction, the State of a Trap Request will be loaded into EVENT 20284 Trace Trap Registers at end of State M1 of each currently executing microinstruction. Similarly, if any Trap Requests are recognized, State AB will be entered at the end of the first clock cycle of the next following State M0 and their State loaded at end of the State AB.

Recognized, or unmasked, Trap Requests may be pushed onto RCWS 10358 as Pending Requests. Unrecognized, or masked, Trace Trap Requests may be pushed onto RCWS 10358 as Not Pending Requests and are subsequently disregarded. Subsequently, when a microinstruction sequence ends in a return to a calling microinstruction sequence, the Trace Trap Request bits in an RCWS 10358 may be used to generate Trace Trap Event Requests.

Upon exit from State AB, all Trace Trap Requests, except Micro-Break-Point and Microinstruction Trace Traps, described below, are loaded into corresponding EVENT 20284 Trace Trap Request Registers as not requesting. Micro-Break-Point and Microinstruction Trace Traps, are, in general, always latched as requesting at completion of State AB. Trace Traps may be explicitly masked by a Trace Mode Mask, an Indivisibility Mode Mask, and by a Trace Enable input, all generated by FUSITT 11012 as described below. Micro-Break-Point Trap may also be masked by clearing a Trace Enable bit in a Trace Enable field of certain microinstructions containing Trace Traps. In general, masking is effective from State M0 of the microinstruction which generates the mask, through completion of a microinstruction which clears the mask Trace Traps generated by a microinstruction which clears a mask are taken so as to abort a following microinstruction during its M0 State.

Referring to Fig. 245, CS 10110 state timing for a typical Trap Request, and generation of a microinstruction address to a corresponding Trace Trap Handler microinstruction sequence by EVENT 20284 is shown. Fig. 245 is drawn using the same conventions as described above with reference to Fig. 244A to 244Z. In Fig. 245, a microinstruction executing in States M0 and M1 causes a Trace Trap Request but does not generate an MR (Memory Reference) Trailer State. Trace Trap Request to EVENT 20284 is signaled by Time A. This Trace Trap Request is latched into EVENT 20284 Trace Trap Event Registers, and an Abort Request is provided to STATE 20294. At Time B, FU 10120 enters States AB and AR. The microinstruction address for a Handler microinstruction sequence of the highest priority Event present in EVENT 20284 is presented to FUSITT 11012 and execution of the addressed microinstruction sequence begins. At Time C, FU 10120 exits States AB and AR and enters State AB. State AB will be exited at end of the next 110 nanosecond clock cycle. Address of the selected Event Handler microinstruction sequence will remain on CSADR Bus 20204 for duration of State AB. At Time D, a pointer into RCWS 10358, described in a following description, is incremented, thereby effectively pushing the first microinstruction's return control word, that is the microinstruction executing at first State M0, onto RCWS 10358. First microinstruction of the Trace Trap Event Handler microinstruction sequence is provided by FUSITT 11012. Execution of Handler microinstruction sequence will begin at start of the third State M0 of the state timing sequence shown in Fig. 245. EVENT 20284's Trace Trap Register for this event is now latched in nonrequesting state and will remain so until transition out of second State M1 shown in Fig. 245. At this time, EVENT 20284 Registers will latch new Trap Requests. Finally, at Time E, Trace Trap Event Registers of EVENT 20284 are latched with new Trap Requests arising from execution of the microinstruction being executed in States M0 and M1 occurring between Times D and E. Traps due to the microinstruction that was executed in States M0 and M1 before Time A, but were not serviced, are requested again when the previously pushed RCW described above is returned from RCWS 10358 upon return from the Trace Trap Event Handler microinstruction sequence initiated at Time D. All Trace Trap Requests which have been serviced are explicitly cleared in RCWS 10358 RCWs by their Event Handler microinstruction sequences to prevent recurrence of those Trap Requests. Since Trace Trap Event Requests arising from reads or writes to MEM 10112 will recur if those requests are repeated, EVENT 20284 generates memory repeat Interrupts after all aborted MEM 10112 read and write requests to insure that these Traps will eventually be serviced. Event Handler microinstruction sequences for these read and write Trace Trap Events explicitly disable serviced Trace Trap Event Requests by clearing bits in the logical descriptor of the aborted memory read and write requests.

Petitioner Apple Inc. - Exhibit 1004, p. 5257

Having described overall structure and operation of Trace Trap Events, certain specific Trace Trap Events will be described in greater detail below. Trace Trap Events occurring in CS 10112 may include Name Trace Traps, SOP Trace Traps, Microinstruction Trace Traps, Micro-Break-Point Trace Traps, Logical Write Trace Traps, Logical Read Trace Traps, UID Read Trace Traps, and UID Write Trace Traps. These Trace Traps will be described below in the order named.

A Name Trace Trap is requested upon every microinstruction sequence that contains an evaluate or resolve of a Name syllable. Name Trace Traps are provided by decoding certain microinstruction fields of those microinstruction sequences. Name Trace Trap field is masked by either Trace Mask, Indivisibility Mask, or Trace Enable, as described above. All of these masks are set and cleared by microinstruction control signals provided during microinstruction sequences calling for resolves or evaluates of Name syllables.

A SOP Trace Trap may be requested whenever FU 10120 enters State FM (First Microinstruction of an SOP). SOP Trace Traps may be masked by Trace Mask, Indivisibility Mask, or Trace Trap Enable, again provided by microinstruction control outputs of FUSITT 11012. In general, the first microinstruction of such a microinstruction sequence interrupting such SOPs is not completed before a Trace Trap is taken.

Microinstruction Trace Traps may be requested upon completion of microinstructions which do not contain a Return Command, that is those microinstructions which do not return microinstruction control of CS 10110 to the calling microinstruction sequence. For microinstruction sequences containing Return Commands, state of microinstruction Trace Trap Request in a corresponding RCW is used. Every microinstruction for which a Microinstruction Trace Trap is not masked is aborted during State M0 of execution of that microinstruction. Microinstruction Trace Traps may be masked by Trace Mask, Indivisibility Mask, or Trace Enable from FUSITT 11012. A Micro-Break-Point Trap may be requested upon execution of microinstructions which do not contain Return Commands, but in which a Trace Enable bit in a microinstruction is asserted. A Micro-Break-Point Trap may be masked by Trace Mask, Indivisibility Mask, or Trace Enable. In addition, a Trace Enable bit of a microinstruction field in these microinstruction sequences controls recognition of Micro-Break-point Traps. Micro-Break-Point Traps are thereby requested whenever a microinstruction Trace Trap is requested, but have additional enabling conditions expressed in the microinstructions. Since only recognized Traps are pushed onto RCWS 10358 in a RCW, a Microinstruction Trace Trap and a Micro-Break-Point Trap having different request states may be present in RCWS 10358 concurrently.

Logical Write Trace Traps may be requested when enabled by a bit set in a logical descriptor during a microinstruction sequence submitting a write request to MEM 10112 and using logical descriptors to do so. Logical Write Trace Traps are recognized only if they occur during a state which will be immediately followed by State MR (Memory Reference Trailer). A Logical Write Trace Trap will result in the MEM 10112 write request being aborted. Logical Write Trace Traps may be masked by Trace Masks, Indivisibility Mask, or Trace Trap Enable. A further condition for recognition of a Logical Write Trace Trap is determined by the state of certain bits in a logical descriptor of the memory write request. Logical Write Trace Traps are, in general, not pushed onto RCWS 10358 as part of a RCW since aborted MEM 10112 requests are re-generated so that Logical Write Trace Traps may be repeated.

Logical Read Trace Traps are similar in all respects to the Logical Write Trace Traps, but occur during MEM 10112 read requests. Generation of Logical Read Trace Traps is controlled again in part by certain bits in logical descriptors of MEM 10112 read requests.

In certain implementations of CS 10110, UID Trace Traps may be requested when FU 10120 requests an MEM 10112 read operation based upon a UID address or pointer. UID Read Trace Traps are recognized if requested and there is, in general, no explicit masking of UID Read Trace Traps. Generation of UID Read Trace Traps is controlled by certain bits in MEM 10112 read request logical descriptors. UID Read Trace Trap Requests result in the MEM 10112 read requests being aborted and CS 10110 entering State AB. Handler microinstruction sequences for UID Read Trace Traps will, in general, reset the trapped enable bit in the MEM 10112 read request logical descriptor before re-issuing the MEM 10112 read request.

UID Write Trace Traps are similar to UID Read Trace Traps, and are controlled by bits in the logical descriptor in MEM 10112 write request based upon UID addresses or pointers.

Having described above structure and operation of Trace Trap Events, CS 10110 Interrupt Events will be described next below.

As previously described, Interrupts form the largest class of CS 10110 Events. Interrupts may be regarded as falling into one or more of several classes. First, Memory Reference Repeat Interrupts are those Interrupt Events associated, in general, with read and write requests to MEM 10112 in which a read or write request is submitted to MEM 10112, and an Interrupt Event results. That Interrupt Event is handled, and the MEM 10112 request repeated. Second, Deferred Service Interrupts are those Interrupts wherein CS 10110 defers service of an Interrupt until entry to a new SIN. Fourth, Microinstruction Service Interrupts occur when a currently executing microinstruction requires assistance of an Event Handler microinstruction sequence to be completed. Finally, Asynchronous Interrupt Events may occur at any time and must be serviced before CS 10110 may exit State M0 of the next microinstruction. These Interrupt Events will be described next below in the order named.

A Memory Reference Repeat Interrupt is requested, for example, if a microinstruction executes a command, and a corresponding RCW read from RCWS 10358 indicates that a memory reference was

Petitioner Apple Inc. - Exhibit 1004, p. 5258

aborted before entrance to the microinstruction sequence from which return was executed. This type of Interrupt Event occurs for all aborted memory references. If an event is honored, that is abort state is entered, for any event and there is a memory reference outstanding, not aborted, the memory reference completes before State AB is exited. No memory Repeat Interrupt Request will be written into the RCW written onto RCWS 10358. Conversely, if a memory reference is aborted, even if the event honored is not that event which aborted the memory reference, a Memory Repeat Interrupt Request will be written into a RCW pushed onto a RCWS 10358.

There are two state timing sequences for execution of Memory Repeat Interrupts. In the first case, there are no MEM 10112 references in the mcroinstruction executing a Return Command. In the second case, a microinstruction executing a Return Command executes a return and also makes a MEM 10112 reference. Referring to Fig. 246, a CS 10110 State Timing Diagram for the first case is shown. Fig. 246 is drawn using the same conventions as used in Fig. 244 and 245. As described above, in the first case a microinstruction executing a Return Command is executed in States M0 and M1 following Time D. An aborted MEM 10112 reference was made in States M0 and M1 preceding Time A. An MEM 10112 Reference Abort Request is made upon CS 10110's entry into State MR following Time A. Since a Memory Repeat Interrupt is requested only from a RCW provided by RCWS 10358, a Memory Repeat Interrupt is indicated only if a microinstruction executes a Return Command resulting in RCWS 10358 providing such an RCW. Therefore, a Memory Repeat Interrupt Request Register of EVENT 20284 is loaded with "not requesting" at this time. At Time B, CS 10110 enters State AB, State AR, and State MA. At this time, a Memory Reference Abort Request is asserted and written into an RCW when State AB is exited just before Time D. At Time D, CS 10110 exits State AR and State MA. As just described, CS 10110 will remain in State B until Time D. At Time D, Memory Reference Abort Request is written into RCWS 10358 as part of an RCW and, as described further below, various RCWS 10358 Stack Pointers are incremented to load that RCW into RCWS 10358. At this time, EVENT 20284's Interrupt Request Register receives "no request" as state of Memory Repeat Interrupt. First microinstruction of Memory Repeat Interrupt Handler microinstruction sequence is provided by FUSITT 11012. At Time E, the last microinstruction of the Memory Repeat Interrupt Handler microinstruction sequence is provided by FUSITT 11012 and a Return Command is decoded. RCWS 10358 Previous Stack Pointer, previously described, is selected to address RCWS 10358 to provide the previously written RCW as output to EVENT 20284's Memory Repeat Interrupt Event Register. At Time F, EVENT 20284's Memory Repeat Interrupt Register is loaded from output of RCWS 10358 and RCWS 10358's Stack Register Pointers are decremented. At this time, Memory Repeat Interrupt Request is made and, as described below, is written into the current Return Control Word, whether honored or not. JP 10114 then repeats the aborted MEM 10112 reference.

In the second case, a State Timing Sequence wherein the microinstruction executing a return also makes a MEM 10112 reference, CS 10110 State Timing is identical up to Time F. At Time F, MEM 10112 Repeat request is not recognized and the state of Memory Repeat Interrupt written into the current Return Control Word is "not requesting" unless a current MEM 10112 reference is aborted. The previous MEM 10112 Repeat Interrupt Request is disregarded as it is assumed that it is no longer required. Thus, there are two ways to avoid, or cancel a Memory Repeat Interrupt Request. First, that portion of a RCW receiving a MEM 10112 Repeat Interrupt Request may be rewritten as "not requesting". Second, an aborted MEM 10112 reference may be made in the same microinstruction that returns from a Handler servicing the aborted MEM 10112 reference.

Certain CS 10110 Events result in aborting a MEM 10112 read or write references and may result in repeat of MEM 10112 references. These events may include:

(1) Logical read and write Traps and, in certain implementations of CS 10110, UID read and write Traps, previously discussed;

(2) A PC 10234 miss;

(3) Detection of a protection Violation by PC 10234;

(4) A Page Crossing in a MEM 10112 read or write request;

(5) A Long Address Translation, that is an ATU 10228 miss requiring JP 10114 to evaluate a logical descriptor to provide a corresponding physical descriptor;

(6) Detection of a reset dirty bit flag from ATU 10228 upon a MEM 10112 write request as previously described;

(7) An FU 10122 stack overflow;

(8) An FU 10122 Illegal Dispatch;

(9) A Name Trace Trap event as previously described;

(10) A Store Back Exception, as will be described below;

(11) EU 10122 Events resulting in aborting of a Store Back, that is a write request to MEM 10112 from EU 10122;

(12) A read request to a non-accelerated Stack Frame, that is a Stack Frame presently residing in MEM 10112 rather than accelerated to JP 10114 Stack Mechanisms; and,

(13) Conditional Branches in SIN sequences resulting outstanding MEM 10112 read reference from PREF 20260; and,

Of these Events, Logical Read and Write Traps, UID Read and Write Traps, and Name Trace Traps have been previously described. Other Events listed above will be described next below in further detail.

Petitioner Apple Inc. - Exhibit 1004, p. 5259

A PC 10234 Miss Interrupt may be requested upon a logical MEM 10112 reference, that is when a logical descriptor is provided as input to ATU 10228 and a protection state is not encached in PC 10234. PC 10234 will, as previously described, indicate that a corresponding PC 10234 entry is not present by providing a Event Protection Violation (EVENTPVIOL) output to EVENT 20284. PC 10234 will concurrently assert an Abort output (ABORT) to force CS 10110 into State AB and thus abort that MEM 10112 reference.

A Page Crossing MEM 10112 Reference Interrupt is requested if a logical MEM 10112 reference, that is a logical descriptor, specifies an operand residing on two logical pages of MEM 10112. An output of ATU 10228 will abort such MEM 10112 references by asserting an Abort output (ABORT).

A Protection Violation Interrupt is requested if a logical MEM 10112 reference does not possess proper access rights, a mode violation, or if that reference appears to refer to an illegal portion of that object, an extent violation. Again, PC 10234 will indicate occurrence of a Protection Violation Event, which may be disabled by a microinstruction control output of FUSITT 11012.

A Long Address Translation Event may be requested upon a logical MEM 10112 reference for which ATU 10228 does not have an encached entry. ATU 10228 will abort that MEM 10112 reference by asserting outputs ABORT and Long Address Translation Event (EVENTLAT).

A Dirty Bit Reset Event Interrupt may be requested when JP 10114 attempts to write to an MEM 10112 page having an encached entry in ATU 10228 whose dirty bit is not set. ATU 10228 will abort that MEM 10112 write request by asserting outputs ABORT and Write Long Address Translation Event (EVENTWLAT).

An FU 10120 User Stack Overflow Event may be requested if the distance between a Current Frame Pointer and a Bottom Frame Pointer, previously described with reference to CS 10110 Stack Mechanisms, is greater than a given value. As previously described, in CS 10110 this value is eight. A User Stack Overflow Event will continue to be requested until either Current Frame Pointer or Bottom Frame Pointer changes value so that the difference limit defined above is no longer violated. A User Stack Overflow Event may be masked by a Trace Mask, an Indivisibility Mask, or by enable outputs of a microinstruction from FUSITT 11012. A Handler microinstruction sequence for User Stack Overflow Events must be executed with one or more of these masks set to prevent recursion of these events. CS 10110 is defined to be running on Monitor Stack (MOS) 10370 when User Stack Overflow Events are masked. User Stack Overflow Events are not loaded into any of EVENT 20284's Event Registers, nor are these events written into a RCW to be written onto RCWS 10358.

Illegal EU 10122 Dispatch Events are requested by EUSDT 20266 if FU 10120 attempts to dispatch, or provide an initial microinstruction sequence address, to EU 10122 to a EUSITT address which is not accessable to a user's program. Illegal EU 10122 Dispatch Events are, in general, not masked. Illegal EU 10122 Dispatch Event Requests are cleared upon CS 10110 exits from State AB. The Handler microinstruction sequence for Illegal EU 10122 Dispatch Events should, in general, reset Illegal EU 10122 Dispatch Event entries in RCWs to prevent recursion of these events.

EU 10122 will indicate a Store Back Exception Event if any one of a number of exceptional conditions arise during arithmetic operations. These events are recognized when CS 10110 enters State SB and are ignored except during Store Back to MEM 10112 of EU 10122 results. These Events may be disabled by microinstruction output of FUSITT 11012 but are, in general, not masked. Store Back Exception Events may be written into RCWs, to be stored in RCWS 10358, and are cleared upon CS 10110's exit from State AB. Again, a Store Back Exception Event Handler microinstruction sequence should reset Store Back Exception Events written into RCWs to prevent recursion of these events.

As described above, the next major class of Interrupt Events are Deferred Service Interrupts. CS 10110 defers service of Deferred Service Interrupts until entry of a new SOP Deferred Service Interrupts which have been recognized will be serviced before completion of execution of the first microinstruction of that new SOP. Deferred Service Interrupts include Nonfatal MEM 10112 Errors, Interval Timer Overflows, and Interrupts from IOS 10116. These Interrupts will be described below, in the order named.

A Nonfatal MEM 10112 Interrupt is signaled by MEM 10112 upon occurrence of a correctable (single bit) MEM 10112 error. Nonfatal Memory Error Interrupts are recognized only during State M0 of the first microinstruction of an SOP. MEM 10112 will continue to assert Nonfatal Memory Error Interrupt until JP 10114 issues an acknowledgement to read MEM 10112's Error Log.

An Interval Timer Overflow Interrupt is indicated by TIMERS 20296 when, as described below, an Interval Timer increments to zero, thus indicating lapse of an allowed time limit for execution of an operation. Interval Timer Overflow Interrupts are recognized during State M0 of the first microinstruction of a SOP. TIMERS 20296 will continue to request such interrupts until cleared by a microinstruction output of FUSITT 11012.

IOS 10116 will indicate an IOS 10116 Interrupt to indicate that an inter-processor message from IOS 10116 to JP 10114 is pending. IOS 10116 will continue to assert an IOS 10116 Interrupt Request, which is stored in a register, until cleared by a microinstruction control output of FUSITT 11012. IOS 10116 Interrupts are recognized during State M0 of the first microinstruction of an SOP.

The next major class of CS 10110 events are Interrupts due to the requirement by microinstruction sequences to be serviced in order to complete execution. These Interrupts must be serviced before a microinstruction sequence may be completed. Microinstruction Service Interrupts include Illegal SOP Events, Microinstructions Not Present in FUSITT 11012 Events, an attempted parse of a hung INSTB 20262, underflow of an FU 10120 Stack, an NC 10226 Cache Miss, or an EU 10122 Stack Overflow. Each of these

Petitioner Apple Inc. - Exhibit 1004, p. 5260

events will be described below, in the order named.

An Illegal SOP Event is indicated by FUSDT 11010 to indicate that a current SOP Code is a Long Code, that is greater than eight bits, while the current dialect (S-Language) expects only Short Operation Codes, that is eight bit SOPs. An Illegal SOP Interrupt is not detected for unimplemented SOPs within the proper code length range. Illegal SOP Events are, in general, not masked. FUSDT 11010 continues to indicate an Illegal SOP Event until a new SOP is loaded into OPCODEREG 20268. Illegal SOP Events are recognized during the first microinstruction of an SOP, that is during State FM. Should a Handler microinstruction sequence for a higher priority event change contents of OPCODEREG 20268, a previous Illegal SOP Event will be indicated again when the aborted SOP is retried.

Absence of a Microinstruction in FUSITT 11012 is indicated by FUSITT 11012 asserting a Control Store Address Invalid (CSADVALID). This FUSITT 11012 output indicates that that particular microinstruction address points outside of FUSITT 11012's address space. Output of FUSITT 11012 in such event is not determined and parity checking, described below, of microinstruction output is inhibited. The Handler microinstruction sequence for these Events will load FUSITT 11012 address zero with the required microinstruction from MEM 10112, as previously described, and return to the original microinstruction sequence.

An attempted parse of a hung INSTB 20262 is indicated by INSTBWC 24110 when a parse operation is attempted, INSTB 20262 is empty, and PREF 20260 is not currently requesting SINs from MEM 10112. In general, these Events are not masked. If a higher priority Event is serviced, these Events are indicated again when the aborted microinstruction is retried if the original conditions still apply.

An FU 10120 Stack Underflow Event is requested when a current microinstruction references a Previous Stack Frame which is not in an accelerated stack, that is, the Current Stack Pointer equals Bottom Stack Pointer. FU 10120 Underflow Events are, in general, not masked and are requested again on a retry if the microinstruction is aborted and this event has not been serviced.

An NC 10226 Miss Interrupt occurs on a MEM 10112 read or write operation when a load or read of NC 10226 is attempted and there is no valid NC 10226 block corresponding to that Name syllable. An NC 10226 Miss Event does not result in a request for a Name evaluate or resolve. In general, these Events are not masked and result in a request being issued again if the microinstruction resulting in that Event is retried and has not been serviced.

An EU 10122 Stack Overflow Event is requested from EU 10122 to indicate that EU 10122 is currently already servicing at least one level of Interrupt an FU 10122 is requesting another. As will be described in a following description of EU 10122, EU 10122 contains a one level deep stack for handling of Interrupts. EU 10122 Stack Overflow Events are enabled during State NW. All previously pending events will have been serviced before EU 10122 Stack Overflow Event requests are recognized. These Events will be serviced immediately upon entry into a following State M0, being the highest priority interrupt event. EU 10122 Stack Overflow Events may, in general, not be masked and once recognized are the next honored event.

Finally, the third major class of CS 10110 Interrupt Events are Asychronous Events. Asynchronous Events must, in general, be serviced before exiting State M0 of a microinstruction after they are recognized. Asychronous Events include Fatal Memory Error Events, AC Power Failure Events, Egg Timer Overflow Events, and EU 10122 Stack Underflow Events. CS 10110 Egg Timer is a part of TIMERS 20296 and will be discussed as part of TIMERS 20296. These events will be described below, in the order referred to.

Fatal MEM 10112 Error Events are requested by MEM 10112 by assertion of control signal output PMODI, previously described, when last data read from MEM 10112 contains a noncorrectable error. Fatal MEM 10112 Error Events are recognized on first State M0 after occurrence. Fatal MEM 10112 Error Events are stored in an EVENT 20284 Event Register and are cleared upon entry into its service microInstruction sequence. In general, Fatal MEM 10112 Error Events may not be masked.

AC Power Failure Events are indicated by DP 10118 by assertion of output signal ACFAAIL when DP 10118 detects a failure of power to CS 10110. Recognition of AC Power Failure Events is disabled upon entry to AC Power Failure Event Handler microinstruction sequence. No further AC Power Failure Events will be recognized until DP 10118 reinitiates JP 10114 operation.

As will be described further below, FUCTL 20214's Egg Timer is a part of TIMERS 20296. Egg Timer Overflow Events are indicated by TIMERS 20296 whenever TIMERS 20296's Egg Timer indicates overflow of Egg Timer Counter. Egg Timer Overflow Events may be masked as described in a following description.

Finally, EU 10122 Stack Underflow Events are signaled by EU 10122 when directed to read a word from EU 10122 Stack Mechanism and there is no accelerated stack frame present. EU 10122 will continue to assert this Event Interrupt until acknowledged by JP 10114 by initiation of a Handler microinstruction sequence.

The above descriptions of CS 10110 events have stated that recognition of certain of those Events may be masked, that is inhibited to allow recognition of other Events having higher priority. Certain of these masking operations were briefly described in the above descriptions and will be described in further detail next below. In general, recognition of Events may be masked in five ways, four of which are properly designated as masks. These four masks are generated by microinstruction control from FUSITT 11012 and include Asychronous Masks for, in general, Asychronous Events. Monitor Masks are utilized for those CS 10110 operations being performed on Monitor Stack (MOS) 10370, as previously described with reference to CS 10110 Stack Mechanisms. Trace Mask is utilized with reference to Trace Trap Events. Indivisible Mask

Petitioner Apple Inc. - Exhibit 1004, p. 5261

is generated or provided by FUSITT 11012 as an integral or indivisible part of certain microinstructions and allow recognition of certain selected events during certain single microinstructions. Certain other Events, for example Logical Read and Write Traps and UID Read and Write Traps, are recognized or masked by flag bits in logical descriptors associated with those operations. Finally, certain microinstructions result in FUSITT 11012 providing microinstruction control outputs enabling or inhibiting recognition of certain events, but differ from Indivisible Masks in not being associated with single particular microinstructions.

Referring to Fig. 247, the relative priority level and applicable masks of certain CS 10110 Events are depicted therein in three vertical columns. Information regarding priority and masking of particular Events is shown in horizontal entries, each comprising an entry in each of these three vertical columns. Left hand column, titled Priority Level, states relative priority of each Event entry. Second column, titled EVENT, specifies which Event is referred to in that table entry. A particular Event will yield priority to all higher priority Events and will take presidence over all lower priority Events. Fig. 247's third column, titled Masked By, specifies for each entry which masks may be used to mask the corresponding Event. A indicates use of Asychronous Masks, M use of Monitor Mask, T use of Trace Trap Mask, and I represents that Indivisible Mask may be used. DES indicates that an Event is enabled or masked by flag bits of logical descriptors, while MCWD indicates that a particular Event may be masked by microinstruction control signal outputs provided by FUSITT 11012. NONE indicates that a particular Event may, in general, not be masked.

The final major class of CS 10110 event was described above as Machine Check Events. In general, if any of these Events are detected by logic gating in EVENT 20284, EVENT 20284 will provide a Check Machine signal to DP 10118. DP 10118 will then stop operation of JP 10114 and Machine Check Event Handler microinstruction sequences will be initiated. Among these Machine Check Events are wherein FU 10120 is attemting to store back an EU 10122 result to MEM 10112 and EU 10122 signals a parity error in EU 10122's Control Store. These events are stored in EVENT 20284 Event Registers and recogized when FU 10120 enters State AB. EU 10122 will have previously ceased operation until a corrective microinstruction sequence may be initiated. The same Event will occur if FU 10120 attempts to use an EU 10122 arithmetic operation result or test operation result having a parity error in EU 10122's Control Store. Should MOS 10370 overflow or underflow, this event will be detected, FU 10120 operations stopped, and corrective microinstruction sequences initiated. MOS 10370 overflow or underflow occurs whenever a previous MOS 10370 Stack Frame is referenced, whenever MOS 10370 Stack Pointer equals MOS 10370 Bottom Stack Pointer, or the difference between MOS 10370 Current and Bottom Stack Pointers is greater than sixteen. Underflows result in a transfer of operation to MIS 10368, while overflows are handled by DP 10118. Finally, a Machine Check Event will be requested when a parity error is detected in a microinstruction currently being provided by FUSITT 11012 during State M0 of that microinstruction.

Having described general operation of EVENT 20284, the structure and operation of EVENT 20284 will be described briefly next below.

Referring to Fig. 248, a partial block diagram of EVENT 20284 is shown. EVENT 20284 includes Event Detector (EDET) 24810, Event Mask and Register Circuitry (EMR) 24812, and Event Handler Selection Logic (EHS) 24814. EDET 24810 is comprised of random logic gating and, as previously described, receives inputs representing event conditions from other portions of CS 10110's circuitry. EDET 24810 detects occurrences of CS 10110 operating conditions indicating that Events have occurred and provides outputs to EMR 24812 indicating what Events are requested.

EMR 24812 includes a set of registers, for example SN74S194s, comprising EVENT 20284's Event Registers. These registers are enabled by mask inputs, described momentarily, to enable masking of those Events which are latched in EVENT 20284's Event Registers. Certain Events, as previously described, are not latched and logic gating having mask enable inputs is provided to enable masking of those events which are not latched. EMR 24812 mask inputs are Asychronous, Monitor, Trace Trap, and Indivisible Masks, respectively AMSK, MMSK, TMSK, and ISMK, provided from FUSITT 11012. Mask inputs derived from FUSITT 11012 microinstruction outputs (mWRD) are provided from microinstruction control outputs of FUSITT 11012. EMR 24812 provides outputs representing mask and unmask events which have been requested to EHS 24814.

EHS 24814 is comprised of logic gating detecting which of EHS 24814's unmasked Event Requests is of highest priority. EHS 24814 selects the highest priority unmasked Event Request input and provides a corresponding Event Handler microinstruction address to EVNTGT 24310 through ADRA Bus 24322. These address outputs of EHS 24814 are five bit addresses selecting the initial microinstruction of the Event Handler microinstruction sequence of the current highest priority unmasked Event. As previously described with reference to NASMUX 24312, certain inputs of ENTGT 24310 are hard-wired to provide a full fifteen bit address output from EVNTGT 24310. EVENT 20284 also provides, from EHS 24814, an Event Enable Select (EES) output to SITTNAS 20286 to enable EVNTGT 24310 to provide microinstruction addresses to CSADR Bus 20204 when EVENT 20284 must provide a microinstruction address for handling of a current Event.

Having described the structure and operation of FUCTL 20214's circuitry providing microinstruction addresses to FUSITT 11012, FUSITT 11012 will be described next below.

c.c.c. Fetch Unit S-Interpreter Table 11012 (Fig. 249)

Referring to Fig. 249, a partial block diagram of FUSITT 11012 is shown. Address (ADR) and Data

Petitioner Apple Inc. - Exhibit 1004, p. 5262

(DATA) inputs of Micro-Instruction Control Store (mCS) 24910 are connected, respectively, from CSADR Bus 20204 through Address Driver (ADRDRV) 24912 and from JPD Bus 10142 through Data Driver (DDRV) 24194. mCS 24910 comprises a memory for storing sequences of microinstructions currently being utilized by CS 10110. mCS 24910 is an 8K (8192) word by 80 bit wide memory. That is, mCS 24910 may contain, for example, up to, 8192 80 bit wide microinstructions. Microinstructions to be written into mCS 24910 are provided, as previously described, to mCS 24910 DATA input from JPD Bus 10142 through DDRV 24914. Addresses of microinstructions to be written into or read from mCS 24910 are provided to mCS 24910 ADR input from CSADR Bus 20204 through ADRDRV 24912. ADRDRV 24912 and DDRV 24914 are buffer drivers comprised, for example, of SN74S240s and SN74S244s.

Also connected from output of ADRDRV 24912 is input of Nonpresent Micro-Instruction Logic (NPmIS) 24916. NPmIS 24916 is comprised of logic gating monitoring read addresses provided to mCS 24910. When a microinstruction read address present on CSADR Bus 20204 refers to an address location not within mCS 24910's address space, that is of a non-present microinstruction, NPmIS 24916 generates an Event Request output indicating this occurrence. As previously described FUCTL 20214 will then call, and execute, microinstructions so addressed from MEM 10112.

As indicated in Fig. 249, mCS 24910 provides three sets of outputs. These outputs are Direct Output (DO), Direct Decoded Output (DDO), and Buffered Decode Output (BDO). In general, control information within a particular microstruction word is used on next clock cycle after the address of that particular microinstruction word has been provided to mCS 24910 ADR input. That is, during a first clock cycle a microinstruction's address is provided to mCS 24910 ADR input. That selected microinstruction appears upon mCS 24910's DO, DDO, BDO outputs during that clock cycle and are used, after decoding, during next clock cycle. Outputs DO, DDO, BDO differ in delay time before decoded microinstruction outputs are available for use.

mCS 24910 DO output provides certain bits of microinstruction words directly to particular destinations, or users, through Direct Output Buffer (DOB) 24918. These microinstructions bits are latched and decoded at their destinations as required. DOB 24918 may be comprised, for example, of SN74SO4s.

mCS 24910's DDO output provides decoded microinstruction control outputs for functions requiring the presence of fully decoded control signals at the start of the clock cycle in which those decoded control signals are utilized. As shown in Fig. 249, mCS 24910's DDO output is connected to input of Direct Decode Logic (DDL) 24920. DDL 24920 is comprised of logic gating for decoding certain microinstruction word bits during same clock cycle in which those bits are provided by mCS 24910's DDO. These microinstruction bits are provided, as described above, during the same clock cycle in which a corresponding address is provided to mCS 24910's ADR input. During this clock cycle, DDL 24920 decodes mCS 24910's DDO microinstruction bits to provide fully decoded outputs by end of this clock cycle. Outputs of DDL 24920 are connected to inputs of Direct Decode Register (DDR) 24922. DDR 24922 is a register comprised, for example, of SN74S374s. DDL 24920's fully decoded outputs are loaded into DDR 24922 at the end of the clock cycle during which, as just described, an address is provided to mCS 24910's ADR input and mCS 24910's corresponding DDO output is decoded by DDL 24920. Fully decoded microinstruction control outputs corresponding to mCS 24910's DDO outputs are thereby available at start of the second clock cycle. Microinstruction control outputs of DDR 24922 are thereby available to FU 10120 at start of the second clock cycle for those FU 10120 operations requiring immediate, that is undelayed, microinstruction control signal outputs from FUSITT 11012.

Finally, mCS 24910's BDO is provided for those FU 10120 operations not requiring microinstruction control signals immediately at the start of the second clock cycle. As shown in Fig. 249, mCS 24910's BDO is connected to inputs of Buffered Decode Register (BDR) 24924. Microinstruction word output bits from mCS 24910's BDO are provided to inputs of BDR 24924 during the clock cycle in which a corresponding address is provided to mCS 24910's ADR input. mCS 24910's BDO outputs are loaded into BDR 24924 at end of this clock cycle. BDR 24924's outputs are connected to inputs of Buffered Decode Logic (BDL) 24926. BDL 24926 is comprised of logic gating for decoding outputs of BDR 24924. BDL 24926 thereby provides decoded microinstruction control outputs to FU 10120 at some delayed time after start of the second clock cycle. Microinstruction control outputs from BDL 24926 are thereby delayed in time from the appearance of microinstruction control outputs of DDR 24922 but, as BDR 24924 stores microinstruction word bits rather than decoded microinstruction word bits, BDR 24924 is required to store proportionately fewer bits than DDR 24922.

Finally, as shown in Fig. 249 outputs of DDR 24922 and BDR 24924, are connected to inputs of Microinstruction Word Parity Checker (mWPC) 24928. mWPC 24928 is comprised of logic gating for checking parity of outputs of DDR 24922 and BDR 24924. A failure in parity of either output of DDR 24922 and BDR 24924 indicates a possible error in microinstruction output from mCS 24910. When such an error is detected by mWPC 24928, mWPC 24928 generates a corresponding Microinstruction Word Parity Error (mWPE).

d.d. CS 10110 Internal Mechanism Control

Associated with SR's 10362, the stack mechanism area of GRF 10354, are two CS 10110 control structures primarily associated with operation of CS 10110's internal mechanisms. A first of these referred to as Machine Control Block, describes current execution environment of JP 10114 microprograms, that is,

Petitioner Apple Inc. - Exhibit 1004, p. 5263

JP 10114 microinstruction sequences. Machine Control Block is comprised of two information words residing in MCW1 20290 and MCW0 20292. These Machine Control Words contain all control state information necessary to execute JP 10114's current microprogram. Second control structure is a portion of RCWS 10358, which as previously described parallels the structure of SR's 10362. Each register frame on MIS 10368 or MOS 10370 has, with exception of Top (Current) Register Frame, associated with it a Return Control Word (RCW) residing in RCWS 10358. RCWs are created when MIS 10362 or MOS 10370 register frames are pushed, that is moved onto MIS 10368 or MOS 10370 due to creation of a new Current Register Frame. A current RCW does not exist in a present embodiment of CS 10110.

RCWS 10358 will be described first next below, followed by Machine Control Block.

a.a.a. Return Control Word Stack 10358 (Fig. 251)

Referring to Fig. 251, a diagramic representation of a RCWS 10358 RCW is shown. As previously described, RCWS 10358 RCWs contain information necessary to reinitiate or continue execution of a microinstruction sequence if execution of that sequence has been discontinued.

Execution of a microinstruction sequence may be discontinued due to a requirement to service a CS 10110 Event, as described above, or if that microinstruction sequence has called for execution of another microinstruction sequence, as in a Branch or Case Operation.

As shown in Fig. 251, each RCW may contain, for example, 32 bits of information. RCW Bits 16 to 31 inclusive are primarily concerned with storing current microinstruction address of microinstruction sequences which have been discontinued, as described above. Bits 17 to 31 inclusive contain microinstruction sequence return address. Return address is, as previously described, address of the microinstruction currently being executed of a microinstruction sequence whose execution has been discontinued. When JP 10114 returns from servicing of an Event or execution of a called microinstruction sequence, return address is provided from RCWS 10358 to SITTNAS 20286 and through CSADR Bus 20204 to FUSITT 11012 as next microinstruction address to resume execution of that microinstruction sequence. Bit 16 of an RCW contains a state bit indicating whether the particular microinstruction referred to by return address field is the first microinstruction of a particular SOP. That is, Bit 16 of an RCW stores CS 10110 State FM.

Bits 8 to 15 inclusive of an RCW contain information pertaining to current condition code of JP 10114 and to pending Interrupt Requests. In particular, Bit 8 contains a condition code bit which, as previously described indicates whether a particular test condition has been met. RCW Bit 8 is thereby, as previously described, a means by which JP 10114 may pass results of a particular test from one microinstruction sequence to another Bits 9 to 15 inclusive of an RCW contain information regarding currently pending Interrupts. These Interrupts have been previously discussed, in general, with reference to EVENT 20284. In particular, RCW Bit 9 contains pending state of Illegal EU 10122 Dispatch Interrupt Requests; RCW Bit 10 contains pending state of Name Trace Trap Request; RCW Bit 11 contains pending state of Store Back Interrupt Request; RCW Bit 12 contains pending state of Memory Repeat Interrupt Request; RCW Bit 13 contains pending state of SOP Trace Trap Request; RCW Bit 14 contains pending state of Microtrace Trap Request; and, RCW Bit 15 contains pending state of Micro-Break Point Trap Request. Interrupt Handling microinstruction sequence which require use of CS 10110 mechanisms containing information regarding pending Interrupts must, in general, save and store that information. This save and restore operation is accomplished by use of Bits 9 to 15 of RCWS 10358's RCWs. Upon entry to an Interrupt Handling microinstruction sequence, these bit flags are set to indicate Interrupts which were outstanding at time of entry to that microinstruction sequence. Because these bits are used to initiate Interrupt Request upon returns, pending Interrupts may be cancelled by resetting appropriate bits of Bits 9 to 15 upon return. This capability may be used to implement Microinstruction Trace Traps, previously described.

As indicated in Fig. 251, RCW Bits 0 to 7 are not utilized in a present embodiment of CS 10110. RCW bits 0 to 7 are not implemented in a present embodiment of CS 10110 but are reserved for future use.

As previously described, RCWs may be written into or read from RCWS 10358 from JPD Bus 10142. This allows contents of RCWS 10358 to be initially written as desired, or read from RCWS 10358 to MEM 10112 and subsequently restored as required for swapping of processes in CS 10110.

b.b.b. Machine Control Block (Fig. 252)

As described above, FUCTL 20214's Machine Control Block is comprised of a Machine Control Word 1 (MCW1) and a Machine Control Word 0 (MCW0). MCW1 and MCW0 reside, respectively, in Registers MCW1 20290 and MCW0 20292. MCW1 and MCW0 described the current execution environment of FUCTL 20214's current microprogram, that is the microinstruction sequence currently being executed by JP 10114.

Referring to Fig. 252, diagramic representations of MCW0 and MCW1 are shown. As indicated therein, MCW0 and MCW1 may each contain, for example, 32 bits of information regarding current microprogram execution environment.

Referring to MCWO, MCW0 includes 6 execution environment subfields. Bits 0 to 3 inclusive contain a Top Of Stack Counter (TOSCNT) subfield which is a pointer to Current Frame of accelerated Microstack (MIS) 10368. TOSCNT field is initially set to point to Frame 1 of MIS 10368. Bits 4 to 7 inclusive comprise a Top of Stack -1 Counter (TOS-1CT) subfield which is a pointer to Previous Frame of accelerated MIS 10368, that is to the MIS 10368 frame proceeding that pointed by TOSCNT subfield. TOSICNT subfield is initially

Petitioner Apple Inc. - Exhibit 1004, p. 5264

set to Frame 0 of MIS 10368. Bits 8 to 11 inclusive comprise a Bottom of Stack Counter (BOSCNT) subfield which is a pointer to Bottom Frame of accelerated MIS 10368. BOSCNT subfield is initially set to point to Frame 1 of MIS 10368. TOSCNT, TOS-1CNT, and BOSCNT subfields of MCW0 may be read, written, incremented and decremented under microprogram control as frames are transferred between MIS 10368 and a SS 10336.

Bits 17 to 23 inclusive and Bits 24 to 31 inclusive of MCW0 comprise, respectively, Page Number Register (PNREG) and Repeat Counter (REPCTR) subfields which, together, comprise a microinstruction address pointing to a microinstruction currently being written into FUSITT 11012.

Bits 12 to 15 inclusive of MCW0 comprise an Egg Timer (EGGT) subfield which will be described further below with respect to TIMERS 20296. Bit 16 of MCW0 is not utilized in a present embodiment of CS 10110.

Referring to MCW1, MCW1 is comprised of four subfields. Of the 32 bits comprising MCW1, Bits 0 to 15 inclusive and Bits 24 and 25 are not utilized in a present embodiment of CS 10110. Bit 16 is comprised of a Condition Code (CC) subfield indicating results of certain test conditions in JP 10114. As previously described CC subfield is automatically saved and restored in RCWS 10358 RCW's.

Bits 17 to 19 inclusive of RCW1 comprise an Interrupt Mask (IM) subfield. The three bits of IM subfield are utilized to indicate a hierarchy of non-interruptible JP 10114 microinstruction control operating states. That is, a three bit code stored therein indicates relative power to interrupt between three otherwise noninterruptible JP 10114 operating states. Bits 20 to 23 inclusive comprise an Interrupt Request (IR) subfield which indicate Interrupt Request. These Interrupt Requests may include, for example, Egg Timer Overflow, Interval Timer Overflow, or Non-Fatal Memory Error, as have been previously described. Finally, Bits 26 to 31 inclusive comprise a Trace Trap Enable (TTR) subfield indicating which Trace Trap Events, previously described, are currently enabled. These enables may include Name Trace Enable, Logical Retrace Enable, Logical Write Trace Enable, SOP Trace Enable, Microinstruction Enable, and Microinstruction Break point Enable.

MCW0 and MCW1 has been described above as if residing in registers having individual, discrete existence, that is MCW1 20290 and MCW0 20292. In a present embodiment of CS 10110, MCW1 20290 and MCW0 20292 do not exist as a unified, discrete register structure but are instead comprised of individual registers having physical existence in other portions of FUCTL 20214. MCW1 20290 and MCW0 20292, and MCW1 and MCWO, have been so described to more distinctly represent the structure of information contained therein. In addition, this approach has been utilized to illustrate the manner by which current JP 10114 execution state may be controlled and monitored through JPD Bus 10142. As indicated in Fig. 202, MCW1 20290 and MCW0 20292 have outputs connected to JPD Bus 10142, thus allowing current execution state of JP 10114 to be read out of FUCTL 20214. Individual bits or subfields of MCW0 and MCW1 may, as previously described, be written by microinstruction control provided by FUSITT 11012. In a present physical embodiment of CS 10110, those registers of MCW0 20292 containing subfields TOSCNT, TOS-1CNT, and BOSCNT reside in RAG 20288. Those portions of MCW0 20292 containing subfield EGGT reside in TIMERS 20296. MCW0 20292 registers contain PNREG and REPCTR subfields are physically comprised of REPCTR 20280 and PNREG 20282. In MCW1 20290, CC subfield exists as output of FUCTL 20214 test circuits. Those MCW1 20290 registers containing IM, IR, and TTE subfields reside within EVENT 20284.

Having described FUCTL 20214 structure and operation as regards RCWS 10358, MCW1 20290 and MCW0 20292, FUCTL 20214, RAG 20288 will be described next below.

c.c.c. Register Address Generator 20228 (Fig. 253)

Referring to Fig. 253, a partial block diagram of RAG 20228, together with diagramic representation of GRF 10354, BIAS 20246 and RCWS 10358, is shown. As previously described, JP 10114 register and stack mechanisms include General Register File (GRF) 10354. BIAS 20246, and RCWS 10358. GRF 10354 is, in a present embodiment of CS 10110, a 256 word by 92 bit wide array of registers. GRF 10354 is divided horizontally to provide Global Registers (GRs) 10360 and Stack Registers (SRs) 10362, each of which contains 128 of GRF 10354's 256 registers. GRF 10354, that is both GRs 10360 and SRs 10362, is divided vertically into three vertical sections designated as AONGRF 20232, OFFGRF 20234, and LENGRF 20236. AONGRF 20232, OFFGRF 20234, and LENGRF 20236 are, respectively, 28 bits, 32 bits, and 32 bits wide. GRs 10360 is utilized as an array of 128 individual registers, each register containing one 92 bit word. SRs 10362 is structured and utilized as an array of 16 register frames wherein each frame contains eight registers and each register contains one 92 bit wide word. Eight of SR 10362's frames are utilized as Microstack (MIS) 10362 and the remaining eight of SR 10362's frames are utilized as Monitor Stack (MOS) 10370. For addressing purposes only, as described further below, GRs 10360 is regarded as being structured in the same manner as SRs 10362, that is as 16 frames of eight registers each.

BIAS 20246, as previously described, is a register array within BIAS 20246. BIAS 20246 contains 128 six bit wide registers, or words, and operates in parallel with and is addressed in parallel with SR 10362 portion of GRF 10354. RCWS 10358 is, as previously described, an array of 16 registers, or words, wherein each register contains one 32 bit RCW. RCWS 10358 is structured and operates in parallel with SRs 10362 with each RCWS 10358 register corresponding to a SR 10362 frame of eight registers. As described below, RCWS 10358 is addressed in parallel with SR 10362's frames.

Source and Destination Register Addresses (SDAR) for selecting a GRF 10354 register to be,

Petitioner Apple Inc. - Exhibit 1004, p. 5265

respectively, read from or written to are provided by RAG 20288. As described above BIAS 20246 operates and is addressed in parallel with SR 10362 portion of GRF 10354, that is parallel with SRs 10362. BIAS 20246 registers are thereby connected to and in parallel with address inputs of SRs 10362 and are addressed concurrently with GRs 10360. Registers RCWS 10358 also operate and are addressed in parallel with SRs 10362. Address inputs of RCWS 10358's registers are thereby connected in parallel with address inputs of SR 10362's registers.

RAG 20288's address inputs to GRF 10354, and to BIAS 20246 and RCWS 10358, may select registers therein to be either source registers, that is registers providing data, or destination registers, that is registers receiving data. RAG 20288's address outputs are designated as output Source and Destination Register Address (SDADR) of RAG 20288. RAG 20288's SDADR output is connected to address input of register comprising GRF 10354, BIAS 20246, and RCWS 10358. As described above, SRs 10362 are structured as 16 frames of 8 registers per frame and RCWS 10358 is structured as a corresponding 16 frames of one register per frame. GRF 10354 and BIAS 20246 are structured and utilized as single registers but, for addressing purposes, are regarded as being comprised of 16 frames of 8 registers per frame. Each SDADR output of RAG 20288 is an 8 bit word wherein the most significant bit indicates whether the addressed register, either a Source or a Destination Register, reside in GRs 10360 or within SRs 10362, BIAS 20246, and RCWS 10358. The four next most significant bits comprise a frame select field for selecting one of 16 frames within GRs 10360 or within SRs 10362, BIAS 20246, and RCWS 10358. The three least significant bits comprise a register select field selecting a particular register within the frame selected by frame select field.

Within a single system clock cycle, SDADR output of RAG 20288 may select a source register and data may be read from that source register, or SDADR output may select a destination register and data may be written into that destination register. As previously described, each JP 10114 microinstruction requires a minimum of two-system clock cycles for execution, that is at first clock cycle in State MO and a second clock cycle in State M1. During a single microinstruction therefore, a source register may be selected and data read from that source register, and a destination register selected and data written into that destination register. Certain operations, however, may require more than one microinstruction for execution. For example, a read-modify-write operation wherein data is read from a particular register, modified, and written back into that register may require two or more microinstructions for execution.

Referring first to RAG 20288 structure, RAG 20288 includes MISPR 10356. MISPR 10356 includes Top Of Stack Counter (TOSCNT) 25310, Top Of Stack-1 Counter (TOS-1CNT) 25312, and Bottom Of Stack Counter (BOSCNT) 25314. Contents of TOSCNT 25310, TOS-1CNT 25312 and BOSCNT 25314 are respectively, pointers to Current, Previous, and Bottom frames of SRs 10362, that is, to MIS 10368. As will be described below, these pointers are also utilized to address MOS 10370. TOSCNT 25310, TOS-1CNT 25312, and BOSCNT 25314 are each four bit binary counters comprised, for example, of SN74S163s.

Data inputs of TOSCNT 25310 to BOSCNT 25314 are connected from JPD Bus 10142. Control inputs of TOSCNT 15310 to BOSCNT 25314 areconnected from microinstruction control outputs of FUSITT 11012. Data outputs of TOSCNT 25310 to BSOCNT 25314 are connected to data inputs of Source Register Address Multiplexer (SRCADR) 25316 and to data inputs of Destination Register Address Multiplexer (DSTADR) 25318. Data outputs of TOSCNT 25310 and BOSCNT 25314 are connected to inputs of Stack Event Monitor Logic (SEM) 25320.

Source and destination frame addresses are selected, as will be described further below, by SRCADR 25316 and DSTADR 25318 respectively. In addition to data inputs from TOSCNT 25310 and BOSCNT 25314, data inputs of SRCADR 25316 and DSTADR 25318 are connected from microinstruction word CONEXT subfield output from FUSITT 11012. Control inputs of SRCADR 25316 and DSTADR 25318 are connected from, respectively, microinstruction word RS and RD subfield outputs from FUSITT 11012. Source Frame Address Field (SRCFADR) output of SRCADR 25316 and Destination Frame Address Field (DSTFADR) output of DSTADR 25318 are connected to inputs of Source and Destination Register Address Multiplexer (SDADRMUX) 25322. SRCFADR and DSTFADR comprise frame select fields of RAG 20288, SDADR outputs for, respectively, source and destination registers.

In addition to SRCFADR and DSTFADR outputs of ADRSRC 25316 and DSTADR 25318, SDADRMUX 25322 receives microinstruction word SRC and DST subfield inputs from microinstruction outputs of FUSITT 11012. As previously described, SRC subfield is a 3 bit number designating a source register, that is, a source register within a frame selected by SRCFADR. DST is similarly a 3 bit number selecting a destination register within a frame indicated by DSTFADR. SRC subfield input to SDADRMUX 25322 is concatenated with SRCADR 25316 to respectively comprise, as described above, register and frame fields of a source register SDADR output of SDADRMUX 25322. Similarly, DST subfield is concatenated with DSTFDADR output of DSTADR 25318 to comprise, respectively, register and frame subfields of a destination register SDADR output of SDADRMUX 25322. Selection between source and destination register address inputs to SDADRMUX 25322, to generate a corresponding source of destination register SDADR output of SDADRMUX 25322 is controlled by microinstruction control inputs (not shown for clarity of presentation) connected to control inputs of SDADRMUX 25322. RDWS 25324 is a PROM decoding MD field from microinstruction words during reads from MEM 10112 and provides register select field of destination register address and selects one of the pointers as frame select field.

An Event output of SEM 25320 is connected to an input of EVENT 20284, previously described.

Petitioner Apple Inc. - Exhibit 1004, p. 5266

SRCADR 25316, DSTADR 25318, and SDADRMUX 25322, as will be described further below, operate as multiplexers and may be comprised, for example, of SN74S153s.

Having described structure and organization of GRF 10354, BIAS 20246, and RCWS 10358, and structure of RAG 20288, operation of RAG 20288 to generate Source of Destination Register Address outputs SDADR will be described next below. Addressing of JP 10114's stack mechanism, comprising SRs 10362 and RCWS 10358, will be described first, followed by addressing of GRs 10360 and BIAS 20246.

SR 10362 portion of GRF 10354, RCWS 10358, and BIAS 20246 are addressed by Current, Previous, and Bottom Frame Pointers contained, respectively, in TOSCNT 25310, TOS-1CNT 25312, and BOSCNT 25314. Current, Previous, and Bottom Pointers comprise frame select fields of SDADRMUX 25322. As previously described, Current, Previous and Bottom Pointer outputs of TOSCNT 25310 to BOSCNT 25314 are provided as inputs of SRCADR 25316 and DSTADR 25318. Microinstruction word RS subfield to control input of SRCADR 25316 selects either Current, Previous or Bottom Pointer input of SRCADR 25316 to comprise SRCFADR output of SRCADR 25316, that is to be frame select field of source register address. Similarly, microinstruction word RD subfield to control input of DSTADR 25318 concurrently selects either Current, Previous, or Bottom Pointer inputs of DSTADR 25318 to comprise DSTADR 25318's concurrently selects either Current, Previous, or Bottom Pointer inputs of DSTADR 25318 to comprise DSTADR 25318s DSTFADR output, that is frame select field of destination register address. As described above, SRCFADR and DSTFADR are provided as inputs to SDADRMUX 25322. Microinstruction word SRC and DST subfield inputs to SDADRMUX 25322 concurrently determine, respectively, source and destination registers within source and destination frames specified by SRCFADR and DSTFADR. SDADRMUX 25322 then, operating under microinstruction control, selects either SRCFADR and SRC to comprise SDADR output to SR 10362 as a source register address or selects DSTFADR and DST as SDADR output specifying a destination register address. By microinstruction control of SRCADR 25316, DSTADR 25318, and SDADRMUX 25322, a CS 10110 microprogram may select a source frame and register within SR 10362 and simultaneously specify a possible different destination frame and register within SR 10362. All possible combinations of source frame and register and destination frame and register in GRs 10360, SRs 10362, BIAS 20246, and RCWS 10358 are valid.

Control of SRCADR 25316, DSTADR 25318, and SDADRMUX 25322 in addressing SR 10362 portion of GRF 10354, and RCWS 10358, is controlled, in part, by current CS 10110 state. Pertinent CS 10110 operating states, previously described, are State MI and State RW. When CS 10110 is in neither State RW nor State M1, SR 10362 is addressed through SRCADR 25316 and microinstruction word SRC subfield, that is SR 10362 and RCWS 10358 are provided with source register addresses when CS 10110 is in neither RW nor M1 States. When CS 10110 enters State M1, SR 10362 and RCWS 10358 is addressed through DSTADR 25318 and by microinstruction word DST subfield. That is, SR 10362 and RCWS 10358 are provided with destination register addresses during State M1. Similarly, SR 10362 and RCWS 10358 are provided with destination register addresses when CS 10110 is operating in State RW, that is when data is being read from MEM 10112 and written into SR 10362 or RCWS 10358. In this case, however, low order 3 bits of destination register address, that is register select field, are provided by RDS 25324, which decodes microinstruction word subfield MD (Memory Destination). RDS 25324 also provides a control input that DSTADR 25318 to select one of Current, Previous, or Bottom pointers from MISPR 10356 to comprise frame select field of destination register address.

As stated above, frame select field of source and destination register addresses are provided from TOSCNT 25310, TOS-1CNT 25312, and BOSCNT 25314. As described above, the most significant bit of source and destination register address are forced to logic 1 or logic 0, depending upon whether GR 10360 or SR 10362, BIAS 20246, and RCWS 10358 are being addressed. Contents of TOSCNT 25310 to BOSCNT 25314, that is Current, Previous, and Bottom Pointers, are controlled by microinstruction control outputs of FUSITT 11012. Current and Previous Pointers change as stacks are "pushed" or "popped" to and from MIS 10368 as JP 10114 performs, respectively, calls and returns. Similarly, Current, Previous and Bottom Pointers will be incremented or decremented as MIS 10368 frames are transferred between MIS 10368 and MEM 10112, as previously described with respect to CS 10110's Stack Mechanisms.

Referring first to Current and Previous Pointer operation, Current and Previous Pointers in TOSCNT 25310 and TOS-1CNT 25312 are initially set, respectively, to point to Frames 1 and 0 of MIS 10368 by being loaded from JPD Bus 10142. TOSCNT 25310 and TOS-1CNT 25312 are enabled to count when two conditions are met. First condition is dependent upon current operating state of CS 10110. TOSCNT 25310 and TOS-1CNT 25312 will be enabled to count during last system clock cycle of CS 10110 operating States M1 or AB. Second condition is dependant upon whether JP 10114 is to execute a call or return. TOSCNT 25310 and TOS-1CNT 25312 may be enabled to count if a current microinstruction indicates JP 10114 is to execute a call or return, or if CS 10110 is exiting State AB as exit from State AB is an implied call operation. Both a call and an implied call, that is exit from State AB, will cause TOSCNT 25310 and TOS-1CNT 25312 to be incremented. A return will cause TOSCNT 25310 and TOS1CNT 25312 to be decremented.

Referring to BOSCNT 25314, Bottom Frame Pointer is initially loaded from JPD Bus 10142 to point to MIS 10368 Frame 1. Again, incrementing or decrementing of BOSCNT 25314 is dependant upon CS 10110 operating state and operation to be performed. BOSCNT 25314 is enabled to count upon exiting from State M1. In addition, DEVCMD subfield of a current microinstruction word must indicate that BOSCNT 25314 is to be incremented or decremented. BOSCNT 25314 will be incremented or decremented upon exit from

Petitioner Apple Inc. - Exhibit 1004, p. 5267

State M1 as indicated by microinstruction word DEVCMD subfield.

SEM 25320 monitors relative values of Current and Bottom Pointers residing in TOSCNT 25310 and BOSCNT 25314 and provides outputs to EVENT 20284 for purposes of controlling operation of MI 10368 and MOS 10370. SEM 25320 is comprised of a Read Only Memory, for example 93S427s, receiving Current and Bottom Pointers as inputs. SEM 25320 detects 3 Events occurring in operation of TOSCNT 25310 and BOSCNT 25314, and thus in operation of MIS 10368 and MOS 10370. First, SEM 25320 detects an MIS 10368 Stack Overflow. This Event is indicated if the present value of Current Frame Pointer is greater than 8 larger than the present value of Bottom Frame Pointer. Second, SEM 25320 detects when MIS 10368 contains only one frame of information. This event is indicated if the value of Current Frame Pointer is equal to the value of Bottom Frame Pointer. In this case, the previous frame of MIS 10368 resides in MEM 10112 and must be fetched from MEM 10112 before a reference to the previous stack frame may be made. Third, SEM 25320 detects when MIS 10368 and MOS 10370 are full. This Event is indicated if the present value of Current Frame Pointer is 16 larger than the present value of Bottom Frame Pointer. When this Event occurs, any further attempt to write a frame onto MIS 10368 or MOS 10370 will result in a MOS 10370 Stack Overflow. EVENT 20284 responds to these Events indicated by SEM 25320 by initiating execution of an appropriate Event Handling microinstruction sequence, as previously decribed. It should be noted that MIS 10368 and MOS 10370 are addressed in the same manner, that is through use of Current, Previous and Bottom Frame Pointers and certain microinstruction word subfields. Primary difference between operation of MIS 10368 and MOS 10370 is in the manner in which stack overflows are handled. In the case of MIS 10368, stack frames are transferred between MIS 10368 and MEM 10112 so that MIS 10368 is effectively a bottomless stack. MOS 10370, however, contains a maximum of 8 stack frames, in a present embodiment of CS 10110, so that no more than eight Events may be pushed onto MOS 10370 at a given time.

GR 10360 is addressed in a manner similar to SR 10362, BIAS 20246, and RCWS 10358, that is through ADRSRC 25316, DSTADR 25318, and SDADRMUX 25322. Again, register select fields of source and destination register addresses are provided by microinstruction word SRC and DST subfields. Frame select field of source and destination register addresses is, however, specified by microinstruction word CONEXT subfield. In this case, microinstruction word RS and RD subfields specify that frame select fields of source and destination register addresses are to be provided by CONEXT subfield. Accordingly, ADRSRC 25316 and DSTADR 25318 provide CONEXT subfield as SRCFADR and DSTFADR inputs to SDADRMUX 25322.

Having described structure and operation of RAG 20288, TIMERS 20296 will be described next below.

Referring to Fig. 254, a partial block diagram of TIMERS 20296 is shown. As indicated therein, TIMERS 20296 includes Interval Timer (INTTMR) 25410, Egg Timer (EGGTMR) 25412, and Egg Timer Clock Enable Gate (EGENB) 25416.

d.d.d. Timers 20296 (Fig. 254)

Referring first to INTTMR 25410, a primary function of INTTMR 25410 is to maintain CS 10110 architectural time as previously described with reference to Fig. 106A and previous descriptions of CS 10110 UID addressing. As described therein, a portion of all UID addresses generated by all CS 10110 systems is an Object Serial Number (OSN) field. OSN field uniquely defines each object created by operation of or for use in a particular CS 10110. OSN field of an object's UID is, in a particular CS 10110, generated by determining time of creation of that object relative to an arbitrary historic starting time common to all CS 10110 systems. That time is maintained within a MEM 10112 storage space, or address location, but is measured by operation of INTTMR 25410.

INTTMR 25410 is a 28 bit counter clocked by a 110 Nano-Second Clock (110NSCLK) input and is enabled to count by a one MHZ Clock Enable input (CLK1MHZENB). INTTMR 25410 may thereby be clocked at a one MHZ rate to measure one microsecond intervals. Maximum time interval which may be measured by INTTMR 25410 is thereby 268.435 seconds.

As indicated in Fig. 254, INTTMR 25410 may be loaded from and read to JPD Bus 10142. In normal operation, the MEM 10112 location containing architectural time for a particular CS 10110 will be loaded with current architectural time at time of start up of that particular CS 10110. INTTMR 25410 will concurrently be loaded with all zeros. Thereafter, INTTMR 25410 will be clocked at one microsecond intervals. Periodically, when INTTMR 25410 overflows, architectural time stored in MEM 10112 will be accordingly updated. At any time, therefore, current architectural time may be determined, down to a one microsecond increment, by reading architectural time from the previous updated architectural time stored in MEM 10112 and elapsed interval since last update of architectural time from INTTMR 25410. In the event of a failure of CS 10110, architectural time in MEM 10112 and INTTMR 25410 may be saved in MEM 10112 by reading elapsed intervals since last architectural time update. When normal CS 10110 operation resumes, INTTMR 25410 may be reloaded with a count reflecting current architectural time. As indicated in Fig. 254, INTTMR 25410 is loaded from JPD Bus 10142 when INTTMR 25410 is enabled by a Load Enable input (LDE) provided from DP 10118.

Referring to EGGTMR 25412, certain CS 10110 Events, in particular Asychronous Events previously described with reference to EVENT 20284, are received or acknowledged by EVENT 20284 only at conclusion of State M1 of first microinstruction of an SOP. As certain CS 10110 microinstructions have long execution times, these Asynchronous Events may be subjected to an extended latency, or waiting, interval

Petitioner Apple Inc. - Exhibit 1004, p. 5268

before being serviced. EGGTMR 25412, in effect, measures latency time of pending Asychronous Events and provides an output to EVENT 20284 if a predetermined maximum latency time is exceeded.

As indicated in Fig. 254, EGGTMR 25412 is clocked by a 110 Nano-Second Clock input (110NSCLK). EGGTMR 25412 is initially set to zero by load input (LDZRO) at end of State MI of the first microinstruction of each SOP executed by CS 10110, or when specifically instructed so by DEVCMD subfield of a microinstruction word. EGGTMR 25412 is incremented when enabled by Clock Enable (CLKENB) input from EGGENB 25416. There are two conditions necessary for EGGTMR 25412 to be incremented. First condition is occurrence of an Asychronous Event, which is indicated by input ASYEVNT to EGGENB 25416 from EVENT 20284 Second condition is that 16 or more microseconds have elapsed since last increment of EGGTMR 25412. This interval is measured by an output from fourth bit of INTTMR 25410 which, as shown in Fig. 254, is connected to an input of EGGENB 25416. EGGTMR 25412 is a four bit counter and will thereby overflow and generate output OVRFLW to EVENT 20284 256 microseconds after beginning of an SOP if an Asychronous Event has occurred and if at least 16 microseconds have elapsed since start of that SOP EGGTMR 25412 thereby insures a maximum service latency of 256 microseconds for Asychronous Events.

e.e.e. Fetch Unit 10120 Interface to Execute Unit 10122

Finally, as previously described FU 10120's interface to EU 10122 is primarily comprised of EUDIS Bus 20206, for providing EUDPs to EU 10122's EUSITT, and FUINT 20298. Operation of EUSDT 20265 and EUDIS Bus 20206 has been previously described and will be described further in a following description of EU 10122. FUINT 20298 is primarily concerned with generating Event Requests for conditions signalled from EU 10122 so that these Events may be serviced. In this regard, FUINT 20298 is primarily comprised of gates receiving Event Requests from EU 10122 and providing corresponding outputs to EVENT 20284. Another interface function performed by FUINT 20298 is generation of a "transfer complete" signal generated by FU 10122 and provided to EU 10122 to assert that a EU 10122 result read from EU 10122 to FU 10120 has been received. This transfer complete signal indicates to EU 10122 that EU 10122's result register, described in a following description of EU 10122, is available for further use by EU 10122. This transfer complete signal is generated by an output of FUSITT 11012 as part of microinstruction sequences for transferring data from EU 10122 to FU 10120 or MEM 10112.

Having described structure and operation of FU 10120, including DESP 20210, MEMINT 20212, and FUCTL 20214, the structure and operation of EU 10122 will be described next below.

C. Execute Unit 10122 (Figs. 203, 255—268)

As previously described, EU 10122 is an arithmetic processor capable of executing integer, packed and unpacked decimal, and single and double precision floating point arithmetic operations. A primary function of EU 10122 is to relieve FU 10120 of certain arithmetic operations, thus enhancing efficiency of CS 10110.

Transfer of operands from MEM 10112 to EU 10122 is controlled by FU 10120, as is transfer of results of arithmetic operations from EU 10122 to FU 10120 or MEM 10112. In addition, EU 10122 operations are initiated by FU 10120 by EU 10122 Dispatch Pointers invited to EU 10122 by EUSDT 20266. EU 10122 Dispatch Pointers may initiate both arithmetic operations required for execution of SINs and certain EU 10122 operations assisting in handling of CS 10110 events. As previously described, EU 10122 Dispatch Pointers are translated into sequences of microinstructions for controlling EU 10122 by EU 10122's EUSITT which is similar in structure and operation to FUSITT 11012. As will be described further below, EU 10122 includes a command queue for receiving and storing sequences of EU 10122 Dispatch Pointers from FU 10120. In addition, EU 10122 includes a general register file, or scratch pad memory, similar to GRF 10354. EU 10122's general register file is utilized, in part, in EU 10122 Stack Mechanisms similar to FU 10120's SR's 10362.

Referring to Fig. 203, a partial block diagram of EU 10122 is shown. EU 10122's general structure and operation will be described first with reference to Fig. 203. Then EU 10122's structure and operation will be described in further detail with aid of subsequent figures which will be presented as required.

As indicated in Fig. 203, major elements of EU 10122 include Execute Unit Control Logic (EUCL) 20310, Execute Unit IO Buffer (EUIO) 20312, Multiplier Logic (MULT) 20314, Exponent Logic (EXP) 20316, Multiplier Control Logic (MULTCNTL) 20318, and Test and Interface Logic (TSTINT) 20320. EUCL 20310 receives Execute Unit Dispatch Pointers (EUDP's) from EUSDT 20266 and provides corresponding sequences of microinstructions to control operation of EU 10122.

EUIO 20312 receives operands, or data, from MEM 10112, translates those operands into certain formats most efficiently used by EU 10122. EUIO 20312 receives results of EU 10122's operations and translates those results into formats to be returned to MEM 10112 or FU 10120, and presents those results to MEM 10112 and FU 10120.

MULT 20314 and EXP 20316 are arithmetic units for performing arithmetic manipulations of EU 10122 operations. In particular, EXP 20316 performs operations with respect to exponent fields of single and double precision floating point operations. MULT 20314 performs arithmetic manipulations with respect to mantissa fields of single and double precision floating point operations, and arithmetic operations with regard to integer and packed decimal operations. MULTCNTL 20318 controls and coordinates operation of

Petitioner Apple Inc. - Exhibit 1004, p. 5269

MULT 20314 and EXP 20316 and prealignment and normalization of mantissa and exponent fields in floating point operations. Finally, TSTINT 20320 performs certain test operations with regard to EU 10122's operations, and is the interface between EU 10122 and FU 10120.

a. General Structure of EU 10122

1. Execute Unit I/O 20312

Referring first to EUIO 20312, EUIO 20312 includes Operand Buffer (OPB) 20322, Final Result Output Multiplexer (FROM) 20324, and Exponent Output Multiplexer (EXOM) 20326. OPB 20322 has first and second inputs connected, respectively, from MOD Bus 10144 and JPD Bus 10142. OPB 20322 has a first output connected to a first input of Multiplier Input Multiplexer (MULTIM) 20328 and MULT 20314. A second output of OPB 20322 is connected to first inputs of Inputs Selector A (INSELA) 20330 and Exponent Execute Unit General Register File Input Multiplexer (EXRM) 20332 in EXP 20316.

FROM 20324 has an output connected to JPD Bus 10142. A first input of FROM 20324 is connected from output of Multiplier Execute in General Register File Input Multiplexer (MULTRM) 20334 and MULT 20314. A second input of FROM 20324 is connected from output of Final Result Register (RFR) 20336 of MULT 20314. EXOM 20326 has an output connected to JPD Bus 10142. EXOM 20326 is a first input connected from output of Scale Register (SCALER) 20338 of EXP 20316. EXOM 20326 has second and third inputs connected from outputs of, respectively, Next Address Generator (NAG) 20340 and Command Queue (COMQ) 20342 of EUCL 20310.

2. Execute Unit Control Logic 20310

Referring to EUCL 20310, EUCL 20310 includes NAG 20340, COMQ 20342, Execute Unit S Interpreter Table (EUSITT) 20344, and Microinstruction Control Register and Decode Logic (mCRD) 20346. COMQ 20342 has an input connected from EUDIS Bus 20206 for receiving SDPs from EUSDT 20266. COMQ 20342 has, as described above, a first output connected to a third input of EXOM 20326, and has a second output connected to an input of NAG 20340. NAG 20340 has, as described above, a first output connected to second input of EXOM 20326. NAG 20340 has a second output connected to a first input of EUSITT 20344. As previously described, EUSITT 20344 corresponds to FUSITT 11012 and stores sequences of microinstructions for controlling operation of EU 10122 in response to EU 10122 Dispatch Pointers from FU 10120. EUSITT 20344 has a second input connected from JPD Bus 10142 and has an output connected to input of mCRD 20346. mCRD 20346 includes a register and logic for receiving and decoding microinstructions provided by EUSITT 20344. In addition to an input from EUSITT 20344, mCRD 20346 has first outputs providing decoded microinstruction control signals to all parts of EU 10122. mCRD 20346 also has a second output connected to a first input of Input Selecter B (INSELB) 20348 and EXP 20316.

3. Multiplexer Logic 20314

Referring to MULT 20314, MULT 20314 includes two parallel arithmetic operation paths for performing addition, subtraction, multiplication, and division operations on packed decimal numbers, integer numbers, and mantissa portions of single and double precision floating point numbers. MULT 20314 also includes a related portion of EU 10122's general register file, a memory for storing constants used in arithmetic operations, and certain input data selection circuits. That portion of EU 10122's GRF residing in MULT 20314 is comprised of Multiplier Register File (MULTRF) 20350. Output of MULTRF 20350 is connected to a second input of MULTIM 20328. A first input of MULTRF 20350 is connected from output of RFR 20336 and a second input of MULTRF 20350 is connected from output of MULTRM 20334. First and second inputs of MULTRM 20334 are in turn connected, respectively, from output of RFR 20336 and from output of Container Size Logic (CONSIZE) 20352 of TSTINT 20320.

MULTIM 20328 selects the data inputs to MULT 20314's arithmetic circuits and has, as previously described, first and second inputs connected respectively from first output of OPB 20322 and from output of MULTRF 20350. Output of MULTIM 20328 is connected through Multiplier (MULT) Bus 20354 to input of Multiplier Quotient Register (MQR) 20356 and to input of Nibble Shifter (NIBSHF) 20358. Another input to MQR 20356 and NIBSHF 20358 is provided by Constant Store (CONST) 20360. CONST 20360 is a memory for storing constant values used n MULT 20314 operations. Output of CONST 20360 is connected to MULT Bus 20354. MULT 20314's arithmetic circuits may thereby be provided with inputs from OPB 20322, MULTRF 20350, and CONST 20360.

MULT 20314's arithmetic circuitry is comprised of two, parallel arithmetic operation paths having, as common inputs, outputs of MULTIM 20328 and CONST 20360. Common termination of these parallel arithmetic operation paths is Final Register Shifter (FRS) 20362. A first arithmetic operation path is provided through NIBSHF 20358, whose input is connected from MULT Bus 20354. NIBSHF 20358's output is connected to a first input of FRS 20362 and a control input of NRBSHF 20358 is connected from an output of Multiplier Control Logic (MULTCNT) 20364 and MULTCNTL 20318.

MULT 20314's second arithmetic operation path is provided through MQR 20356. As described above, MQR 20356's input is connected from MULT Bus 20354. MQR 29356's output is connected to first and

Petitioner Apple Inc. - Exhibit 1004, p. 5270

second inputs of Times 1 And Times 2 Multiply Shifter (MULTSHFT12) 20366 and Times 4 And Times 8 Multiply Shifter (MULTSHFT48) 20368. Outputs of MULTSHFT12 and MULTSHFT8 are connected, respectively, to first and second inputs of First Multiplier Arithmetic and Logic Unit (MULTALU1) 20370. MULTALU1 20370's output is connected to input of Multiplier Working Register (MWR) 20372. Output of MWR 20372 is connected to a first input of Second Multiplier Arithmetic and Logic Unit (MULTALU2) 20374. A second input of MULTALU2 20374 is connected from output of RFR 20336. Output of MULTALU2 is connected to a second input of FRS 20362. As described above, first input of FRS 20362 is connected from output of NIBSHF 20368. Output of FRS 20362 is connected to input of RFR 20336.

As described above, output of RFR 20336 is connected to second input of MULTALU2 20374, to first input of MULTRF 20350, to first input of MULTRM 20334, and to second input of FROM 20324. Output of RFR 20336 is also connected to input of Leading Zero Detector (LZD) 20376 of MULTCNTL 20318, and to inputs of Exception Logic (ECPT) 20378, CONSIZE 20352, and TSTINT 20320.

4. Exponent Logic 20316

Referring to EXP 20316, as previously described EXP 20316 performs certain operations with respect to exponent fields of single and double precision floating point number in EU 10122 floating point operations. EXP 20316 includes a second portion of EU 10122's general register file, shown herein as Exponent Register File (EXPRF) 20380. Although indicated as individual register files, MULTRF 20350 and EXPRF 20380 comprise, as in GRF 10354, a unitary register file structure with common, parallel addressing of corresponding registers therein.

Output of EXPRF 20380 is connected to a second input of INSELA 20330. A first input of EXPRF 20380 is connected from output of EXRM 20332. As previously described, a first input of EXRM 20332 is connected from second output of OPB 20322 through EXPQ Bus 20325. A second input of EXRM 20332 is connected from output Scale Register (SCALER) 20338. A second input of EXPRF 20380 is connected from output of Sign Logic (SIGN) 20382. Input of SIGN 20382 is connected from second output of SCALER 20338.

INSELA 20330, INSELB 20348, Exponent ALU (EXPALU) 20384 and SCALER 20338 comprise EXP 20316's arithmetic circuitry for manipulating exponent fields of floating point numbers. INSELA 20330 and INSELB 20348 select, respectively, first and second inputs to EXPALU 20384. As previously described, a first input of INSELA 20330 is connected from second output of OPB 20322 through EXPQ Bus 20325. Second input of INSELA 20330 is connected from output of EXPRF 20380. Output of INSELA 20330 is connected to first input of EXPALU 20384. First input of INSELB 20348 is, as previously described, connected from a second output of mCRD 20346. Second input of INSELB 20348 is connected from output of OPB 20322 through EXPQ Bus 20325. Third input of INSELB 20348 is connected from output of SCALER 20338 and fourth input of INSELB 20348 is connected from output of LZD 20376. Output of INSELB 20348 is connected to second input of EXPALU 20348. Output of EXPALU 20348 is connected to input of SCALER 20338.

As previously described, second output of SCALER 20338 is connected with input of SIGN 20382 and first output is connected to second input of EXRM 20332 and to third input of INSELB 20348. First output of SCALER 20338 is also connected to EXPQ Bus 20325, to first input of EXOM 20326, and to a second input of MULTCNT 20364.

5. Multiplier Control 20318

As previously described, MULTCNTL 20318 provides certain control signals and information for controlling and coordinating operation of EXP 20316 and MULT 20314 in performing arithmetic operations on floating point numbers. MULTCNTL 20318 includes LZD 20376 and MULTCNT 20364. Input of LZD 20376 is connected from output of RFR 20336 through FR Bus 20337. Output of LZD 20376 are connected to a second input of MULTCNT 20364 and to fourth input of INSELB 20348. A second input of MULTCNT 20364 is connected from output of SCALER 20338. As previously described, control output of MULTCNT 20364 is connected to control inputs of NIBSHF 20358.

6. Test and Interface Logic 20320

Finally, TSTINT 20320 includes ECPT 20378, CONSIZE 20352, and Testing Condition Logic (TSTCON) 20386. Input of ECPT 20378 and first input of CONSIZE 20352 are connected from output of RFR 20336 through FR Bus 20337. A second input of CONSIZE 20352 is connected from LENGTH Bus 20226. An output of CONSIZE 20352 is connected, together with other inputs from EU 10122 (not shown for clarity of presentation) to TSTCON 20386. Output of TSTCON 20386 (not shown for clarity of presentation) are connected to NAG 20340. TSTCON 20386 and ECPT 20378 have outputs to and inputs from FU 10120's FUINT 20298.

Having described the overall structure of EU 10122 above, operation of EU 10122 will be described next below with aid of further diagrams which will be introduced as required. Finally, operation of TSTINT 20320 will be described, including a description of the detailed control signal interface between EU 10122 and FU 10120 through TSTINT 20320 and FUINT 20298. In addition to defining the interface between EU 10122 and FU 10120, certain features of EU 10122 operation will be described wherein those operations are executed

Petitioner Apple Inc. - Exhibit 1004, p. 5271

in cooperation with MEM 10112 and FU 10120. For example, EU 10122's Stack Mechanisms, comprising in part portions of MULTRF 20350 and EXPRF 20380, resides partly in MEM 10112 so that operation of EU 10122's Stack Mechanisms requires cooperative operations by EU 10122, MEM 10112 and FU 10120.

b. Execute Unit 10122 Operation (Fig. 255)

1. Execute Unit Control Logic 20310 (Fig. 255)

Referring to Fig. 255, a more detailed block diagram of EUCL 20310 is shown. As described above, EUCL 20310 receives EU 10122 Dispatch Pointers through EUDIS Bus 20206 from EUSDT 20266 and FUCTL 20214. EU 10122 Dispatch Pointers select certain EU 10122 microinstruction sequences for executing EU 10122 arithmetic operations as required to execute user's programs, that is SOPs, and to assist in handling JP 10114 Events. As described above, major elements of EUCL 20310 include COMQ 20342, EUSITT 20344, mCRD 20346, and NAG 20340.

a.a. Command Queue 20342

Inputs of COMQ 20342 are connected from EUDIS Bus 20206 to receive and store EU 10122 Dispatch pointers provided from EUSDT 20266. Each such EU 10122 Dispatch Pointer is comprised of two information fields. A first information field contains a 10 bit starting address of a corresponding sequence of microinstructions residing in EUSITT 20344. Second field of each EU 10122 Dispatch Pointer is a 6 bit field containing certain control information, such as information identifying data format of corresponding operands to be operated upon. In this case unit dispatch pointer control field bits specify whether operands to be operated upon comprise signed or unsigned integer, packed or unpacked decimal, or single or double precision floating point numbers.

COMQ 20342 is comprised of two one word wide by two word deep register files. A first of these register fields is comprised of SOP Command Queue Control Store (CQCS) 25510 and SOP Command Queue Address Store (CQAS) 25512. Together, CQCS 25510 and CQAS 25512 comprise a one word wide by two word deep register file for receiving and storing EU 10122 Dispatch Pointers corresponding to SOPs, that is Dispatch Pointers for initiating EU 10122 operations directly concerned with executing a user's program. Address fields of these SOPs are received in CQAS 25512, while control fields are received and stored in CQCS 25510. COMQ 20342 is thereby capable of receiving and storing up to two sequential EU 10122 Dispatch Pointers corresponding to user program SOPs These SOP derived Dispatch Pointers are executed in the order received from FU 10120. EU 10122 is thereby capable of receiving and storing one currently executing SOP Dispatch Pointer and one pending SOP Dispatch Pointer. Further SOP Dispatch Pointers may be read into COMQ 20342 as previous SOPs are executed.

b.b. Command Queue Event Control Store 25514 and Command Queue Event Address Control Store 25516

Command Queue Event Control Store (CQCE) 25514 and command Queue Event Address Control Store (CQAE) 25516 are similar in function and operation to, respectively, CQCS 25510 ad CQAS 25512. CQCE 25514 and CQAE 25516 receive and store, however, EU 10122 Dispatch Pointers initiating EU 10122 operations requested by FU 10120 as required to handle JP 10114 Events. Again, CQCE 25514 and CQAE 25516 comprise a one word wide by two word deep register file. CQAE 25516 receives and stores address fields of Event Dispatch Pointers, while CQCE 25514 receives and stores corresponding control fields of Event Dispatch Pointers. Again, COMQ 20342 is capable of receiving and storing up to two sequential Event Dispatch Pointers at a time.

As indicated in Fig. 255, outputs of CQAS 25512 and CQAE 25516, that is address fields of EU 10122 Dispatch Pointers are provided as inputs to Select Case Multiplexer (SCASE) 25518 and Starting Address Select Multiplexer (SAS) 25520 and NAG 20340, which will be described further below. Control field outputs of CQCS 25510 and CQCE 25514 are provided as inputs to OPB 20322, described further below.

c.c. Execute Unit S-Interpreter Table 20344

Referring to EUSITT 20344, as described above EUSITT 20344 is a memory for storing sequences of microinstructions for controlling operation of EU 10122 in response to EU 10122 Dispatch Pointers received from FU 10120. These microinstruction sequences may, in general, direct operation of EU 10122 to execute arithmetic operations in response to SOPs of user's programs, or aid direct execution of EU 10122 operations required to service JP 10114 Events. EUSITT 20344 may be, for example, a 60 bit wide by 1,280 word long memory structured as pages of 128 words per page. A portion of EUSITT 20344's pages may be contained in Read Only Memory, for example for storing sequence of microinstructions for handling JP 10114 Events. Remaining portions of EUSITT 20344 may be constructed of Random Access Memory, for example for storing sequences of microinstructions for executing EU 10122 operations in response to user program SOPs. This structure allows EU 10122 microinstruction sequences concerned with operation of JP 10114's internal mechanisms, for example handling of JP 10114 Events, to be effectively permanently

Petitioner Apple Inc. - Exhibit 1004, p. 5272

stored in EUSITT 20344. That portion of EUSITT 20344 constructed of Random Access Memory may be used to store sequences of microinstructions for executing SOPs. These Random Access Memories may be used as writable control store to allow sequences of microinstructions for executing SOPs of one or more S-Languages currently being utilized by CS 10110 to be written into EUSITT 20344 from MEM 10112 as required.

As previously described, EUSITT 20344's second input is a Data (DATA) input connected from JPD Bus 10142. EUSITT 20344's data input is utilized to write sequences of microinstructions into EUSITT 20344 from MEM 10112 through JPD Bus 10142. EUSITT 20344's first input is an address (ADR) input connected from output of Address Driver (ADRD) 25522 and NAG 20340. Address inputs provided by ADRD 25522 select word locations within EUSITT 20344 for writing of microinstructions into EUSITT 20344, or for reading of microinstructions from EUSITT 20344 to mCRD 20346 to control operation of EU 10122. Generation of these address inputs to EUSITT 20344 by NAG 20340 will be described further below.

d.d. Microcode Control Decode Register 20346

Output of EUSITT 20344 is connected to input of mCRD 20346. As previously described, mCRD 20346 is a register for receiving microinstructions from EUSITT 20344, and decoding logic for decoding those microinstructions and providing corresponding control signals to EU 10122. As indicated in Fig. 255, Diagnostic processor Micro-Program Register (DPmR) 25524 is a 60 bit register connected in parallel with output of EUSITT 20344 to input of mCRD 20346. DPmR 25524 may be loaded with 60 bit microinstructions by DP 10118. Diagnostic microinstructions may thereby be provided directly to input of mCRD 20346 to provide direct microinstruction by microinstruction control of EU 10122.

Outputs of mCRD 20346 are provided, in general, to all portions of EU 10122 to control detailed operations of EU 10122. Certain outputs of mCRD 20346 are connected to inputs of Next Address Source Select Multiplexer (NASS) 25526 and Long Branch Page Address Gate (LBPAG) 25528 and NAG 20340. As will be described further below, these outputs of mCRD 23046 are used in generating address inputs to EUSITT 20344 when particular microinstructions sequences call for Jumps or Long Branches to other microinstruction sequences. Outputs of mCRD 20346 are also connected in parallel to inputs of Execution Unit Micro-Instruction Parity Check Logic (EUmIPC) 25530. EUmIPC 25530 checks parity of all microinstruction outputs of mCRD 20346 to detected errors in mCRD 20346's outputs.

e.e. Next Address Generator 20340

As described above, read and write addresses to EUSITT 20344 provided by NAG 20340 through ADRD 25522. Address inputs to ADRD 25522 are provided from either NASS 25526 or Diagnostic Processor Address Register (DPAR) 25532. In normal operation, address inputs to EUSITT 20344 are provided from NASS 25526 as will be described momentarily. DP 10118, however, may load EUSITT 20344 addresses into DPAR 25532. These addresses may then be read from DPAR 25532 through ADRD 25522 to individually select address locations within EUSITT 20344. DPAR 25532 may be utilized, in particular, to provide addresses to allow stepping through of EU 10122 microinstruction sequences microinstruction by microinstruction.

As described above, NASS 25526 is a multiplexer having inputs from three NAG 20340 address sources. NASS 25526's first address input is from Jump (JMP) output of mCRD 20346 and LBPAG 25528. These address inputs are utilized, in part, when a current microinstruction calls for a Jump or Long Branch to another microinstruction or microinstruction sequence. Second address source is provided from SAS 25520 and, in general, is comprised of starting addresses of microinstruction sequences. SAS 25520 is a multiplexer having a first input from COAS 25512 and COAE 25516, that is starting addresses of microinstruction sequences corresponding to SOPs or for servicing JP 10114 Events. A second SAS 25520 input is provided from Sub-routine Return Address Stack (SUBRA) 25534. In general, and as will be described further below, SUBRA 25534 operates as a stack mechanism for storing current microinstruction addresses of interrupted microinstruction sequences. These stored addresses may subsequently be utilized to resume execution of those interrupted microinstruction sequences. Third address source to NASS 25526 is provided from Sequential and Case Address Generator (SCAG) 25536. In general, SCAG 25536 generates address to select sequential microinstructions within particular microinstruction sequences. SCAG 25536 also generates microinstruction address for microinstruction Case operations. As indicated in Fig. 255, outputs of SCAG 25536 and of SAS 25520 are bused together to comprise a single NASS 25526 input. Selection between outputs of SCAG 25536 and SAS 25520 are provided by control inputs (not shown for clarity of presentation) to SCAG 25536 and SAS 25520. Selection between NASS 25526's address inputs is controlled by Next Address Source Select Control Logic (NASSC) 25538, which provides control inputs to NASS 25526. NASSC 25538 is effectively a multiplexer receiving control inputs from TSTCON 20386 and TSTINT 20320. As will be described further below, TSTCON 20386 monitors certain operating conditions or states within EU 10122 and provides corresponding inputs to NASSC 25538. NASSC 25538 effectively decodes these control inputs from TSTCON 20386 to provide selection control input to NASS 25526.

Having described overall structure and operation of NAG 20340, operation of NAG 20340 will be

Petitioner Apple Inc. - Exhibit 1004, p. 5273

described in further detail next below.

Referring first to NASS 25526's address inputs provided from JMP output of mCRD 20346 and LBPAG 25528, this address source is provided to allow selection of a next microinstruction by a current microinstruction. JMP output of mCRD 20346 allows a current microinstruction to direct a Jump to another microinstruction within the same page of EUSITT 20344. NASS 25526's input through LBPAG 25528 is provided from another portion of mCRD 20346's output specifying pages within EUSITT 20344. This input through LBPAG 25528 allows execution of Long Branch operations, that is jumps from a microinstruction in one page of EUSITT 20344 to a microinstruction in another page. In addition, NASS 25526's input from JMP output of mCRD 20346 and through LBPAG 25528 is utilized to execute an Idle, or Standby, routine when EU 10122 is not currently executing a microinstruction sequence requested by FU 10120. In this case, Idle routine directs TSTCON 20386 to monitor EU 10122 Dispatch Pointer inputs to EU 10122 from FU 10120. If no EU 10122 Dispatch Pointers are present in COMQ 20342, or none are pending, TSTCON 20386 will direct NASSC 25538 to provide control inputs to NASS 25526 to select NASS 25526's input from mCRD 20346 and LBPAG 25528. Idle routine will continually test for EU 10122 Dispatch pointer inputs until such a Dispatch Pointer is received into COMQ 20342. At this time, TSTCON 20386 will detect the pending Dispatch Pointer and direct NASS 25538 to provide control outputs to NASS 25526 to select NASS 25526's input from, in general, SAS 25520. TSTCOND 20386 and NASSC 25538 will also direct NASS 25526 to select inputs from SAS 25520 upon return from a called microinstruction to a previously interrupted microinstruction sequence.

As described above, SAS 25520 receives starting addresses from COMQ 20342 and from SUBRA 25534. SAS 25520 will select the output of CQAS 25512 or of CQAE 25516 as the input to NASS 25526 when a new microinstruction sequence is to be initiated to execute a user's program SOP or to service a JP 10114 Event. SAS 25520 will select an address output of SUBRA 25534 upon return from a called sub-routine to a previously executing but interrupted sub-routine. SUBRA 25534, as described above, is effectively a stack mechanism for storing addresses of currently executing microinstructions when those microinstruction sequences are interrupted. SUBRA 25534 is an 11 bit wide by 8 word deep register with certain registers dedicated for use in stacking Event Handling microinstruction sequences. Other portions of SUBRA 25534 are utilized for stacking of microinstruction sequences for executing SOPs, that is for stacking microinstruction sequences wherein a first microinstruction sequence calls for a second microinstruction sequence. SUBRA 25534 is not operated as a first-in-first out stack, but as a random access memory wherein address inputs selecting registers and SUBRA 25534 are provided by microinstruction control outputs of mCRD 20346. Operations of SUBRA 25534 as a stack mechanism is thereby controlled by the microinstruction sequences stored in EUSITT 20344. As indicated in Fig. 255, addresses of current microinstructions of interrupted microinstruction sequences are provided to data input of SUBRA 25534 from output of SCAG 25536, which will be described next below.

As described above, SCAG 25536 generates sequential addresses to select sequential microinstructions within microinstruction sequences and to generate microinstruction addresses for Case operations. SCAG 25536 includes Next Address Register (NXTR) 25540, Next Address Arithmetic and Logic Unit (NAALU) 25542, and SCASE 25518. NAALU 25542 is a 12 bit arithmetic and logic unit. A first eleven bit input of NAALU 25542 is connected from output of ADRD 25522 and is thereby current address provided to EUSITT 20344. A second four bit input to NAALU 25542 is provided from output of SCASE 25518. During sequential execution of a microinstruction sequence, output of SCASE 25518 is binary zeros and carry input of NAALU is forced to 1. Output of NAALU 25542 will thereby be and address one greater than the current microinstruction address provided to EUSITT 20344 and will thereby be the address of the next sequential microinstruction. As indicated in Fig. 255, SCASE 25518 receives an input from output of SCALER 20338. This input is utilized during Case operations and allows a data sensitive number to be selected as SCASE 25518's output into second input of NAALU 25542. SCASE 25518's input from SCALER 20338 thereby allows NAG 20340 to perform microinstruction Case operations wherein Case Values are determined by the contents of SCALER 20338.

Next address outputs of NAALU 25542 are loaded into NXTR 25540, which is comprised of tri-state output registers. Next address outputs of NXTR 25540 are connected, in common with outputs of SAS 25520, to second input of NASS 25526 as described above. During normal execution of microinstruction sequences, therefore, SCAG 25536 will, through NASS 25526 and ADRD 25522, select sequential microinstructions from EUSITT 20344. SCAG 25536 may also, as just described, provide next microinstruction addresses in microinstruction Case operations.

In summary, NAG 20340 is capable of performing all usual microinstruction sequence addressing operations. For example, NAG 20340 allows selection of next microinstructions by current microinstructions, either for Jump operations or Long Branch operations, through NASS 25526's input from mCRD 20346's JMP or through LBPAG 25528. NAG 20340 may provide microinstruction sequence starting addresses through COMQ 20342 and SAS 25520, or may provide return addresses to interrupted and stacked microinstruction sequences through SUBRA 25534 and SAS 25520. NAG 20340 may sequentially address microinstructions of a particular microinstruction sequence through operation of SCAG 25536, or may perform mciroinstruction Case operations through SCAG 25536.

Petitioner Apple Inc. - Exhibit 1004, p. 5274

2. Operand Buffer 20322

Having described structure and operation of EUCL 20310, structure and operation of OPB 20322 will be described next below. As previously described, OPB 20322 receives operands, that is data, from MEM 10112 and FU 10120 through MOD Bus 10144 and JPD Bus 10142. OPB 20322 may then perform certain operand format translations to provide data to MULT 20314 and EXP 20316 in the formats most efficiently utilized by MULT 20314 and EXP 20316. As previously described, EU 10122 may perform arithmetic operations on integer, packed and unpacked decimal, and single or double precision floating point numbers.

In summary, therefore, OPB 20322 is capable of accepting integer, single and double precision floating point, and packed and unpacked decimal operands from MEM 10112 and FU 10120 and providing appropriate fields of those operands to MULT 20314 and EXP 20316 in the formats most efficiently utilized by MULT 20314 and EXP 20316. In doing so, OPB 20322 extracts exponent and mantissa fields from single and double precision floating point operands to provide exponent and mantissa fields of these operands to, respectively, EXP 20316 and MULT 20314, and also unpacks, or converts, unpacked decimal operands to packed decimal operands most efficiently utilized by MULT 20314.

Having described structure and operation of OPB 20322, structure and operation of MULT 20314 will be described next below.

3. Multiplier 20314 (Figs. 257, 258)

MULT 20314, as previously described, performs addition, subtraction, multiplication, and division operations on mantissa fields of single and double precision floating point operands, integer operands, and decimal operands. As described above with reference to OPB 20322, OPB 20322 converts unpacked decimal operands to packed decimal operands to be operated upon by MULT 20314. MULT 20314 is thereby effectively capable of performing all arithmetic operations on unpacked decimal operands.

a.a. Multiplier 20314 Data Paths and Memory (Fig. 257)

Referring to Fig. 257, a more detailed block diagram of MULT 20314's data paths and memory is shown. As previously described, major elements of MULT 20314 include memory elements comprised of MULTRF 20350 and CONST 20360, operand input and result output multiplexing logic including MULTIM 20328 and MULTRM 20334, and arithmetic operation logic. MULT 20314's operand input and result output multiplexing logic and memory elements will be described first, followed by description of MULT 20314's arithmetic operation logic.

As previously described, input data, including operands, is provided to MULT 20314's arithmetic operation logic through MULTIN Bus 20354. MULTIN Bus 20354 may be provided with data from three sources. A first source is CONST 20360 which is a 512 word by 32 bit wide Read Only Memory. CONST 20360 is utilized to store constants used in arithmetic operations. In particular, CONST 20360 stores zone fields for unpacked decimal, that is ASCI character, operands. As previously described, unpacked decimal operands are received by OPB 20322 and converted to packed decimal operands for more efficient utilization by MULT 20314. As such, final result outputs generated by MULT 20314 from such operands are in packed decimal format. As will be described below, MULT 20314 may be utilized to convert these packed decimal results into unpacked decimal results by insertion of zone fields. As indicated in Fig. 257, address inputs are provided to CONST 20360 from EXPQ Bus 20325 and from output of mCRD 20346. Selection between these address inputs is provided through CONST Address Multiplexer (CONSTAM) 25710. CONST 20360 addresses will, in general, be provided from EUCL 20310 but alternately may be provided from EXPQ Bus 20325 for special operations.

Operand data is provided to MULTIN Bus 20354 through MULTIM 20328, which is a dual input, 64 bit multiplexer. A first input of MULTIM 20328 is provided from OPQ Bus 20323 and is comprised of operand information provided from OPB 20322. OPQ Bus 20323 is a 56 bit wide bus and operand data appearing thereon may be comprised of 32 bit integer operands; 32 bit packed decimal operands, either provided directly from OPB 20322 or as a result of OPB 20322's conversion of an unpacked decimal to a packed decimal operand; 24 bit single precision operand mantissa fields; or 56 bit double precision floating point operand mantissa fields. As previously described, certain OPQ Bus 20323 may be zero or sign extension filled, depending upon the particular operand.

Second input of MULTIM 20328 is provided from MULTRF 20350. MULTRF 20350 is a 16 word by 64 bit wide random access memory. As indicated in Figs. 203 and 257, MULTRF 20350 is connected between output of RFR 20336, through FR Bus 20337, and to input of MULT 20314's arithmetic operation logic through MULTIM 20328 and MULTIN Bus 20354. MULTRF 20350 may therefore be utilized as a scratch pad memory for storing intermediate results of arithmetic operations, including reiterative arithmetic operations. In addition, a portion of MULTRF 20350 is utilized, as in GRF 10354, as an EU 10122 Stack Mechanism similar to MIS 10368 and MOS 10370 in FU 10120. Operation of EU 10122 Stack Mechanism will be described in a following descripion of EU 10122's interfaces to MEM 10112 and FU 10120. Address Inputs (ADR) of MULTRF 20350 are provided from Multiplier Register File Address Multiplexer (MULTRFAM) 25712.

Petitioner Apple Inc. - Exhibit 1004, p. 5275

MULTRFAM 25712 is a dual four bit multiplexer comprised, for example, of SN74S258s. In addition to address inputs to MULTRF 20350, MULTRFAM 25712 provides address inputs to EXPRF 20380. As previously described, MULTRF 20350 and EXPRF 20380 together comprise an EU 10122 general register file similar to GRF 10354 and FU 10120. As such, MULTRF 20350 and EXPRF 20380 are addressed in parallel to read and write parallel entries from and to MULTRF 20350 and EXPRF 20380. Address inputs to MULTRFAM 25712 are provided, first, from outputs of mCRD 20346, thus providing microinstruction control of addressing of MULTRF 20350 and EXPRF 20380. Second address input to MULTRFAM 25712 is provided from output of Multiplier Register File Address Counter (MULTRFAC) 25714.

MULTRFAC 25714 is a four bit counter and is used to generate sequential addresses to MULTRF 20350 and EXPRF 20380. Initial addresses are loaded into MULTRFAC 25714 from Multiplier Register File Address Counter Multiplexer (MULTRFACM) 25716. MULTRFACM 25716 is a dual four bit multiplexer. Inputs to MULTRFACM 25716 are provided, first, from outputs of mCRD 20346. This input allows microinstruction selection of an initial address to be loaded into MULTRFAC 25714 to be subsequently used and generating sequential MULTRF 20350 and EXPRF 20380 addresses. Second address input to MULTRFACM 25716 is provided from OPQ Bus 20323. MULTRFACM 25716's input from OPQ Bus 20323 allows a single address, or a starting address of a sequence of addresses, to be selected through JPD Bus 10142 or MOD Bus 10144, for example from MEM 10112 or FU 10120.

Intermediate and final result outputs of MULT 20314 arithmetic logic are provided to data inputs of MULTRF 20350 directly from FR Bus 20337 and from MULTRM 20334. Inputs to MULTRM 20334, in turn, are provided from FR Bus 20337 and from output of CONSIZE 20352 and TSTINT 20320.

FR Bus 20337 is a 64 bit bus connected from 64 bit output of RFR 20336 and carries final and intermediate results of MULT 20314 arithmetic operations. As will become apparent in a following description of MULT 20314 arithmetic operation logic, RFR 20336 output, and thus FR Bus 20337, are 64 bits wide. Sixty-four bits are provided to insure retention of all significant data bits of certain MULT 20314 arithmetic operation intermediate results, in particular operations involving double precision floating point 64 bit mantissa fields. In addition, as will be described momentarily and has been previously stated, MULT 20314 may convert a final result in packed decimal format into a final result in unpacked decimal format. In this operation, a single 32 bit, or one word, packed decimal result is converted into a 64 bit, or two word, unpacked decimal format by insertion of zone fields.

As described above, two parallel data paths are provided to transfer information from FR Bus 20337 into MULTRF 20350. First path is directly from FR Bus 20337 and second path is through Unpacked Decimal Multiplexer (UPDM) 25718 of MULTRM 20334. Direct path is utilized for thirty-two bits of information comprising bits 0 to 23 and bits 56 to 63 of FR Bus 20337. Data path through UPDM 25718 may comprise either bits 24 to 55 of FR Bus 20337, which are connected into a first input of UPDM 25718, or bits 40 through 55 which are connected to a second input of UPDM 25718. Single precision floating point numbers are 32 bit numbers plus two or more guard bits and are thus written into MULTRF 20350 through bits 0 to 23 of the direct path into MULTRF 20350 and through first input (bits 24 to 55) of UPDM 25718. Double precision floating point numbers are 5 bits wide, plus guard bits, and thus utilize the direct path into MULTRF 20350 and the path through first input of UPDM 25718. Bits 56 to 63 of direct path are utilized for guard bits of double precision floating point numbers. Both integer and packed decimal numbers utilize bits 24 through 55 of FR Bus 20337, and are thus written into MULTRF 20350 through first input of UPDM 25718. As previously described, bits 0 to 23 of these operands are filled by sign extension.

a.a.a. Container Size Check

As stated above, MULTRM 20334 has an Input from CONSIZE 20352. As will be described below with reference to TSTINT 20320, CONSIZE 20352 performs a "container size" check upon each store back of results from EU 10122 to MEM 10112. CONSIZE 20352 compares the number of significant bits in a result to be stored back to the logical descriptor describing the MEM 10112 address space that result is to be written into. Where reiterative write operations to MEM 10112 are required to transfer a result into MEM 10112, that is a string transfer, container size information may read from CONSIZE 20352 through Container Size Driver (CONSIZED) 25720 and MULTRM 20334 and written into MULTRF 20350. This allows EU 10122, using container size information stored in MULTRM 20350, to perform continuous container size checking during a string transfer of result from EU 10122 to MEM 10112. In addition, as will be described momentarily, container size information may be read from CONSIZE 20352 to JPD Bus 10144.

b.b.b. Final Result Output Multiplexer 20324

Referring finally to FROM 20324, as previously described FROM 20324 is utilized to transfer, in general, results of EU 10122 arithmetic operations onto JPD Bus 10142 for transfer to MEM 10112 or FU 10120. As indicated in Fig. 257, FROM 20324 is comprised of 24 bit Final Result Bus Driver (FRBD) 25722 and Result Bus Driver (RBR) 25724. Input of FRBD 25722 is connected from FR Bus 20337 and allows data appearing thereon to be transferred onto JPD Bus 10142. In particular, FRBD 25722 is utilized to transfer 24 bit mantissa fields of single precision floating point results onto JPD Bus 10142 in parallel with a corresponding exponent field from EXP 20316. RBR 25724 input is connected from RSLT Bus 20388 to allow

Petitioner Apple Inc. - Exhibit 1004, p. 5276

output of UPDM 25718 to be transferred onto JPD Bus 10142. RBR 25724, RSLT Bus 20388, and UPDM 25718 are used, in general, to transfer final results of EU 10122 operations from output of MULT 20314 onto JPD Bus 10142. Final results transferred by this data path include integer, packed and unpacked decimal results, and mantissa fields of double precision floating point results. Both unpacked decimal numbers and mantissa fields of double precision floating point numbers are comprised of two 32 bit words and are thus transferred onto JPD Bus 10142 in two sequential transfer operations.

Having described structure and operation of MULT 20314's memory elements and input and output circuitry, MULT 20314's arithmetic operation logic will be described next below.

4. Test and Interface Logic 20320 (Figs. 260—268)

As previously described, TSTINT 20320 includes CONSIZE 20352, ECPT 20328, TSTCOND 20384, and INTRPT 20388. CONSIZE 20352, as previously described, performs "container size" check operations when results of EU 10122 operations are to be written into MEM 10112. That is, CONSIZE 20352 compares size or number of significant bits, of an EU 10122 result to the capacity, or container size, of the MEM 10112 location that EU 10122 result is to be written into. As indicated, in Fig. 203, CONSIZE 20352 receives a first input, that is the results of EU 10122 operations, from FR Bus 20337. A second input of CONSIZE 20352 is connected to LENGTH Bus 20226 to receive length field of logical descriptors identifying MEM 10112 address space into which those EU 10122 results are to be written. CONSIZE 20352 includes logic circuitry, for example a combination of Read Only Memory and Field Programmable Logic Arrays, for examining EU 10122 operation results appearing on FR Bus 20337 and determining the number of bits of data in those results. CONSIZE 20352 compares EU 10122 result size to logical descriptor length field and, in particular, if result size exceeds logical descriptor length, provides an alarm output to ECPT 20328, described below.

TSTCOND 20384, previously described and which will be described further below, is an interface circuit between FU 10120 and EU 10122. TSTCOND 20384 allows FU 10120 to specify and examine results of certain test operations performed by EU 10122 with respect to EU 10122 operations.

ECPT 20328 monitors certain EU 10122 operations and provides outputs indicating when certain "exceptions" have occurred. These exceptions include attempted divisions by zero, floating point exponent underflow or overflow, and integer container size fault.

INTRPT 20388 is again an interface between EU 10122 and FU 10120 allowing FU 10120 to interrupt EU 10122 operations. INTRPT 20388 allows FU 10120 to direct EU 10122 to execute certain operations to aid in handling of certain FU 10120 events previously described.

Operation of CONSIZE 20352, ECPT 20328, TSTCOND 20384, INTRPT 20388, and other features of EU 10122's interface with FU 10120 will be described further below in the following description of operation of that interface and of operation of certain EU 10122 internal mechanisms, such as FU 10120 Stack Mechanisms.

a.a. FU 10120/EU 10122 Interface

As previously described, EU 10122 and FU 10120 are asychronous processors, each operating under its own microcode control. EU 10122 and FU 10120 operate simultaneously and independently of each other but are coupled, and their operations coordinated, by interface signals described below. Should EU 10122 not be able to respond immediately to a request from FU 10120, FU 10120 will idle until EU 10122 becomes available; conversely, should EU 10122 not receive, or have present, operands or a request for operations from FU 10120, EU 10122 will remain in idle state until operands and requests for operations are received from FU 10120.

In normal operation, EU 10122 manipulates operands under control of FU 10120, which in turn is under control of SOPs of a user's program. When FU 10120 requires arithmetic or logical manipulation of an operand, FU 10120 dispatches a command, that is an Execute Unit Dispatch Pointer (EUDP) to EU 10122. As previously described, an EUDP is basically an initial address into EUSITT 20344. An EUDP identifies starting location of a EU 10122 microinstruction sequence performing the required operation upon operands. Operands are fetched from MEM 10112 under FU 10120 control, as previously described, and are transferred into OPB 20322. Those operands are then called from OPB 20322 by EU 10122 and transferred into MULT 20314 and EXP 20316 as previously described. After the required operation is completed, FU 10120 is notified that a result is ready. At this point, FU 10120 may check certain test conditions, for example through TSTCOND 20384, such as whether an integer or decimal carry bit is set or whether a mantissa sign bit is set or reset. This test operation is utilized by FU 10120 for conditional branching and synchronization of FU 10120 and EU 10122 operations. Exception checking, by ECPT 20328, is also performed at this time. Exception checking determines, for example, whether division by zero was attempted or if a container size fault has occurred. In general, FU 10120 is not informed of exception errors until FU 10120 requests exception checking. After results are transferred into FU 10120 or MEM 10112 by EU 10122, EU 10122 goes to idle operation until a next operation is requested by FU 10120.

Having briefly described overall interface operation between FU 10120 and EU 10122, operation of that interface, referred to as handshaking, will be described in greater detail next below. In general, handshaking operation between EU 10122 and FU 10120 during normal operation may be regarded as following into six operations. These operations may include, for example, loading of COMQ 20342, loading of OPB 20322, storeback or transfer of results from EU 10122 to FU 10120 or MEM 10112, check of test

Petitioner Apple Inc. - Exhibit 1004, p. 5277

conditions, exception checking, and EU 10122 idle operation. Handshaking between FU 10120 and EU 10122 will be described below for each of these classes of operation, in the order just referred to.

a.a.a. Loading of Command Queue 20342 (Fig. 260)

Referring to Fig. 260, a schematic representation of EU 10122's interface with FU 10120 for purposes of loading COMQ 20342 as shown. During normal SOP directed JP 10114 operation, 8 bit operation (OP) codes are parsed from the instruction stream, as previously described, and concatenated with dialect information to address EUSDT 20266 also as previously described. EUSDT 20266 provides corresponding addresses, that is EUDPs, to EUSITT 20344.

Dialect information specifies the S-Language currently being executed and, consequently, the group of microinstruction sequences available in EUSITT 20344 for that S-Language. As previously described, FU 10120 may specify four S-Language dialects with up to 256 EU 10122 microinstruction sequences per dialect, or 8 dialects with up to 128 microinstruction sequences per dialect.

EUDPs provided by EUSDT 20266 are comprised of a 9 bit address field, a 2 bit operand information field, and a 1 bit flag field, as previously described. Address field is starting address of a microinstruction sequence in EUSITT 20344 and EU 10122 will perform the operation directed by that microinstruction sequence. EUSITT 20344 requires 11 bits of address field and the 9 bit address field of EUDPs are mapped into an 11 bit address field by left justification and zero filling.

FU 10120 may also dispatch, or select, any EU 10122 microinstruction controlled operation from JPD Bus 10142. Such EUDPs are provided from JPD Bus 10142 to data input of EUSITT 20344 and passed directly through to mCRD 20346. Before a EUDP may be provided from JPD Bus 10142, however, FU 10120 provides a check operation comparing that EUDP to a list of legal, or allowed, EUSITT 20344 addresses stored in MEM 10112. A fault will be indicated if an EUDP provided through JPD Bus 10142 is not a legal EUSITT 20344 address. Alternately, FU 10120 may effectively provide an EUDP, or EUSITT 20344 addresses, from a literal field in a FU 10120 microinstruction word. Such a FU 10120 microinstruction word literal field may be effectively utilized as an SOP into EUSDT 20266.

Handshaking between EU 10122 and FU 10120 during load COMQ 20342 operations may proceed as illustrated in Fig. 260. A twelve bit EUDP may be placed on EUDIS Bus 20206 and Control Signal Load Command Queue (LDCMQ) asserted. If COMQ 20342 is full, EU 10122 raises control signal Command Hold (CMDHOLD) which causes FU 10120 to remain in State M0 until there is room in COMQ 20342. As previously described, COMQ 20342 is comprised of two, two word buffers wherein one buffer is utilized for normal SOP operation and the other utilized for control of FU 10120 and EU 10122 internal mechanism operation.

EUDPs are loaded into COMQ 20342 when state timing signals M1CPT and M1 are asserted. If a EUDP being transferred into COMQ 20342 concerns a double precision floating point operation, control signal Set Double Precision (SETDP) is asserted. SETDP is utilized to control OPB 20322, and because single precision and precision floating point operations otherwise utilize the same SOP and thus would otherwise refer to same EUSITT 20344 microinstruction sequence.

At this point, a EUDP has been loaded into COMQ 20342 and will be decoded to control FU 10120 operation by EUCL 20310 as previously described. Each particular EUDP will be cleared by that EUDPs EUSITT 20344 microinstruction sequence after the requested microinstruction sequence has been executed.

b.b.b. Loading of Operand Buffer 20320 (Fig. 261)

Referring to Fig. 261, a diagramic representation of the interface and handshaking between EU 10122, FU 10120 and MEM 10112 for loading OPB 20322 is shown. Control signal Clear Queue Full (CLQF) from EU 10122 must be asserted by EU 10122 before FU 10120 initiates a request to MEM 10112 for an operand to be transferred to EU 10122. CLQF clears and "EU 10122's OPB 20322 Full" condition in FU 10120. CLQF indicates, thereby, that there is room in OPB 20322 to receive operands. If FU 10120 is in a "EU 10122's OPB 20322 Full" condition and further operand is required to be transferred to EU 10122, FU 10120 will remain in State M1 until CLQF is asserted.

At the beginning of execution of a particular SOP, FU 10120 may transfer two operands to OPB 20322 without "EU 10122's OPB 20322 Full" condition occurring. This is because EU 10122 is idle at the beginning of an SOP execution and generally immediately unloads a first operand from OPB 20322 before a second operand arrives.

Control signal Job Processor Operand (JPOP) provided from FU 10120 must be non-asserted for operands to be transferred from MEM 10112 to OPB 20322 through MOD Bus 10144. This is the normal condition of JPOP. If JPOP is asserted, OPB 20322 is loaded with data from JPD Bus 10142. Data is strobed into OPB 20322 from JPD Bus 10142 by control signals M1CPT and JPOP. Operands read from MEM 10112, however, are transferred into OPB 20322 through MOD Bus 10144 when MEM 10112 asserts DAVEB to indicate that valid data from MEM 10112 is available on MOD Bus 10144. DAVEB is also utilized to strobe data on MOD Bus 10144 into OPB 20322. If control signal ZFILL from MEM 10112 is asserted at this point, ZFILL is interpreted during integer operand operations to indicate that those operands are unsigned and

Petitioner Apple Inc. - Exhibit 1004, p. 5278

should be left zero filled, rather than sign extended. If data is being provided from JPD Bus 10142 rather than from MEM 10112, that is if JPOP is asserted, bit 11 of current EUDP may be utilized to perform the same function as ZFILL during loading of OPB 20322 from MOD Bus 10144.

Loading of OPB 20322 is controlled, in part, by bits 9 and 10 of EUDPs provided from FU 10120 through EUDIS Bus 20206. Bit 9 indicates length of a first operand while bit 10 indicates length of a second operand. Operand length, together with operand type specified in address portion of a EUDP, determines how a particular operand is unloaded from OPB 20322 and transferred into MULT 20314 and EXP 20316.

At this point, both COMQ 20342 and OPB 20322 have been loaded with, respectively, EUDPs and operands. It should be noted that operands are generally not transferred into OPB 20322 before a corresponding EUDP is loaded into COMQ 20342. Operands and EUDPs may, however, be simultaneously transferred into EU 10122. If other operands are required for a particular operation, those operands are loaded into OPB 20322 as described above.

c.c.c. Storeback (Fig. 262)

Referring to Fig. 262, a diagramic representation of a storeback, or transfer, of results to MEM 10112 from EU 10122 and handshaking performed therein is shown. When a final result of a EU 10122 operation is available, EU 10122 asserts control signal Data Ready (DRDY). FU 10120 thereupon responds with control signal Transfer to JPD Bus 10142 (XJPD), which gates EU 10122's result onto JPD Bus 10142. In normal operation, that is execution of SOPs, FU 10120 causes EU 10122's result to be stored back into a destination in MEM 10112, as selected by a physical descriptor provided from FU 10120. Alternately, a result may be transferred into FU 10120, 32 bits, or one word, at a time.

FU 10120 may, as described above and described further below, check EU 10122 test conditions during storeback of results. FU 10120 generates control signal Transfer Complete (XFRC) once the storeback operation is completed. XFRC also indicates to EU 10122 that EU 10122's results and test conditions have been accepted by FU 10120, so that EU 10122 need no longer assert these results and test conditions.

d.d.d. Test Conditions (Fig. 263)

Referring to Fig. 263, a diagramic representation of checking of EU10122 test conditions by FU 10120, and handshaking therein, is shown. As previously described, test results indicating certain conditions and operations of EU 10122 are sampled and stored in TSTCOND 20384 and may be examined by FU 10120. When DRDY is asserted by EU 10122, FU 10120 may select, for example, one of 8 EU 10122 conditions to test, as well as transferring results as described above. EU 10122 conditions which may be tested by FU 10120 are listed and described below. Such conditions, as whether a final result is positive, negative, or zero, may be checked in order to facilitate conditional branching of FU 10120 operations as previously described. FU 10120 specifies a condition to be tested through Test Condition Select signals (TEST(24)). FU 10120 asserts control signal EU Test Enable (EUTESTEN) to EU 10122 to gate the selected test condition. That selected test condition then appears as Data Signal Test Condition (TC) from EU 10122 to FU 10120. A TC of logic 1 may, for example, indicate that the selected condition is false while a TC of logic 0 may indicate that the selected condition is true. FU 10120 indicates that FU 10120 has sensed the requested test condition, and that the test condition need no longer be asserted by EU 10122, by asserting control signal XFRC.

e.e.e. Exception Checking (Fig. 264)

Referring to Fig. 264, a diagramic representation of exception checking of EU 10122 exceptions by FU 10120, and handshaking therein, is shown. As previously described, any EU 10122 exception conditions may be checked by FU 10120 as FU 10120 is initiating storeback of EU 10122 results. Exception checking may detect, for example, attempted division by zero, floating point exponent underflow or overflow, or a container size fault. An attempted division by zero or floating point underflow or overflow may be checked before storeback, that is without specific request by FU 10120.

As previously described, a container size fault is detected by CONSIZE 20352 by comparing length of result with size of destination container in MEM 10112. Container size exception checking occurs during store back of EU 10122 results, that is while FU 10120 is in State SB. Container size is automatically performed by EU 10122 hardware, that is by CONSIZE 20352, only on results of less than 33 bits length. Size checking of larger results, that is larger integers and BCD results, is performed by a microcode routine, using CONSIZE 20352's output, as transfer of such larger results is executed as string transfer. It is unnecessary to perform container size check for either single or double precision floating point results as these data types always occupy either 32 or 64 bits. Destination container size is provided to CONSIZE 20352 through LENGTH Bus 20226.

Control signal Length to Memory AON or Random Signals (LMAONRS) is generated by FU 10120 from Type field of the logical descriptor corresponding to a particular EU 10122 result. LMAONRS indicates that the results data type is an unsigned integer. LMAONRS determines the manner in which a required container size of the EU 10122 result is determined. After receivng this information from LMAONRS, EU 10122 determines whether destination container size in MEM 10112 is sufficiently large to contain the EU

Petitioner Apple Inc. - Exhibit 1004, p. 5279

10122 result. If that destination container size is not sufficiently large, a container size fault is detected by CONSIZE 20352, or through an EU 10122 microinstruction sequence.

Container size faults, as well as division by zero and exponent underflow and overflow faults, are signaled to FU 10120 when FU 10120 asserts control signal Check Size (CKSIZE). At this time, EU 10122 asserts control signal Exception (EXCPT) if any of the above faults has occurred. If a fault has occurred, an Event request to FU 10120 results. When an Event request is honored by FU 10120, FU 10120 may interrupt EU 10122 and dispatch EU 10122 to a microinstruction routine that transfers those exception conditions onto JPD Bus 10142. If a container size fault has caused that exception condition, EU 10122 may transfer to FU 10120 the required container size through JPD Bus 10142.

f.f.f. Idle Routine

Finally, when a current EU 10122 operation is completed, EU 10122 goes into an Idle loop microinstruction routine. If necessary, FU 10120 may assert control signal Excute Unit Abort (EUABORT) to force EU 10122 into Idle loop microinstruction routine until EU 10122 is required for further operations.

g.g.g. EU 10122 Stack Mechanism (Figs. 265, 266, 267)

As previously described, EU 10122 may perform either of two classes of operations. First, EU 10122 may perform arithmetic operations in execution of SOPs of user's programs. Second, EU 10122 may operate as an arithmetic calculator assisting operation of FU 10120's internal mechanisms and operations, referred to as kernel operations.

In kernel operation, EU 10122 acts as an arithmetic calculator for FU 10120 during address generation, address translation, and other kernel functions. In kernel mode, EU 10122 is executing microinstruction sequences at request of FU 10120 kernel microinstruction sequences, rather than at request of an SOP. In general, these kernel operations are vital to operation of JP 10114. FU 10120 may interrupt EU 10122 operations with regard to SOPs and initiate EU 10122 microinstruction sequences to perform kernel operations.

When interrupted, EU 10122 saves EU 10122's current operating state in a one level deep stack. EU 10122 may then accept an EUDP from that portion of COMQ 20342 utilized to receive and store EUDPs regarding FU 10120's and EU 10122's internal, or kernel, operations. When requesting kernel operations by EU 10122, FU 10120 generally transfers operands to OPB 20322 through JPD Bus 10142, and receives EU 10122 final results through JPD Bus 10142. Operands may also be provided to EU 10122 through MOD Bus 10144. After EU 10122 has completed a requested kernel operation, EU 10122 reloads operating state from its internal stack and continues normal operation from the point normal operation was interrupted.

Should another interrupt from FU 10120 occur while a prior interrupt is being executed, EU 10122 moves current state and data, that is of first interrupt, to MEM 10112. EU 10122 requests FU 10120 store state and date of first interrupt in MEM 10112 by requesting an "EU 10122 Stack Overflow" Event. EU 10122's "normal" state, that is state and data pertaining to the operation EU 10122 is executing at time of occurrence of first interrupt, is stored in an EU 10122 internal stack and remains there. EU 10122 then begins executing second interrupt. When EU 10122 has completed operations for second interrupt, state from first interrupt is reloaded from MEM 10112 by EU 10122 requesting a "EU 10122 Stack Underflow" Event to FU 10120. EU 10122 then completes execution of first interrupt and reloads state and resumes execution of normal operation, that is the operation being executed before the first interrupt.

EU 10122 is therefore capable of handling interrupts from FU 10120 during two circumstances. First interrupt circumstance is comprised of interrupts occurring during normal operation, that is while executing SOPs of user's programs. Second circumstance arises when interrupts occur during kernel operations, that is during execution of microinstruction sequences for handling interrupts. EU 10122 operation will be described next below for each of these circumstances, and in the order referred to.

Referring to Fig. 265, a diagramic representation of EU 10122's stack mechanisms, previously described, is shown. Those portions of EU 10122's stack mechanisms residing within EU 10122 are comprised of EU 10122's Current State Registers (EUCSRs) 26510 and EU 10122's Internal Stack (EUIS) 26512. EUCSR 26510 is comprised of EU 10122's internal registers which contain data and state of current EU 10122 operation. EUCSR 26510 may be comprised, for example, of mCRD 20346, registers of TSTINT 20320, and the previously described registers within MULT 20314 and EXP 20316.

State and data contained in EUCSR 26510 is that of the operation currently being executed by EU 10122. This current state may, for example, be that of a SOP currently being executed by EU 10122, or that of an interrupt, for example a fourth interrupt of a nested sequence of interrupts, requested by FU 10120.

EUIS 26512 is comprised of certain registers of MULTRF 20350 and EXPRF 20380. EUIS 26512 is utilized to store and save current state of an SOP operation currently being executed by EU 10122 and which has been interrupted. State and data of that SOP operation will remain stored in EUIS 26512 regardless of the number of interrupts which may occur on a nested sequence of interrupts requested by FU 10120. State and data of the interrupted SOP operation will be returned from EUIS 26512 to EUCSR 26510 when all interrupts have been completed.

Final portion of EU 10122's stack mechanism is that portion of EU 10122's internal stack (EUES) 26514

Petitioner Apple Inc. - Exhibit 1004, p. 5280

residing in MEM 10112. EUES 26514 is comprised of certain MEM 10112 address locations used to store state and data of successive interrupt operations of sequences of nested interrupts. That is, if a sequence of four interrupts is requested by FU 10120, state and data of fourth interrupt will reside in EUCSR 26510 while state and data of first, second, and third interrupts have been transferred, in sequence, into EUES 26514. In this respect, and as previously described operation of EU 10122's stack mechanisms is similar to that of, for example, MIS 10368 and SS 10336 previously described with reference to Fig. 103.

As described above, an interrupt may be requested of EU 10122 by FU 10120 either during EU 10122 normal operation, that is during execution of SOPs by EU 10122, or while EU 10122 is executing a previous interrupt requested by FU 10120. Operation of EU 10122 and FU 10120 upon occurrence of an interrupt during EU 10122 normal operation will be described next below.

Referring to Fig. 266, a diagramic representation of handshaking between EU 10122 and FU 10120 during an interrupt of EU 10122 while EU 10122 is operating in normal mode is shown and should be referred to in conjunction with Fig. 265. For purposes of the following discussions, interrupts of EU 10122 operations by FU 10120 are referred to as nanointerrupts to distinguish from interrupts internal to FU 10120.

FU 10120 interrupts normal operation of EU 10122 by assertion of control signal Nano-Interrupt (NINTP) during State M0 of FU 10120 operation. NINTP may be masked by EU 10122 during certain critical EU 10122 operations, such as arithmetic operations. If NINTP is masked by EU 10122, FU 10120 will remain in State NW until EU 10122 acknowledges the interrupt.

Upon receiving NINTP from FU 10120, EU 10122s transfers state and data of current SOP operation from EUCSR 26510 to EUIS 26512. EU 10122 then asserts control signal Nano-Interrupt Acknowledge (NIACK) to FU 10120 to acknowledge availability of EU 10122 to accept a nanointerrupt. FU 10120 will then enter State M1 and place an EUDP on EUDIS Bus 20206. Loading of COMQ 20342 then proceeds as previously described, with EU 10122 loading nanointerrupt EUDPs into the appropriate registers of COMQ 20342. COMQ 20342 is loaded as previously described and, if JPOP is asserted, data transferred into OPB 20322 from JPD Bus 10142. If JPOP is not asserted, data is taken into OPB 20322 from MOD Bus 10144. EU 10122 then proceeds to execute the required nanointerrupt operation and storing back of results and checking of test conditions proceeds as previously described for EU 10122 normal operation. In general, exception checking is not performed. When EU 10122 has completed execution of the nanointerrupt operation, EU 10122 transfers state and data of the interrupted SOP operation from EUIS 26512 to EUCSR 26510 and resumes execution of that SOP. At this point, EU 10122 asserts control signal Nano-Interrupt Trap Enable (NITE). NITE is received and tested by FU 10120 to indicate end of nanointerrupt processing.

Referring to Fig. 267, a diagramic representation of interfaces between EU 10122, FU 10120, and MEM 10112 during nested, or sequential, EU 10122 interrupts for kernel operations, and handshaking therein, is shown. During the following discussion, it is assumed that EU 10122 is already processing a nanointerrupt for a kernel operation submitted to EU 10122 by FU 10120. FU 10120 may then submit a second, third, or fourth, nanointerrupt to EU 10122 for a further kernel operation. FU 10120 will assert NINTP to request a nanointerrupt of EU 10122. EU 10122's normal mode state and data from a previously executing SOP operation has been stored in, and remains in, EUIS 26512. Current state and data of currently executing nanointerrupt operation in EUCSR 26510 will be transferred to EUES 26514 in MEM 10112 to allow initiation of pending nanointerrupt. EU 10122 will at this time assert NIACK and control signal Execute Unit Event (EXEVT). EXEVT to FU 10120 informs FU 10120 that an EU 10122 Event has occurred, specifically, and in this case, EXEVT requests FU 10120 service of an EU 10122 Stack Overflow. FU 10120 is thereby trapped to an "EU 10122 Stack Overflow" Event Handler microinstruction sequence. This handler transfers current state and data of interrupted nanointerrupt previously executing in EU 10122 into EUES 26514. State and data of interrupted nanointerrupt is transferred to EUES 26514, one 32 bit word at a time. FU 10120 asserts control signals XJPD to gate each of these state and data words onto JPD Bus 10142 and controls transfer of these words into EUES 26514.

Processing of new nanointerrupt proceeds as described above with reference to interrupts occurring during normal operation. If any subsequent nanointerrupts occur, they are handled in the same manner as just described; FU 10120 signals a nanointerrupt to FU 10120, current EU 10122 state and data is saved by FU 10120 in EUES 26514, and new nanointerrupt is processed. After a nested nanointerrupt, that is a nanointerrupt of a sequence of nanointerrupts, has been serviced, EU 10122 asserts control signal EU 10122 Trap (ETRAP) to FU 10120 to request a transfer of a previous nanointerrupt's state and data from EUES 26514 to EUCSR 26510. FU 10120 will retrieve that next previous nanointerrupt state and data from EUES 26514 through MOD Bus 10144 and will transfer that data and state onto JPD Bus 10142. This state and data is returned, one 32 bit word at a time, and is strobed into EU 10122 by JPOP from FU 10120. Processing of that prior nanointerrupt will then resume. The servicing of successively prior nanointerrupts will continue until all previous nanointerrupts have been serviced. Original state and data of EU 10122, that is that of SOP operation which was initially interrupted, is then returned to EUCSR 26510 from EUIS 26512 and execution of that SOP resumed. At this time, EU 10122 asserts NITE to indicate end of EU 10122 kernel operations in regard to nanointerrupts.

Having described structure and operation of EU 10122, FU 10120 and MEM 10112, with respect to servicing of kernel operation nanointerrupts by EU 10122, loading of EU 10122's EUSITT 20344 with microinstruction sequences will be described next below.

Petitioner Apple Inc. - Exhibit 1004, p. 5281

h.h.h.h Loading of Execute Unit S-Interpreter Table 20344 (Fig. 268)

Referring to Fig. 268, a diagramic representation of interface and handshaking between EU 10122, FU 10120, MEM 10112, and DP 10118 during loading of microinstructions into EUSITT 20344 is shown. As previously described, EUSITT 20344 contains all microinstructions required for control of EU 10122 in executing kernel nanointerrupt operations and in executing arithmetic operations in response to SOPs of user's programs. EUSITT 20344 may store microinstruction sequences for interpreting arithmetic SOPs of user's programs for, for example, up to 4 different S-Language Dialects. In general, a capacity of storing microinstruction sequences for arithmetic operations in up to 4 S-Language Dialects is sufficient for most requirements, so that EUSITT 20344 need be loaded with microinstruction sequences only at initialization of CS 10110 operation. Should microinstruction sequences for arithmetic operations of more than 4 S-Language Dialects be required, those microinstruction sequences may be loaded into EUSITT 20344 in the manner as will be described below.

As previously described, a portion of the microinstructions stored in EUSITT 20344 is contained in Read Only Memories and is thus permanently stored in EUSITT 20344. Microinstruction sequences permanently stored in EUSITT 20344 are, in general, those required for execution of kernel operations. Microinstruction sequences permanently stored in EUSITT 20344 include those used to assist in writing other EU 10122 microinstruction sequences into EUSITT 20344 as required. Certain microinstruction sequences are stored in a Random Access Memory, referred to as the Writeable Control Store (WCS) portion of EUSITT 20344, and include these for interpreting arithmetic operation SOPs of various S-Language Dialects.

Writing of microinstruction sequences into EU 10122 is initialized by forcing EU 10122 into an Idle state. Initialization of EU 10122 is accomplished by FU 10120 asserting EUABORT or by DP 10118 asserting control signal clear (CLEAR). Either EUABORT or CLEAR will clear a current operation of EU 10122 and force EU 10122 into Idle state, wherein EU 10122 waits for further EUDPs provided from FU 10120. FU 10120 then dispatches a EUDP initiating loading of EUSITT 20344 to EU 10122 through EUDIS Bus 20206. Load EUSITT 20344 EUDP specifies starting address of a two step microinstruction sequence in the PROM portion of EUSITT 20344. This two step microinstruction sequence first loads zeros into SCAG 25536, which as previously described provides read and write addresses to EUSITT 20344. EUSITT 20344 load microinstruction sequence then reads a microinstruction from EUSITT 20344 to mCRD 20346. This microinstruction specifies conditions for handshaking operations with FU 10120 so that loading of EUSITT 20344 may begin. At this time, and from this microinstruction word, EU 10122 asserts control signal DRDY to FU 10120 to indicate that EU 10122 is ready to accept EUDPs from FU 10120 for directing loading of EUSITT 20344. This initial microinstruction also generates a write enable control signal for the WCS portion of EUSITT 20344, inhibits loading of mCRD 20346 from EUSITT 20344, and inhibits normal loading operations of NXTR 25540 and SCAG 25536. This first microinstruction also directs NASS 25526 to accept address inputs from SCAG 25536 and, finally, causes NITE to FU 10120 to be asserted to unmask nanointerrupts from FU 10120.

FU 10120 then generates a read request to MEM 10112, and MEM 10112 transfers a first 32 bit word of a EU 10122 microinstruction word onto JPD Bus 10142. Each such 32 bit word from MEM 10112 comprises one half of a 64 bit microinstruction word of EU 10122. When FU 10120 receives DRDY from EU 10122, FU 10120 generates control signal Load Writeable Control Store (LDWCS). LDWCS in turn transfers a 32 bit word on JPD Bus 10142 into a first address of the WCS portion of EUSITT 20344. A next 32 bit half word of a EU 10122 microinstruction word is then read from MEM 10112 through JPD Bus 10142 and transferred into the second half of that first address within the WCS portion of EUSITT 20344. The address in SCAG 25536 is then incremented to select a next address within EUSITT 20344 and the process just described repeated automatically, including generation of DRDY and LDWCS, until loading of EUSITT 20344 is completed.

After loading of EUSITT 20344 is completed, the loading process is terminated when FU 10120 asserts NINTP, or DP 10118 asserts Control Signal Load Complete (LOADCR). Either NINTP or LOADCR releases control of operation of NAG 20340 to allow EU 10122 to resume normal operation.

The above descriptions have described structure and operation of EU 10122, including: execution of various arithmetic operations utilizing various operand formats; operation of EU 10122, FU 10120, and MEM 10112 with regard to handshaking; loading of EUDPs and operands; storeback of results; checking of test conditions and exceptions; EU 10122 Stack Mechanisms during normal and kernel operations; and loading of EU 10122 microinstruction sequences into EUSITT 20344. IOS 10116 and DP 10118 will be described next below, in that order.

D. I/O System 10116 (Figs. 204, 206, 269)

Referring to Fig. 204, a partial block diagram of IOS 10116 is shown. As previously described, IOS 10116 operates as an interface between CS 10110 and the external world, for example, ED 10124. A primary function of IOS 10116 is the transfer of data between CS 10110, that is MEM 10112, and the external world. In addition to performing transfers of data, IOS 10116 controls access between various data sources and sinks of ED 10124 and MEM 10112. As previously described, IOS 10116 directly addresses MEM 10112's physical address space to write data into or read data from MEM 10112. As such, IOS 10116 also performs address translation, a mapping operation required in transferring data between MEM 10112's physical

Petitioner Apple Inc. - Exhibit 1004, p. 5282

address space and address spaces of data sources and sinks in ED 10124.

As shown in Fig. 204, IOS 10116 includes Data Mover (DMOVR) 20410, Input/Output Control Processor (IOCP) 20412, and one or more data channel devices. IOS 10116's data channel devices may include ECLIPSE® Burst Multiplexer Channel (EBMC) 20414, NOVA Data Channel (NDC) 20416, and other data channel devices as required for a particular configuration of a CS 10110 system. IOCP 20412 controls and directs transfer of data between MEM 10112 and ED 10124, and controls and directs mapping of addresses between ED 10124 and MEM 10112's physical address space. IOCP 20412 may be comprised, for example, of a general purpose computer, such as an ECLIPSE® M600 computer available from Data General Corporation of Westboro, Massachusetts.

EBMC 20414 and NDC 20416 comprise data channels through which data is transferred between ED 10124 and IOS 10116. EBMC 20414 and NDC 20416 perform actual transfers of data to and from ED 10124, under control of IOCP 20412, and perform mapping of ED 10124 addresses to MEM 10112 physical addresses, also under control of IOCP 20412. EBMC 20414 and NDC 20416 may respectively be comprised, for example, of an ECLIPSE® Burst Multiplexer Data Channel and a NOVA® Data Channel, also available from Data General Corporation of Westboro, Massachusetts.

DMOVR 20410 comprises IOS 10116's interface to MEM 10112. DMOVR 20410 is the path through which data and addresses are transferred between EBMC 20414 and NDC 20416 and MEM 10112. Additionally, DMOVR 20410 controls access between EBMC 20414, NDC 20416, and other IOS 10116 data channels, and MEM 10112.

ED 10124, as indicated in Fig. 204, may be comprised of one or more data sinks and sources. ED 10124 data sinks and sources may include commercially available disc drive units, line printers, communication lengths, tape units, and other computer systems, including other CS 10110 systems. In general, ED 10124 may include all such data devices as are generally interfaced with a computer system.

a. I/O System 10116 Structure (Fig. 204)

Referring first to the overall structure of IOS 10116, data input/output of ECLIPSE® Burst Multiplexer Channel Adapter and Control Circuitry (BMCAC) 20418 of EBMC 20414 is connected to bi-directional BMC Address and Data (BMCAD) Bus 20420. BMCAD Bus 20420 in turn is connected to data and address inputs and outputs of data sinks and sources of ED 10124.

Similarly, data and address inputs and outputs of NOVA® Data Channel Adapter Control Circuits (NDCAC) 20422 in NDC 20416 is connected to bi-directional NOVA® Data Channel Address and Data (NDCAD) Bus 20424. NDCAD Bus 20424 in turn is connected to address and data inputs and outputs of data sources and sinks of ED 10124. BMCAD Bus 20420 and NDCAD Bus 20424 are paths for transfer of data and addresses between data sinks and sources of ED 10124 and IOS 10116's data channels and may be expanded as required to include other IOS 10116 data channel devices and other data sink and source devices of ED 10124.

Within EBMC 20414, bi-directional data input and output of BMCAC 20418 is connected to bi-directional input and output of BMC Data Buffer (BMCDB) 20426. Data inputs and data outputs of BMCDB 20426 are connected to, respectively, Data Mover Output Data (DMOD) Bus 20428 and Data Mover Input Data (DMID) Bus 20430. Address outputs of BMCAC 20418 are connected to address inputs of Burst Multiplexer Channel Address Translation Map (BMCATM) 20432 and address outputs of BMCATM 20432 are connected onto DMID Bus 20430. A bi-directional control input and output of BMCATM 20432 is connected from bi-directional IO Control Processor Control (IOCPC) Bus 20434.

Referring to NDC 20416, as indicated in Fig. 204 data inputs and outputs of NDCAC 20422 are connected, respectively, from DMOD Bus 20428 and to DMID Bus 20430. Address outputs of NDCAC 20422 are connected to address inputs of NOVA® Data Channel Address Translation Map (NDCATM) 20436. Address outputs of NDCATM 20436 are, in turn, connected onto DMID Bus 20430. A bi-directional control input and output of NDCATM 20436 is connected from IOCPC Bus 20434.

Referring to IOCP 20412, a bi-directional control input and output of IOCP 20412 is connected from IOCPC Bus 20434. Address and data output of IOCP 20412 is connected to NDCAD Bus 20424. An address output of IOCP Address Translation Map (IOCPATM) 20438 within IOCP 20412 is connected onto DMID Bus 20430. Data inputs and outputs of IOCP 20412 are connected, respectively, to DMOD Bus 20428 and DMID Bus 20430. A bi-directional control input and output of IOCP 20412 is connected to a bi-directional control input and output of DMOVR 20410.

Referring finally to DMOVR 20410, DMOVR 20410 includes Input Data Buffer (IDB) 20440, Output Data Buffer (ODB) 20442, and Priority Resolution and Control (PRC) 20444. A data and address input of IDB 20440 is connected from DMID Bus 20430. A data and address output of IDB 20440 is connected to IOM Bus 10130 to MEM 10112. A data output of ODB 20442 is connected from MIO Bus 10129 from MEM 10112, and a data output of ODB 20442 is connected to DMOD Bus 20428. Bi-directional control inputs and outputs of IDB 20440 and ODB 20442 are connected from bi-directional control inputs and outputs of PRC 20444. A bi-directional control input and output of PRC 20444 is connected from a bi-directional control input and output of IOCP 20412 as described above. Another bi-directional control input and output of PRC 20444 is connected to and from IOMC Bus 10131 and thus from a control input and output of MEM 10112. Having described overall structure of IOS 10116, operation of IOS 10116 will be described next below.

Petitioner Apple Inc. - Exhibit 1004, p. 5283

b. I/O System 10116 Operation (Fig. 269)

1. Data Channel Devices

Referring first to EBMC 20414, BMCAC 20418 receives data and addresses from ED 10124 through BMCAD Bus 20420. BMCAC 20418 transfers data into BMCDB 20426, where that data is held for subsequent transmission to MEM 10112 through DMOVR 20410, as will be described below. BMCAC 20418 transfers addresses received from ED 10124 to BMCATM 20432. BMCATM 20432 contains address mapping information correlating ED 10124 addresses with MEM 10112 physical addresses. BMCATM 20432 thereby provides MEM 10112 physical addresses corresponding to ED 10124 addresses provided through BMCAC 20418.

When, as will be described further below, EBMC 20414 is granted access to MEM 10112 to write data into MEM 10112, data stored in BMCDB 20426 and corresponding addresses from BMCATM 20432 are transferred onto DMID Bus 20430 to DMOVR 20410. As will be described below, DMOVR 20410 then writes that data into those MEM 10112 physical address locations. When data is to be read from MEM 10112 to ED 10124, data is provided by DMOVR 20410 on DMOD Bus 20428 and is transferred into BMCDB 20426. BMCAC 20418 then reads that data from BMCDB 20426 and transfers that data onto BMCAD Bus 20420 to ED 10124. During transfers of data from MEM 10112 to ED 10124, MEM 10112 does not provide addresses to be translated into ED 10124 addresses to accompany that data. Instead, those addresses are generated and provided by BMCAC 20418.

NDC 20416 operates in a manner similar to that of EBMC 20414 except that data inputs and outputs of NDCAC 20422 are not buffered through a BMCDB 20426.

As previously described, MEM 10112 has capacity to perform block transfers, that is sequential transfers of four 32 bit words at a time. In general, such transfers are performed through EBMC 20414 and are buffered through BMCDB 20426. That is, BMCDB 20426 allows single 32 bit words to be received from ED 10124 by EBMC 20414 and stored therein until a four word block has been received. That block may then be transferred to MEM 10112. Similarly, a block may be received from MEM 10112, stored in BMCDB 20426, and transferred one word at a time to ED 10124. In contrast, NDC 20416 may generally be utilized for single word transfers.

As indicated in Fig. 204, EBMC 20414, NDC 20416, and each data channel device of IOS 10116 each contain an individual address translation map, for example BMCATM 20432 in EBMC 20414 and NDCATM 20436 in NDC 20416. Address translation maps stored therein are effectively constructed and controlled by IOCP 20412 for each data channel device. IOS 10116 may thereby provide an individual and separate address translation map for each IOS 10116 data channel device. This allows IOS 10116 to insure that no two data channel devices, nor two groups of data sinks and sources in ED 10124, will mutually interfere by writing into and destroying data in a common area of MEM 10112 physical address space. Alternately, IOS 10116 may generate address translation maps for two or more data channel devices wherein those maps share a common, or overlapping, area of MEM 10112's physical address space. This allows data stored in MEM 10112 to be transferred between IOS 10116 data channel devices through MEM 10112, and thus to be transferred between various data sink and source devices of ED 10124. For example, a first ED 10124 data source and a first IOS 10116 data channel may write data to be operated upon into a particular area of MEM 10112 address space. The results of CS 10110 operations upon that data may then be written into a common area shared by that first data device and a second data device and read out of MEM 10112 to a second ED 10124 data sink by that second data channel device. Individual mapping of IOS 10116's data channel devices thereby provides total flexibility in partitioning or sharing of MEM 10112's address space through IOS 10116.

2. I/O Control Processor 20412

As described above, IOCP 20412 is a general purpose computer whose primary function is overall direction and control or data transfer between MEM 10112 and ED 10124. IOCP 20412 controls mapping of addresses between IOS 10116's data channel devices and MEM 10112 address space. In this regard, IOCP 20412 generates address translation maps for IOS 10116's data channel devices, such EBMC 20414 and NDC 20416. IOCP 20412 loads these address translation maps into and controls, for example, BMCATM 20432 of EBMC 20414 and NDCATM 20436 and NDC 20416 through IOCPC Bus 20434. IOCP 20412 also provides certain control functions to DMOVR 20410, as indicated in Fig. 204. In addition to these functions, IOCP 20412 is also provided with data and addressing inputs and outputs. These data addressing inputs and outputs may be utilized, for example, to obtain information utilized by IOCP 20412 in generating and controlling mapping of addresses between IOS 10116's data channel devices and MEM 10112. Also, these data and address inputs and outputs allow IOCP 20412 to operate, in part, as a data channel device. As previously described, IOCP 20412 has data and address inputs and outputs connected from and to DMID Bus 20430 and DMOD Bus 20428. IOCP 20412 thus has access to data being transferred between ED 10124 and MEM 10112, providing IOCP 20412 with direct access to MEM 10112 address space. In addition, IOCP 20412 is provided with control and address outputs to NDCAD Bus 20424, thus allowing IOCP 20412 partial control of certain data source and sink devices in ED 10124.

Petitioner Apple Inc. - Exhibit 1004, p. 5284

3. Data Mover 20410 (Fig. 269)

a.a. Input Data Buffer 20440 and Output Data Buffer 20442

As described above, DMOVR 20410 comprises an interface between IOS 10116's data channels and MEM 10112. DMOVR 20410 performs actual transfer of data between IOS 10116's data channel devices and MEM 10112, and controls access between IOS 10116's data channel devices and MEM 10112. IDB 20440 and ODB 20442 are data and address buffers allowing asynchronous transfer of data between IOS 10116 and MEM 10112. That is, ODB 20442 may accept data from MEM 10112 as that data becomes available and then hold that data until an IOS 10116 data channel device, for example EBMC 20414, is ready to accept that data. IDB 20440 accepts data and MEM 10112 physical addresses from IOS 10116's data channel devices. IDB 20440 holds that data and addresses for subsequent transmission to MEM 10112 when MEM 10112 is ready to accept data and addresses. IDB 20440 may, for example, accept a burst, or sequence, of data from EBMC 20414 or single data words from NDC 20416 and subsequently provide that data to MEM 10112 in block, or four word, transfers as previously described. Similarly, ODB 20442 may accept one or more block transfers or data from ODB 20442 and subsequently provide that data to NDC 20416 as single words, or to DMID 20430 as a data burst. In addition, as previously described, a block transfer from MEM 10112 may not appear as four sequential words. In such cases, ODB 20442 accepts the four words of a block transfer as they appear on MIO Bus 10129 and assembles those words into a block comprising four sequential words for subsequent transfer to ED 10124.

Transfer of data through IDB 20440 and ODB 20442 is controlled by PRC 20444, which exchanges control signals with IOCP 20412 and has an interface, previously described, to MEM 10112 through IOMC Bus 10131.

b.b. Priority Resolution and Control 20444 (Fig. 269)

As previously described, PRC 20444 controls access between IOS 10116 data channel devices and MEM 10112. This operation is performed by means of a Ring Grant Access Generator (RGAG) within PRC 20444.

Referring to Fig. 270, a diagramic representation of PRC 20444's RGAG is shown. In general, PRC 20444's RGAG is comprised of a Ring Grant Code Generator (RGCG) 26910 and one or more data channel request comparators. In Fig. 269, PRC 20444's RGAC is shown as including ECLIPSE® Burst Multiplexer Channel Request Comparator (EBMCRC)26912, NOVA® Data Channel Request Comparator (NDCRC) 26914, Data Channel Device X Request Comparator (DCDXRC) 26916, and Data Channel Device Z Request Comparator (DCDZRC) 26918. PRC 20444's RGAG may include more or fewer request comparators as required by the number of data channel devices within a particular IOS 10116.

As indicated in Fig. 269, Request Grant Code (RGC) outputs of RGCG 26910 are connected in parallel to first inputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918. Second inputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are connected from other portions of PRC 20444 and receive indications that, respectively, EBMC 20414, NDC 20416, DCDX, or DCDZ has submitted a request for a read or write access to MEM 10112.

Request Grant Outputs (GRANT) of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are in turn connected to other portions of PRC 20444 circuitry to indicate when read or write access to MEM 10112 has been granted in response to a request by a particular IOS 10116 data channel device. When indication of such a grant is provided to those other portions of PRC 20444, PRC 20444 proceeds to generate appropriate control signals to MEM 10112, through IOMC Bus 10131 as previously described, to IDB 20440 and ODB 20442, and to IOCP 20412. PRC 20444's control signals initiate that read or write request to that IOS 10116 data channel device. Grant outputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are also provided as inputs to RGCG 26910 to indicate, as described further below, when a particular IOS 10116 has requested and been granted access to MEM 10112.

As indicated in Fig. 269, a diagramic figure above RGCG 26910, RGCG generates a repeated sequence of unique RGCs. Herein indicated as numeric digits 0 to 15. Each RGC identifies, or defines, a particular time slot during which a IOS 10116 data channel device may be granted access to MEM 10112. Certain RGCs are, effectively, assigned to particular IOS 10116 data channel devices. Each such data channel device may request access to MEM 10112 during its assigned RGC identified access slots. For example, EBMC 20414 is shown as being allowed access to MEM 10112 during those access slots identified by RGCs 0, 2, 4, 6, 8, 10, 12, and 14. NDC 20416 is indicated as being allowed access to MEM 10112 during RGC slots 3, 7, 11, and 15. DCDX is allowed access during slots 1 and 9, and DCDZ is allowed access during RGC slots 5 and 13.

As described above, RGCG generates RGCs 0 to 15 in a repetitive sequence. During occurrence of a particular RGC, each request comparator of PRC 20444's RGAG examines that RGC to determine whether its associated data channel device is allowed access during that RGC slot, and whether that associated data channel device has requested access to MEM 10112. If that associated data channel device is allowed access during that RGC slot, and has requested access, that data channel device is granted access as indicated by that request comparator's GRANT output. The request comparators GRANT output is also provided as an input to RGCG 26910 to indicate to RGCG 26910 that access has been granted during that RGC slot.

If a particular data channel device has not claimed and has not been granted access to MEM 10112

Petitioner Apple Inc. - Exhibit 1004, p. 5285

during that RGC slot, RGCG 26910 will go directly to next RGC slot. In next RGC slot, PRC 20444's RGAG again determines whether the particular data channel device allowed access during that slot has submitted a request, and will grant access if such a request has been made. If not, RGCG 26910 will again proceed directly to next RGC slot, and so on. In this manner, PRC 20444's RGAG insures that each data channel device of IOS 10116 is allowed access to MEM 10112 without undue delay. In addition, PRC 20444's RGAG prevents a single, or more than one, data channel device from monopolizing access to MEM 10112. As described above, each data channel device is allowed access to MEM 10112 at least once during a particular sequence of RGCs. At the same time, by not pausing within a particular RGC in which no request for access to MEM 10112 has occurred, PRC 20444's RGAG effectively automatically skips over those data channel devices which have not requested access to MEM 10112. PRC 20444's RGAG thereby effectively provides, within a given time interval, more frequent access to those data channel devices which are most busy. In addition, the RGCs assigned to particular IOS 10116 data channel devices may be reassigned as required to adapt a particular CS 10110 to the data input and output requirements of a particular CS 10110 configuration. That is, if EBMC 20414 is shown to require less access to MEM 10112 then NDC 20416, certain RGCs may be reassigned from EBMC 20414 to NDC 20416. Access to MEM 10112 by IOS 10116's data channel devices may thereby be optimized as required.

Having described structure and operation of IOS 10116, structure and operation of DP 10118 will be described next below.

E. Diagnostic Processor 10118 (Figs. 101, 205)

Referring to Fig. 101, as previously described, DP 10118 is interconnected with IOS 10116, MEM 10112, FU 10120, and EU 10122 through DP Bus 10138. DP 10118 is also interconnected, through DPIO Bus 10136, with the external world and in particular with DU 10134. In addition to performing diagnostic and fault monitoring and correction operations, DP 10118 operates, in part, to provide control and display functions allowing an operator to interface with CS 10110. DU 10134 may be comprised, for example, of a CRT and keyboard unit, or a teletype, and provides operators of CS 10110 with all control and display functions which are conventionally provided by a hard console, that is a console containing switches and lights. For example, DU 10134, through DP 10118, allows an operator to exercise control of CS 10110 for such purposes as system initialization and startup, execution of diagnostic processes, fault monitoring and identification, and control of execution of programs. As will be described further below, these functions are accomplished through DP 10118's interfaces with IOS 10116, MEM 10112, FU 10120, and EU 10122.

DP 10118 is a general purpose computer system, for example a NOVA® 4 computer of Data General Corporation of Westboro, Massachusetts. Interface of DP 10118 and DU 10134, and mutual operation of DP 10118 and DU 10134, will be readily apparent to one of ordinary skill in the art. DP 10118's interface and operation, with IOS 10116, MEM 10112, FU 10120, and EU 10122 will be described further next below.

DP 10118, operating as a general purpose computer programmed specificially to perform the functions described above, has, as will be described below, read and write access to registers of IOS 10116, MEM 10112, FU 10120 and EU 10122 through DP Bus 10138. DP 10118 may read data directly from and write data directly into those registers. As will be described below, these registers are data and instruction registers and are integral parts of CS 10110's circuitry during normal operation of CS 10110. Access to these registers thereby allows DP 10118 to directly control or effect operation of CS 10110. In addition, and as also will be described below, DP 10118 provides, in general, all clock signals to all portions of CS 10110 circuitry and may control operation of that circuitry through control of these clock signals.

For purposes of DP 10118 functions, CS 10110 may be regarded as subdivided into groups of functionally related elements, for example DESP 20210 in FU 10120. DP 10118 obtains access to the registers of these groups, and control of clocks therein, through scan chain circuits, as will be described next below. In general, DP 10118 is provided with one or more scan chain circuits for each major functional sub-element of CS 10110.

Referring to Fig. 205, a diagramic representation of DP 10118 and a typical DP 10118 scan chain is shown. As indicated therein, DP 10118 includes a general purpose Central Processor Unit, or computer, (DPCPU) 27010. A first interface of DPCPU 27010 is with DU 10134 through DPIO Bus 10136. DPCPU 27010 and DU 10134 exchange data and control signals through DPIO Bus 10136 in the manner to direct operations of DPCPU 27010, and to display the results of those operations through DU 10134.

Associated with DPCPU 27010 is Clock Generator (CLKG) 27012. CLKG 27012 generates, in general, all clock signals used within CS 10110.

DPCPU 27010 and CLKG 27012 are interfaced with the various scan chain circuits of CS 10110 through DP Bus 10138. As described above, CS 10110 may include one or more scan chains for each major sub-element of CS 10110. One such scan chain, for example DESP 20210 Scan Chain (DESPSC) 27014 is illustrated in Fig. 205.

Interface between DPCPU 27010 and CLKG 27012 and, for example, DESPSC 27014 is provided through DP Bus 10138. As indicated in Fig. 205, DESPSC 27014 includes Scan Chain Clock Gates (SCCG) 27016 and one or more Scan Chain Registers (SCRs) 27018 to 27024.

SCCG 27016 receives clock signals from CLKG 27012 and control signals from DPCPU 27010 through DP Bus 10138. SCCG 27016 in turn provides appropriate clock signals to the various registers and circuits

Petitioner Apple Inc. - Exhibit 1004, p. 5286

of, for example, DESP 20210. Clock control signals provided by DPCPU 27010 to SCCG 27016 control, or gate, the various clock signals to these registers and circuits of DESP 20210, thereby effectively allowing DPCPU 27010 to control of DESP 20210.

SCRs 27018 to 27024 are comprised of various registers within DESP 20210. For example, SCRs 27018 to 27024 may include the output buffer registers of AONGRF 20232, OFFGRF 20234, LENGRF 20236, output registers of OFFALU 20242 and LENALU 20252, and registers within OFFMUX 20240 and BIAS 20246. Such registers are indicated in the present description, as previously described, by arrows appended to ends of those registers, with a first arrow indicating an input and a second an output. In normal CS 10110 operations, as previously described, SCRs 27018 to 27024 operate as parallel in, parallel out buffer registers through which data and instructions are transferred. SCRs 27018 to 27024 are also capable of operating as shift registers and, as indicated in Fig. 205, are connected together to comprise a single shift register circuit having an input from DPCPU 27010 and an output to DPCPU 27010. Control inputs to SCRs 27018 to 27024 from DPCPU 27010 control operation of SCRs 27018 to 27024, that is whether these registers shall operate as parallel in, parallel out registers, or as shift registers of DESPSC 27014's scan chain. The shift register scan chain comprising SCRs 27018 to 27024 allows DPCPU 27010 to read the contents of SCRs 27018 to 27024 by shifting the content of these registers into DPCPU 27010. Conversely, DPCPU 27010 may write into SCRs 27018 to 27024 by shifting information generated by DPCPU 27010 from DPCPU 27010 and through the shift register scan chain to selected locations within SCRs 27018 to 27024.

Scan chain clock generator circuits and scan chain registers of each scan chain circuit within CS 10110 thereby allow DP 10118 to control operation of each major sub-element of CS 10110. For example, to read information from the scan chain registers therein, and to write information into those scan chain registers as required for diagnostic, monitoring, and control functions.

Having described structure and operation of each major element of CS 10110, including MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118, certain operations of, in particular, FU 10120 will be described further next below. The following descriptions will further disclose operational features of JP 10114, and in particular FU 10120, by describing in greater detail certain operations therein by further describing microcode control of JP 10114.

F. CS 10110 Micromachine Structure and Operation (Figs. 270—274)

a. Introduction

The preceding descriptions have presented the hardware structures and operation of FU 10120 and EU 10122. The following description will describe how devices in FU 10120, and certain EU 10122 devices, function together as a microprogrammable computer, henceforth termed the FU micromachine. The FU micromachine performs two tasks: it interprets SINs, and it responds to certain signals generated by devices in FU 10120, EU 10122, MEM 10112, and IOS 10116. The signals to which the FU micromachine responds are termed Event signals. In terms of structure and operation, the FU micromachine is characterized by the following:

- Registers and ALUs specialized for the handling of logical descriptors.
- Registers organized as stacks for invocations of microroutines (microinstruction sequences).
- Mechanisms allowing microroutine invocations by means of event signals from hardware.
- Mechanisms which allow an invoked microroutine to return either to the microinstruction following the one which resulted in the invocation or to the microinstruction which resulted in the invocation.
- Mechanisms which allow the contents of stack registers to be transferred to MEM 10112, thereby creating a virtual microstack of limitless size.
- Mechanisms which guarantee response to an event signal within a predictable length of time.
- The division of the devices comprising the micromachine into two groups: those devices which may be used by all microcode and those which may be used only by KOS (Kernel Operating System, previously described) microcode.

These devices and mechanisms allow the FU micromachine to be used in two ways: as a virtual micromachine and as a monitor micromachine. Both kinds of micromachine use the same devices in FU 10120, but perform different functions and have different logical properties. In the following discussion, when the FU micromachine is being used as a virtual micromachine, it is said to be in virtual mode, and when it is being used as a monitor micromachine, it is said to be in monitor mode. Both modes are introduced here and explained in detail later.

When the FU micromachine is being used in virtual mode, it has the following properties:

- It runs on an essentially infinite micromachine stack belonging to a Process 610.
- It can respond to any number of event signals in the M0 cycle (state) of a single microinstruction.
- A page fault may occur on the invocation of any microroutine or on return from any microroutine.
- When the FU micromachine is in virtual mode, any microroutine may not run to completion, i.e., complete its execution in a predictable length of time, or complete it at all.
- It is executing a Process 610.

The last four properties are consequences of the first: Event signals result in invocations, and since the micromachine stack is infinite, there is no limit to the number of invocations. The infinite micromachine stack is realized by placing micromachine stack frames on Secure Stack 10336 belonging to a Process 610,

Petitioner Apple Inc. - Exhibit 1004, p. 5287

and the virtual micromachine therefore always runs on a micromachine stack belonging to some Process 610. Furthermore, if the invocation of a microroutine or a return from a microroutine requires micromachine frames to be transferred from Secure Stack 10336 to the FU micromachine, a page fault may result, and Process 610 which is executing the microroutine may be removed from JP 10114, thereby making the time required to execute the microroutine unpredictable. Indeed, if process 610 is stopped or killed, the execution of the microroutine may never finish. As will be seen in descriptions below, the Virtual Processor 612 is the means by which the virtual micromachine gains access to a Process 610's micromachine stack.

When in monitor mode, the FU micromachine has the following properties:

- It has a micromachine stack of fixed size, the stack is always available to the FU micromachine, and it is not associated with a Process 610.
- It can respond to only a fixed number of events during the M0 cycle of a single microinstruction.
- In monitor mode, invocation of a microroutine or return from a microroutine will not cause a page fault.
- Microroutines executing on the FU micromachine when the micromachine is in monitor mode are guaranteed to run to completion unless they themselves perform an action which causes them to give up JP 10114.
- Microroutines executing in monitor mode need not be performing functions for a Process 610.

Again, the remaining properties are consequences of the first: because the monitor micromachine's stack is of fixed size, the number of events to which the monitor micromachine can respond is limited; furthermore, since the stack is always directly accessible to the micromachine, microroutine invocations and returns will not cause page faults, and microroutines running in monitor mode will run to completion unless they themselves perform an action which causes them to give up JP 10114. Finally, the monitor micromachine's stack is not associated with a Process 610's Secure Stack 10336, and therefore, the monitor micromachine can both execute functions for Processes 610 and execute functions (which are related to no Process 610, for example,) the binding and removal of Virtual Processors 612 from JP 10114.

The description which follows first gives an overview of the devices which make up the micromachine, continues with descriptions of invocations on the micromachine and micromachine programming, and concludes with detailed discussions of the virtual and monitor modes and an overview of the relationship between the micromachine and CS 10110 subsystems. The manner in which the micromachine performs specific operations such as SIN parsing, Name resolution, or address translation may be found in previous descriptions of CS 10110 components which the micromachine uses to perform the operations.

b. Overview of Device Comprising FU Micromachine (Fig. 270)

Fig. 270 presents an overview of the devices comprising the micromachine. Fig. 270 is based on Fig. 201, but has been simplified to improve the clarity of the discussion. Devices and subdivisions of the micromachine which appear in Fig. 201 have the numbers given them in that figure. When a device in Fig. 270 appears in two subdivisions, it is shared by those subdivisions.

Fig. 270 has four main subdivisions. Three of them are from Fig. 201: FUCTL 20214, which contains the devices used to select the next microinstruction to be executed by the micromachine, DESP 20210, which contains stack and global registers and ALUs for descriptor processing; and MEMINT 20212, which contains the devices which translate Names into logical descriptors and logical descriptors into physical descriptors. The fourth subdivision, EU Interface 27007, represents those portions of EU 10122 which may be manipulated by FU 10120 microcode.

Fig. 270 further subdivides FUCTL 20214 and MEMINT 20212. FUCTL 20214 has four subdivisions:

- I-Stream Reader 27001, which contains the devices used to obtain SINs and parse them into SOPs and Names.
- SOP Decoder 27003, which translates SOPs into locations in FU microcode (FUSITT 11012), and in some cases EU microcode (EUSITT 20344), which contain the microcode that performs the corresponding SINs.
- Microcode Addressing 27013, which determines the location of the next microinstruction to be executed in FUSITT 11012.
- Register Addressing 27011, which contains devices which generate addresses for GRF 10354 registers.

MEMINT 20212 also has three subdivisions:

- Name Translation Unit 27015, which contains devices which accelerate the translation of Names into logical descriptors.
- Memory Reference Unit 27017, which contains devices which accelerate the translation of logical descriptors into physical descriptors.
- Protection Unit 27019, which contains devices which accelerate primitive access checks on memory references made with logical descriptors.

Fig. 270 also simplifies the bus structure of Fig. 202 by combining LENGTH Bus 20226, OFFSET Bus 20228, and AONR Bus 20230 into a single structure, Descriptor Bus (DB) 27021. In addition, internal bus connections have been reduced to those necessary for explaining the logical operation of the

Petitioner Apple Inc. - Exhibit 1004, p. 5288

micromachine. The following discussion first describes those devices used by most microcode executing on FU 10120, and then describes devices used to perform special functions, such as Name translation or protection checking.

1. Devices used by Most Microcode

The subdivisions of the micromachine which contain devices used by most microcode are Microcode Addressing 27013, Register Addressing 27011, DESP 20210, and EU Interface 27007. In addition, most microcode uses MOD Bus 10144, JPD Bus 10142, and DB Bus 27021. The discussion begins with the buses and then describes the other devices in the above order.

a.a. MOD Bus 10144, JPD Bus 10142, and DB Bus 27021

MOD Bus 10144 is the only path by which data may be obtained from MEM 10112. Data on MOD Bus 10144 may have as its destination Instruction Stream Reader 27001, DESP 20210, or EU Interface 27007. In the first case, the data on MOD Bus 10144 consists of SINs; in the second, it is data to be processed by FU 10120, and in the third, it is data to be processed by EU 10122. In the present embodiment, data to be processed by FU 10120 is generally data which is destined for internal use in FU 10120, for example in Name Cache 10226. Data to be processed by EU 10122 is generally operands represented by Names in SINs.

JPD Bus 10142 has two uses: it is the path by which data returns to MEM 10112 after it has been processed by JP 10114, and it is the path by which data other than logical descriptors moves between the subdivisions of the micromachine. For example, when CS 10110 is initialized, the microinstructions which are loaded into FUSITT 11012 are transferred from MEM 10112 to DESP 20210 via MOD Bus 10144, and from DESP 20210 to FUSITT 11012 via JPD Bus 10142.

DB 27021 is the path by which logical descriptors are transferred in the micromachine. DB 27021 connects Name Translation Unit 27015, DESP 20210, Protection Unit 27019, and Memory Reference Unit 27017. Typically, a logical descriptor is obtained from Name Translation Unit 27015, placed in a register in DESP 20210, and then presented to Protection Unit 27019 and Memory Reference Unit 27017 whenever a reference is made using a logical descriptor. However, DB 27021 is also used to transmit cache entries fabricated in DESP 20210 to ATU 10228, Name Cache 10226 and Protection Cache 10234.

b.b. Microcode Addressing

As discussed here, microcode addressing is comprised of the following devices: Timers 20296, Event Logic 20284, RCWS 10358, BRCASE 20278, mPC 20276, MCW0 20292, MCW1 20290, SITTNAS 20286, and FUSITT 11012. All of these devices have already been described in detail, and they are discussed here only as they affect microcode addressing. Other devices contained in Fig. 202, State Registers 20294, Repeat Counter 20280, and PNREG 20282 are not directly relevant to microcode addressing, and are not discussed here.

As has already been described in detail, devices in Microcode Addressing 27013 are loaded from JPD Bus 10142. The microcode addresses provided by these devices and by FUSDT 11010 are transmitted among the devices and to FUSITT 11012 by CSADR Bus 20204. There are six ways in which the next microcode address may be obtained:

- Most commonly, the value in mPC 20276 is incremented, by 1 by a special ALU in mPC 20276, thus yielding the address of the microinstruction following the current microinstruction.
- If a microinstruction specifies a call to a microroutine or a branch, the microinstruction contains a literal which an ALU in BRCASE 20278 adds to the value in mPC 20276 to obtain the location of the next microinstruction.
- If a microinstruction specifies the use of a case value to calculate the location of the next microinstruction, BRCASE 20278 adds a value calculated by DESP 20210 to the value in mPC 20716. The value calculated by DESP 20210 may be obtained from a field of a logical descriptor, thus allowing the micromachine to branch to different locations in microcode on the basis of type information contained in the logical descriptor. On return from an invocation of a microroutine, the location at which execution of the microroutine in which the invocation occurred is to continue is obtained from RCWS 10358.
- At the beginning of the execution of an SIN, the location at which the microcode for the SIN begins is obtained from the SIN's SOP by means of FUSDT 11010.
- Certain hardware signals cause invocations of microroutines. There are two classes of such signals: Event signals, which Event Logic 20284 transforms into invocations of certain microroutines, and JAM signals, which are translated directly into locations in microcode.

The addresses obtained as described above are transmitted to SITTNAS 20286, which selects one of the addresses as the location of the next microinstruction to be executed and transmits the location to FUSITT 11012. As the location is transmitted to FUSITT 11012, it is also stored in mPC 20276. All addresses except those for Jams are tranferred to SITTNAS 20286 via CSADR Bus 20204. Addresses obtained from

Petitioner Apple Inc. - Exhibit 1004, p. 5289

JAM signals are transferred by separate lines to SITTNAS 20286.

As will be explained in detail below, microroutine calls and returns also involve pushing and popping micromachine stack frames and saving state contained in MCW1 20290.

Register Addressing 27011 controls access to micromachine registers contained in GRF 10354. As explained in detail below, GRF 10354 contains both registers used for the micromachine stack and global registers, that is, registers that are always accessible to all microroutines. The registers are grouped in frames, and individual registers are addressed by frame number and register number. Register Addressing 27011 allows addressing of any frame and register in the GRs 10360 of GRF 10354, but allows addressing of registers in only three frames of the SR's 10362: the current (top) frame, the previous frame (i.e., the frame preceding the top frame), and the bottom frame, that is, the lowest frame in a virtual micromachine stack which is still contained in GRF 10354. The values provided by Register Addressing 27011 are stored in MCW0 20292. As will be explained in the discussion of microroutine invocations which follows, current and previous are incremented on each invocation and decremented on each return.

c.c. Description Processor 20210 (Fig. 271)

DESP 20210 is a set of devices for storing and processing logical descriptors. The internal structure of DESP 20210's processing devices has already been explained in detail; here, the discussion deals primarily with the structure and contents of GRF 10354. In a present embodiment of CS 10110, GRF 10354 contains 256 registers. Each register may contain a single logical descriptor. Fig. 271 illustrates a Logical Descriptor 27116 in detail. In a present embodiment of CS 10110, a Logical Descriptor 27116 has four main fields:

- RS Field 27101, which contains various flags which are explained in detail below.
- AON Field 27111, which contains the AON portion of the address of the data item represented by the Logical Descriptor 27116.
- OFF Field 27113, which contains the offset portion of the address of the data item represented by Logical Descriptor 27116.
- LEN Field 27115, which contains the length of the data item represented by the Logical Descriptor 27116.

RS Field 27101 has subfields as follows:

- RTD Field 27103 and WTD Field 27105 may be set by microcode to disable certain Event signals provided for debuggers by CS 10110. For details, see a following description of debugging aids in CS 10110.
- FIU Field 27107 contains two bits. The fields are set from information in the Name Table Entry used to construct the Logical Descriptor 27116. The bits determine how the data specified by the Logical Descriptor 27116 is to be justified and filled when it is fetched from MEM 10112.
- TYPE Field 27109's four bits are also obtained from the Name Table Entry used to construct the Logical Descriptor 27116. The field's settings vary from S-Language to S-Language, and are used to communicate S-Language-specific type information to the S-Language's S-Interpreter microcode.

The four fields of a Logical Descriptor 27116 are contained in three separately-accessible fields in a GRF 10354 register: one containing RS Field 27101 and AON Field 27111, one containing OFF Field 27113, and one containing LEN Field 27115. In addition, each GRF 10354 register may be accessed as a whole. GRF 10354 is further subdivided into 32 frames of eight registers each. An individual GRF 10354 register is addressed by means of its frame number and its register number within the frame. In a present embodiment of CS 10110, half of the frames in GRF 10354 belong to SR's 10362 and are used for micromachine stacks, and half belong to GRs 10360 for storing "global information". In SR's 10362, each GRF 10354 frame contains information belonging to a single invocation of a microroutine. As previously explained, Register Addressing 27011 allows addressing of only three GRF 10354 frames in SR's tack 10362, the current top frame in the stack, the previous frame, and the bottom frame. Registers are accessed by specifying one of these three frames and a register number.

The global information contained in GRs 10360, is information which is not connected with a single invocation. There are three broad categories of global information:

- Information belonging to Process 610 whose Virtual Processor 612 is currently bound to JP 10114. Included in this information are the current values of Process 610's ABPs and the pointers which KOS uses to manage Process 610's stacks.
- Information required for the operation of KOS. Included in this information are such items as pointers to KOS data bases which occupy fixed locations in MEM 10112.
- Constants, that is, fixed values required for certain frequently performed operations in FU 10120.

Remaining registers are available to microprogrammers as temporary storage areas for data which cannot be stored in a microroutine's stack frame. For example, data which is shared by several microroutines may best be placed in a GR 10360. Addressing of registers in the GRs 10360 of GRF 10354 requires two values: a value of 0 through 15 to specify the frame and a value of 0 through 7 to specify the register in the frame.

As previously discussed in detail, each of the three components AONP 20216, OFFP 20218, and LENP 20220 of DESP 20210 also contains ALUs, registers, and logic which allows operations to be performed on individual fields of GRF 10354 registers. In particular, OFFP 20218 contains OFFALU 20242, which may be

Petitioner Apple Inc. - Exhibit 1004, p. 5290

used as a general purpose 32 bit arithmetic and logical unit. OFFALU 20242 may further serve as a source and destination for JPD Bus 10142, the offset portion of DB 27021, and NAME Bus 20224, and as a destination for MOD Bus 10144. Consequently, OFFALU 20242 may be used to perform operations on data on these buses and to transfer data from one bus to another. For example, when an SIN contains a literal value used in address calculation, the literal value is transferred via NAME Bus 20224 to OFFALU 20242, operated on, and output via the offset portion of DB 27021.

d.d. EU 10122 Interface

FU 10120 specifies what operation EU 10122 is to perform, what operands it is to perform it on, and when it is finished, what is to be done with the operands. FU 10120 can use two devices in EU 10122 as destinations for data, and one device as a source for data. The destinations are COMQ 20342 and OPB 20322. COMQ 20342 receives the location in EUSITT 20344 of the microcode which is to perform the operation desired by the FU 10120. COMQ 20342 may receive the location in microcode either from an FU 10120 microroutine or from an SIN's SOP. In the first case, the location is transferred via JPD Bus 10142, and in the second, it is obtained from EUSDT 20266 and transferred via EUDIS Bus 20206. OPB 20322 receives the operands upon which the operation is to be performed. If the operands come directly from MEM 10112, they are transferred to OPB 20322 via MOD Bus 10144; if they come from registers or devices in FU 10120, they are transferred via JPD Bus 10142.

Result Register 27013 is a source for data. After EU 10122 has completed an operation, FU 10120 obtains the result from Result Register 27013. FU 10120 may then place the result in MEM 10112 or in any device accessible from JPD Bus 10142.

2. Specialized Micromachine Devices

Each of the groups of specialized devices serves one of CS 10110's subsystems. I-Stream Reader 27001 is part of the S-Interpreter subsystem, Name Translation Unit 27015 is part of the Name Interpreter subsystem, Memory Reference Unit 27017 is part of the Virtual Memory Management System, and Protection Unit 27019 is part of the Access Control System. Here, these devices are explained only in the context of the micromachine; for a complete understanding of their functions within the subsystems to which they belong, see previous descriptions of the subsystems.

a.a. I-Stream Reader 27001

I-Stream Reader 27001 reads and parses a stream of SINs (termed the I-Stream) from a Procedure Object 604, 606, 608. The I-Stream consists of SOPs (operation codes), Names, and literals. As previously mentioned, in a present embodiment of CS 10110, the I-Stream read from a given Procedure 602 has a fixed format: the SOPs are 8 bits long and the Names and literals all have a single length. Depending on the procedure, the length may be 8, 12, or 16 bits. I-Stream Reader 27001 parses the I-Stream by breaking it up into its constituent SOPs and Names and passing the SOPs and Names to appropriate parts of the micromachine. I-Stream Reader 27001 contains two groups of devices:

— PC Values 27006, which is made up of three registers which contain locations in the I-Stream. When added to ABP PBP, the values contained in these registers specify locations in Procedure Object 901 containing the Procedure 602 being executed. CPC 20270 contains the location of the SOP or Name currently being interpreted; IPC 20272 contains the location of the beginning of the SIN currently being executed; EPC 20274, finally, is of interest only at the beginning of the execution of an SIN; at that time, it contains the location of the last SIN to be executed.

— Parsing Unit 27005, which is made up of INSTB 20262, PARSER 20264, and PREF 20260. The micromachine uses PREF 20260 to create Logical Descriptors 27116 for the I-Stream, which are then placed on DB Bus 27021 and used in logical memory references. The data returned from these references is placed in INSTB 20262, and parsed by PARSER 20264.

SOPs, Names, and literals obtained by PARSER 20264 are placed on NAME Bus 20224, which connects PARSER 20264, SOP Decoder 27003, Name Translation Unit 27015, and OFFALU 20242.

b.b. SOP Decoder 27003

SOP Decoder 27003 decodes SOPs into locations in FU 10120 and EU 10122 microcode. SOP Decoder 27003 comprises FUSDT 11010, EUSDT 20266, Dialect Register (RDIAL) 24212, and LOPDCODE 24210. FUSDT 11010 are further comprised of FUDISP 24218 and FALG 24220. The manner in which these devices translate SOPs contained in SINs into locations in FUSITT 11012 and EUSITT 20344 has been previously described.

c.c. Name Translation Unit 27015

Name Translation Unit 27015 accelerates the translation of Names into Logical Descriptors 27116. This operation is termed name resolution. It is comprised of two components: NC 10226 and Name Trap 20254. NC 10226 contains copies of information from a Procedure Object 604's Name Table 10350, and thereby makes it possible to translate Names into Logical Descriptors 27116 without referring to Name Table 10350. When a Name is presented to Name Translation Unit 27015, it is latched into Name Trap 20254 for later use by Name Translation Unit 27015 if required. As will be explained in detail later, in the present embodiment,

Petitioner Apple Inc. - Exhibit 1004, p. 5291

Name translation always begins with the presentation of a Name to NC 10226. If the Name has already been translated, the information required to construct its Logical Descriptor 27116 may be contained in NC 10226. If there is no information for the Name in NC 10226, Name Resolution Microcode obtains the Name from Name Trap 20254, uses information from Name Table 10350 for the procedure being executed to translate the Name, places the required information in NC 10226, and attempts the translation again. When the translation succeeds, a Logical Descriptor 27116 corresponding to the Name is produced from the information in Name Cache 10115, placed on DB Bus 27021, and loaded into a GRF 10354 register.

d.d. Memory Reference Unit 27017

Memory Reference Unit 27017 performs memory references using Logical Descriptors 27116. Memory Reference Unit 27017 receives a command for MEM 10112 and a Logical Descriptor 27116 describing the data upon which the command is to be performed. In the case of a write operation, Memory Reference Unit 27017 also receives the data being written via JPD Bus 10142. Memory Reference Unit 27017 translates Logical Descriptor 27116 to a physical descriptor and transfers the physical descriptor and the command to MEM 10112 via PD Bus 10146. A Memory Reference Unit 27017 has four components: ATU 10228, which contains copies of information from KOS virtual memory management system tables, and thereby accelerates logical-to-physical descriptor translation; Descriptor Trap 20256, which traps Logical Descriptors 27116, Command Trap 27018, which traps memory commands; and Data Trap 20258, which traps data on write operations. When a logical memory reference is made, a Logical Descriptor 27116 is presented via DB Bus 27021 to ATU 10228, and at the same time, Logical Descriptor 27116 and the memory command are trapped in Descriptor Trap 20256 and Command Trap 27018. On write operations, the data to be written is trapped in Data Trap 20258. If the information needed to form the physical descriptor is present in ATU 10228, the physical descriptor is transferred to MEM 10112 via PD Bus 10146. If the information needed to form the physical descriptor is not present in ATU 10228, an Event Signal from ATU 10228 invokes a microroutine which retrieves Logical Descriptor 27116 from Descriptor Trap 20256 and uses information contained in KOS virtual memory management system tables to make an entry in ATU 10228 for Logical Descriptor 27116. When the microroutine returns, the logical memory reference is repeated using Logical Descriptor 27116 from Descriptor Trap 20256, the memory command from Command Trap 27018, and on write operations, the data in Data Trap 20258. As will be described in detail in the discussion of virtual memory management, if the data referenced by a logical memory reference is not present in MEM 10112, the logical memory reference causes a page fault.

e.e. The Protection Unit 27019

On each logical memory reference, Protection Unit 27019 checks whether the subject making the reference has access rights which allow it to perform the action specified by the memory command on the object being referenced. If the subject does not have the required access rights, a signal from Protection Unit 27019 causes MEM 10112 to abort the logical memory reference. Protection Unit 27019 consists of Protection Cache 10234, which contains copies of information from KOS Access Control System tables, and thereby speeds up protection checking, and shares Descriptor Trap 20256, Command Trap 27018, and Data Trap 20258 with Memory Reference Unit 27017. When a logical memory reference is made, the AON and offset portions of the logical descriptor are presented to Protection Cache 10234. If Protection Cache 10234 contains protection information for the object specified by the AON and offset and the subject performing the memory reference has the required access, the memory reference may continue; if Protection Cache 10234 contains protection information and the subject does not have the required access, a signal from Protection Cache 10234 aborts the memory reference. If Protection Cache 10234 does not contain the required access information, a signal from Protection Cache 10234 aborts the memory reference and invokes a microroutine which obtains the access information from KOS Access Control System tables and places it in Protection Cache 10234. When Protection Cache 10234 is ready, the memory access is repeated, using the logical descriptor from Descriptor Trap 20256, the memory command from Command Trap 27018, and in the case of write operations, the data in Data Trap 20258.

f.f. KOS Micromachine Devices

As mentioned in the above introduction to the micromachine, the devices making up the micromachine may be divided into two classes: those which any microcode written for the micromachine may manipulate, and those which may be manipulated exclusively by KOS microcode. The latter class consists of certain registers in GRs 10360 of GRF 10354, the bottom frame of the portion of the virtual micromachine stack in the stack portion (Stack Registers 10362) of GRF 10354, and the devices contained in Protection Unit 27019 and Memory Reference Unit 27017. Because Protection Unit 27019 and Memory Reference Unit 27017 may be manipulated only by KOS microcode, non-KOS microcode may not use Descriptor Trap 20256 or Command Trap 27018 as a source or destination, may not load or invalidate registers in ATU 10228 or Protection Cache 10234, and may not perform physical memory references, i.e., memory references which place physical descriptors directly on PD Bus 10146, instead of presenting logical

Petitioner Apple Inc. - Exhibit 1004, p. 5292

descriptors to Memory Reference Unit 27017 and Protection Unit 27019. Similarly, non-KOS microcode may not specify KOS registers in the GRs 10360 of GRF 10354 or the bottom frame of the stack portion of GRF 10354 when addressing GRF 10354 registers. Further, in embodiments allowing dynamic loading of FUSITT 11012, only KOS microcode may manipulate the devices provided for dynamic loading.

In a present embodiment of CS 10110, the distinction between KOS devices and registers and devices and registers accessible to all microprograms is maintained by the microbinder. The microbinder checks all microcode for microinstructions which manipulate devices in Protection Unit 27019, or Memory Reference Unit 27017, or which address GRF 10354 registers reserved for KOS use. However, it is characteristic of the micromachine that KOS devices are logically and physically separate from devices accessible to all microprograms and, consequently, other embodiments of CS 10110 may use hardware devices to prevent non-KOS microprograms from manipulating KOS devices.

c. Micromachine Stacks and Microroutine Calls and Returns (Figs. 272, 273)

1. Micromachine Stacks (Fig. 272)

As previously mentioned, the FU micromachine is a stack micromachine The properties of the FU micromachine's stack depends on whether the FU micromachine is in virtual or monitor mode. In virtual mode, the micromachine stack is of essentially unlimited size; if it contains more frames than allowed for inside FU 10120, the top frames are in GRF 10354 and the remaining frames are in Secure Stack 10336 belonging to Process 610 being executed by the FU micromachine. In the following, the virtual mode micromachine stack is termed the virtual micromachine stack. In monitor mode, the micromachine stack consists of a fixed amount of storage; in a present embodiment of CS 10110, the monitor mode micromachine stack is completely contained in the stack portion, SRs 10362, of GRF 10354; in other embodiments of CS 10110, part or all of the monitor mode micromachine stack may be contained in an area of MEM 10112 which has a fixed size and a fixed location known to the monitor micromachine. In yet other embodiments of CS 10110, monitor mode micromachine stack may be of flexible depth in a manner similar to the virtual micromachine stack. In either mode, microroutines other than certain KOS microroutines which execute state save and restore operations may access only two frames of GRF 10354 stack: the frame upon which the microroutine is executing, called the current frame, and the frame upon which the microroutine that invoked that microroutine executed, called the previous frame. KOS microroutines which execute state save and restore operations may in addition access the bottom frame of that portion of the virtual micromachine stack which is contained in GRF 10354.

Fig. 272 illustrates stacks for the FU micromachine. Those portions of the micromachine stack which are contained in the FU are contained in SR's 10362 (of GRF 10354) and in RCWS 10358. Each register of RCWS 10358 is permanently associated with a GRF frame in SRs 10362 of GRF 10354, and the RCWS 10358 register and the GRF frame together may contain one frame of a micromachine stack. As previously describe, each register of GRF 10354 contains three fields: one for an AON and other information, one for an offset, and one for a length. As illustrated in Fig. 251, each register in RCWS 10358 contains four fields:

- A one bit field which retains the value of the Condition Code register in MCW1 20290 at the time that the invocation which created the next frame occurred.
- A field indicating what Event Signals were pending at the time that the invocation to which the RCWS register belongs invoked another microroutine.
- A flag indicating whether the microinstruction being executed when the invocation occurred was the first microinstruction in an SIN.
- The address at which the execution of the invoking microroutine is to continue.

The uses of these fields will become apparent in the ensuing discussion.

The space available for micromachine stacks in SRs 10362 and RCWS 10358 is divided into two parts: Frames 27205 reserved for MOS 10370 and Frames 27206 available for the MIS 27203. Frames 27206 may contain no MIS Frames 27203, or be partially or completely occupied by MIS Frames 27203. Space which contains no MIS Frames is Free Frames 27207. The size of the space reserved for Monitor Micromachine Stack Frames 27205 is fixed, and Spaces 27203, 27205, and 27207 always come in the specified order. Register Addressing 27011 handles addressing in Stack Portion 27201 of GRF 10354 and RCWS 10358 in such fashion that the values for the locations of current, previous, and bottom frames specifying registers in RCWS 10358 or frames in Stack Portion 27201 automatically "wrap around" when they are incremented beyond the largest index value allowed by the sizes of the registers or decremented below the smallest index value. Thus, though Spaces 27203, 27205, and 27207 always have the same relative order, their GRF 10354 frames and RCWS registers may be located anywhere in Stack Portion 27201 and RCWS 10358.

2. Microroutine Invocations and Returns

In CS 10110, microroutines may be invoked by other microroutines or by signals from CS 10110 hardware. The methods of invocation aside, microroutine invocations and returns resemble invocations of and returns from procedures written in high-level languages. In the following, the general principles of microroutine invocations and returns are discussed, and thereafter, the specific methods by which microroutines may be invoked in CS 10110. The differences between invocations in monitor mode and

Petitioner Apple Inc. - Exhibit 1004, p. 5293

invocations in virtual mode are explained in the detailed discussions of the two modes.

The microroutine which is currently being executed runs on the frame specified by Current Pointer 27215. When an invocation occurs, either because the executing microroutine performs a call, or because a signal which causes invocations has occurred, JP 10114 hardware does three things:

- It stores state information for the invoking microroutine in the RCWS 10358 register associated with the current frame. The state information includes the location at which execution of the invoking microroutine will resume, as well as other state information.
- It increments Current Pointer 27215 and Previous Pointer 27213, thereby providing a frame for the new invocation.
- It begins executing the first instruction of the newly invoked microroutine.

Because the newly-invoked microroutine can access registers of the invoking microroutine's frame, the invoking microroutine can pass "arguments" to the invoked microroutine by placing values in registers in its frame used by the invoked microroutine. However, the invoking microroutine cannot specify which registers contain "arguments" on an invocation, so the invoked microroutine must know which registers of the previous frame are used by the invoking microroutine. Since the only "arguments" which a microroutine has access to are those in the previous frame, a microroutine can pass arguments which it received from its invoker to a microroutine which it invokes only by copying the arguments from its invoker's frame to its own frame, which then becomes the newly-invoked routine's previous frame.

The return is the reverse of the above: Current Pointer 27215 and Previous Pointer 27213 are decremented, thereby "popping off" the finished invocation's frame and returning to the invoker's frame. The invoker then resumes execution at the location specified in the RCWS 10358 register and using the state saved in the RCWS 10358. The saved state includes the value of the Condition Code in MCW1 20290 at the time of the invocation and flags indicating various pending Events. The Condition Code field in MCW1 20290 is set to the saved value, and the pending event flags may cause Events to occur as described in detail below.

3. Means of Invoking Microroutines

In the micromachine, invocations may be produced either by commands in microinstructions or by hardware signals. In the following, invocations produced by commands in microinstructions are termed Calls, while those produced by hardware signals are termed Event invocations and Jams. Invocations are further distinguished from each other by the locations to which they return. Calls and Jams return to the microinstruction following the microinstruction in which the invocation occurs; Event invocations return to that microinstruction, which is then repeated.

In terms of implementation, the different return locations are a consequence of the point in the micromachine cycle at which Calls, Jams, and Event invocations save a return location and transfer control to the called routine. With Calls and Jams, these operations are performed in the M1 cycle; with Event invocations, on the other hand, the Event signal during the M0 cycle causes the M0 cycle to be followed by a MA cycle instead of the M1 cycle, and the operations are performed in the MA cycle. In the M1 cycle, the value in mPC 20276 is incremented; in the MA cycle, it is not. Consequently, the return value saved in RCWS 10358 on a Call or Jam is the incremented value of mPC 20276, while the return value saved on an Event invocation is the unincremented value of mPC 20276. The following discussion will deal first with Calls and Jams, and then with Event invocations.

A Call command in a microinstruction contains a literal value which specifies the offset from the microinstruction containing the Call at which execution is to continue after the Call. When the microinstruction with the Call command is executed in micromachine cycle M1, BRCASE 20278 adds the offset contained in the command to the current value of mPC 20276 in order to obtain the location of the invoked microroutine and sets SITTNAS 20286 to select the location provided by BRCASE 20278 as the location of the next microinstruction. Then the Call command increments mPC 20276 and stores the incremented value of mPC 20276 in the RCWS 10358 register associated with the current frame in SRs 10362 and increments Current Pointer 27215 and Previous Pointer 27213 to provide a new frame in SRs 10362. The Jam works exactly like the Call, except that a hardware signal during micromachine cycle M1 causes the actions associated with the invocation to occur and provides the location of the invoked microroutine directly to SITTNAS 20286.

With Events, Event Logic 20284 causes an invocation to occur during cycle M0 and provides the location of the invoked microroutine via CSADR 20299. Since the Event occurs during cycle M0, the location stored in RCWS 10358 is the unincremented value of mPC 20276, and SITTNAS 20286 selects the location provided by Event Logic 20284 as the location of the next microinstruction. Since the return from the Event causes the microinstruction during which the Event occurred to be re-executed, the microinstruction and the microroutine to which it belongs may be said to be "unaware" of the Event's occurrence. The only difference between the execution of a microinstruction during which an Event occurs and the execution of the same microinstruction without the Event is the length of time required for the execution.

Petitioner Apple Inc. - Exhibit 1004, p. 5294

4. Occurrence of Event Invocations (Fig. 273)

As described previously, Event invocations are produced by Event Logic 20284. The location in microcode to which Event Logic 20284 transfers control is determined by the following:

— The operation being commenced by FU 10120. Certain Event invocations may occur only at the beginning of certain FU 10120 operations.
— The state of Event signal lines from hardware and internal registers in Event Logic 20284.
— The state of certain registers visible via MCW1 20290. Some of these registers enable Events and others mask Events. Of the registers which enable Events, some are set by Event signals and others by the microprogram.
— On returns from invocations of microroutines, the settings of certain bits in the RCWS 10358 register belonging to the micromachine frame for the invocation that is being returned to.

Microprograms may use these mechanisms to disable Event signals and to delay an Event Invocation from an Event signal for a single microinstruction or an indefinite period, and FU 10120 uses them to automatically delay Event invocations resulting from certain Event signals. Using traditional programming terminology, the mechanisms allow a differential masking of Event signals. An Event signal may be explicitly masked for a single microinstruction, it may be masked for a sequence of microinstructions; it may be automatically masked until a certain operation occurs, or it may be automatically masked for a certain maximum length of time. Event signals which occur while they are masked are not lost. In some cases, the Event signal continues until it is serviced; in others, a register is set to retain the fact that the Event signal occurred. When the Event signal is unmasked, the set register causes the Event signal to reoccur. In some cases, finally, the Event signal is not retained, but recurs when the microinstruction which caused it is repeated.

In the following, the relationship between FU 10120 operations and Event signals is first presented, and then a detailed discussion of the enabling registers in MCW1 20290 and of the bits in RCWS 10358 registers which control Event invocations.

FU 10120 allows Event invocations resulting from Event signals to be inhibited for a single microinstruction; it also delays certain Event invocations for certain Event signals until the first microinstruction of an SIN. Other Event signals occur only at the beginning of an SIN, at the beginning of a Namespace Resolve or Evaluate operation, or at the beginning of a logical memory reference.

Event Invocations may be delayed for a single microinstruction by setting a field of the microinstruction itself. Setting this field delays almost all Event invocations, and thereby guarantees that an Event invocation will not occur during the microinstruction's M0 cycle.

Event signals relating to debugging occur at the beginnings of certain micromachine operations. Such Event signals are called Trace Event signals. As will be explained in detail, in the discussion of the debugger, Trace Event signals can occur on the first microinstruction of an SIN, at the beginning of an Evaluate or Resolve operation, at the beginning of a logical memory reference, or at the beginning of a microinstruction. IPM interrupt signals and Interval Timer Overflow Event signals are automatically masked until the beginning of the next SIN or until a maximum amount of time has elapsed, which ever occurs first. The mechanisms involved here are explained in detail in the discussion of interrupt handling in the FU 10120 micromachine.

Turning now to the registers used to mask and enable Event signals, Fig. 273 is a representation of the masking and enabling registers in MCW1 20290 and of the field in RCWS 10358 registers which controls Event invocations. Beginning with the registers in MCW1 20290, there are three registers which control Event invocations: Event Mask Register (EM) 27301, Events Pending Register (EP) 27309, and Trace Enable Register (TE) 27319. Bits in EM 27301 mask certain Event signals as long as they are set; bits in EP Register 27309 record the occurrence of certain Event signals while they are masked; when bits in TE Register 27319 are set, Trace Event signals occur before certain FU 10120 operations.

EM 27301 contains three one bit fields: Asynchronous Mask Field 27303, Monitor Mask Field 27305, and Trace Event Mask Field 27307. As explained in detail in the discussion of FU 10120 hardware, these bits establish a hierarchy of Event masks. If Asynchronous Mask Field 27303 is set, only two Event signals are masked: that resulting from an overflow of EGGTMR 25412 and that resulting from an overflow of EU 10122's stack. If Monitor Mask Field 27305 is set, those Events are masked, and additionally, the FU Stack Overflow Event signal is masked. As will be explained in detail later, when the FU 10120 Stack Overflow Event signal is masked, the FU micromachine is executing in monitor mode. If Trace Event Mask Field 27307 is set, Trace Trap Event signals are masked in addition to the above signals. Each of the fields in EM 27301 may be individually set and cleared by the microprogram.

Four Event signals set fields in EP 27309: the EGGTMR 25412 Runout signal sets ET Field 27311, the INTTMR 25410 Runout signal sets IT Field 27313, the Non-Fatal Memory Error signal sets ME Field 27315, and the Inter-Process Message signal sets IPM Field 27317. Event invocations for all of these Event signals but the Egg Timer Runout signal occur at the beginning of an SIN; in these cases the fields in EP 27309 retain the fact that the Event signal has occurred until that time; the Event invocation for the Egg Timer Runout signal occurs as soon after the signal as the settings of mask bits in EM 27301 allow. The bit in ET Field 27311 retains the fact of the Egg Timer Runout signal until the masking allows the Event invocation to occur. All of the fields in EP 27309 but ME Field 27315 may be reset by microcode. The microroutines invoked by the Events must reset the appropriate fields; otherwise, they will be reinvoked when they

Petitioner Apple Inc. - Exhibit 1004, p. 5295

return. ME Field 27315 is automatically reset when the memory error is serviced.

TE Register Field 27319 enables tracing. Each bit in the register enables a kind of Trace Event signal when it is set. Depending on the kind of tracing, the Trace Event signal occurs at the beginning of an SIN, at the beginning of a Resolve or Evaluate operation, at the beginning of a logical memory reference, or at the beginning of a microinstruction. For details, see the following description of debugging.

Turning now to the registers contained in RCWS 10358, each RCWS Register 27322 contains eight fields which control Event signals. The first field is FM Field 27323. FM Field 27323 reflects the value of a register in Event Logic 20284 when the invocation to which RCWS Register 27322 belongs occurs. The register in Event Logic 20284 is set only when the microinstruction currently being executed is the first microinstruction of an SIN. Thus, FM Field 27323 is set only in RCWS Registers 27322 belonging to Event invocations which occur in the M0 cycle of the first microinstruction in the SIN, i.e., at the beginning of the SIN. The value of the register in Event Logic 20284 is saved in FM Field 27323 because several Event invocations may occur at the beginning of a single SIN. The Event invocations occur in order of priority: when the one with the highest priority returns, the fact that FM Field 27323 is set causes the register in Event Logic 20284 to again be set to the state which it has on the first microinstruction of an SIN. The register's state, thus set, causes the next Event invocation which must occur at the beginning of the SIN to take place. After all such invocations are finished, the first microinstruction enters its M1 cycle and resets the register in Event Logic 20284. In its reset state, the register inhibits all Event invocations which may occur only at the beginning of an SIN. It is again set at the beginning of the next SIN.

The remaining fields in RCWS Register 27322 which control Event invocations are the fields in Return Signals Field 27331. These fields allow the information that an Event signal has occurred to be retained through Event invocations until the Event signal's Event invocation takes place. When an invocation occurs, these fields are set by Event Logic 20284. On return from the invocation, the values of the fields are input into Event Logic 20284, thereby producing Event signals. The Event signal with the highest priority results in an Event invocation, and the remaining Event signals set fields in Return Signals Field 27331 belonging to RCWS Register 27322 belonging to the invocation which is being executed when the Event signals occur. Because the fields in Return Signals Field 27330 are input into Event Logic 20284, microcode invoked as a consequence of Event signals which sets one of these fields must reset the field itself. Otherwise, the return from the microcode will simply result in a reinvocation of the microcode.

The seven fields in Return Signals Field 27330 have the following significance:

- When EG Field 27333 is set, an EU 10122 dispatch operation produced an illegal location in EU 10122 microcode EUSITT 20344.
- When NT Field 27335, ST Field 27341, mT Field 27343, or mB Field 27345 is set, a trace signal has occurred. These are explained in detail in the discussion of debugging.
- When ES Field 27337 is set, an EU 10122 Storeback Exception has occurred, i.e., an error occurred when EU 10122 attempted to store the result of an operation in MEM 10112.
- When MRR Field 27339 is set, a condition such as an ATU 10228 miss or a Protection Cache 10234 miss has occurred, and it is necessary to reattempt a memory reference.

d. Virtual Micromachines and the Monitor Micromachine

As previously described, microcode being executed on FU 10120's micromachine can run in either monitor mode or virtual mode. In this portion of the discussion, the distinguishing features and applications of the two modes are explained in detail.

1. Virtual Mode

As previously mentioned, the chief distinction between virtual mode and monitor mode is MIS 10368. The fact that MIS 10368 is of essentially unlimited size has the following consequences for microroutines which execute in virtual mode.

- An invocation of a microroutine executing in virtual mode may have as its consequence further invocations to any depth.
- Any invocation of or return from a microroutine executing in virtual mode may cause a page fault.

The FU micromachine is in virtual mode when all bits in the Event Masks portion of MCW1 20290 are cleared. In this state, no enabled Event signals are masked, and Event invocations may occur in any microinstruction which does not itself mask them.

Because invocations may occur to any depth in virtual mode, microroutines executing in this mode may be recursive. Such recursive microroutines are especially useful for the interpretation of Names. Often, as previously described, the Name Table Entry for a Name will contain Names which resolve to other Names, and the virtual micromachine's limitless stack allows the use of recursive Name Resolution microroutines in such situations. Recursive microroutines may also be used for complex SINs, such as Calls.

Because invocations can occur to any depth, any number of Events may occur while a microroutine is executing in monitor mode. This in turn greatly simplifies Event handling. If an Event signal occurs while an Event with a given priority is being handled and the Event being signalled has a higher priority than the one

Petitioner Apple Inc. - Exhibit 1004, p. 5296

being handled, the result is simply the invocation of the new Event's handler. Thus, the order in which the Event handlers finish corresponds exactly to the priorities of their Events: those with the highest finish first.

A page fault may occur on any microinvocation or return executed in virtual mode because an invocation in virtual mode which occurs when there are no more Free Frames 27207 on SRs 10362 causes an Event signal which invokes a microroutine running in monitor mode. The microroutine transfers MIS Frames 27203 from GRF 10354 to Secure Stack 10336 in MEM 10112, and the transfer may cause a page fault. Similarly, when a microreturn takes place from the last frame on MIS Frames 27203 on SRs 10362, an Event signal occurs which invokes a microroutine that transfers additional frames from Secure Stack 10336 to GRF 10354, and this transfer, too, may cause a page fault.

The fact that page faults may occur on microinvocations or microreturns in virtual mode has two important consequences: microroutines which cannot tolerate page faults other than those explicitly generated by the microroutine itself cannot execute in virtual mode, and because unexpected page faults cause execution to become indeterminate, microroutines which must run to completion cannot execute in virtual mode. For example, if the microroutine which handles page faults executed in virtual mode, its invocation could cause a page fault, which would cause the microroutine to be invoked again, which would cause another page fault, and so on through an infinite series of recursions.

2. Monitor Micromachine

As previously described, the essential feature of monitor mode is MOS 10370. In a present embodiment of CS 10110, this stack has a fixed minimum size, and is always contained in GRF Registers 10354. The nature of MOS 10370 has four consequences for microroutines which execute in monitor mode:

- When the micromachine is in monitor mode, the depth of invocations is limited; recursive microroutines therefore cannot be executed in monitor mode, and Event invocations must be limited.
- Invocations of microroutines or returns from microroutines in monitor mode never result in page faults.
- Microroutines executing in monitor mode are guaranteed to run to completion if they do not suspend the Process 610 which they are executing or perform a Call to software.
- When the micromachine is executing in monitor mode, it is guaranteed to return to virtual mode within a reasonable period of time, either because a microroutine executing in monitor mode has run to completion, or because the microroutine has suspended the Process 610 which it is executing, or has made a Call to software. The result in both cases is the execution of a new sequence of SOPs, and thus a return to virtual mode.

In a present embodiment of CS 10110, the FU micromachine is in monitor mode when a combination of masking bits in MCW1 20290 is set which results in the masking of the FU Stack Overflow Event and the Egg Timer Overflow Event. As previously described, these Events are masked if Fields 27303, 27305, or 27307 is set. These Events and the consequences of masking them are explained in detail below.

The event signal for the FU Stack Overflow Event occurs on microinvocations for which there is no frame available in MIS Frames 27203. If the Event signal is not masked, it causes the invocation of a microroutine which moves MIS Frames from MIS Frames 27203 onto a Process 610's Secure Stack 10336. When the FU Stack Overflow Event is masked, all frames in SRs 10362 of GRs 10360 are available for microroutine invocations and microroutine invocations will not result in page faults, but if the capacity of SRs 10362 is exceeded, FU 10120 ceases operation.

The Egg Timer Overflow event signal occurs when Egg TMR 25412 runs out. As will be explained in detail later, Egg TMR 25412 ensures that an Interval Timer Runout, an Inter-processor Message, or a Non-fatal Memory Error will be serviced by JP 10114 within a reasonable amount of time. If an Interval Timer Runout Event signal or an Inter-processor Message Event signal occurs at a time when it is inefficient for the FU micromachine to handle the Event, Egg TMR 25412 begins running. When Egg TMR 25412 runs out, the Event is handled unless the micromachine is in monitor mode. If the Egg TMR 25412 Runout Event signal occurs while the FU micromachine is in monitor mode, i.e., while the Event is masked, the Event signal sets Field 27311 in MCW1 20290. When the FU micromachine reverts to virtual mode, i.e., when all Event Mask bits in MCW1 20290 are cleared, the Egg TMR 25412 Runout Event occurs, and the Interval Timer Runout Inter-processor Message Event handlers are invoked by Event Logic 20284.

e. Interrupt and Fault Handling

1. General Principles

Any computer system must be able to deal with occurrences which disrupt the normal execution of a program. Such occurrences are generally divided into two classes: faults and interrupts. A fault occurs as a consequence of an attempt to execute a machine instruction, and its occurrence is therefore synchronous with the machine instruction. Typical faults are floating point overflow faults and page faults. A floating point overflow fault occurs when a machine instruction attempts to perform a floating point arithmetic operation and the result exceeds the capacity of the CS 10110's floating point hardware, that is EU 10122. A page fault occurs when a machine instruction in a computer system with virtual memory attempts to reference data which is not presently available in the computer system's primary memory, that is MEM

Petitioner Apple Inc. - Exhibit 1004, p. 5297

10112. Since faults are synchronous with the execution of machine instructions and in many cases the result of the execution of specific machine instructions, their occurrence is to some extent predictable.

The occurrence of an interrupt is not predictable. An interrupt occurs as a consequence of some action taken by the computer system which has no direct connection with the execution of a machine instruction by the computer system. For example, an I/O interrupt occurs when data transmitted by an I/O device (IOS 10116) reaches the central processing unit (FU 10120), regardless of the machine instruction the central processing unit is currently executing.

In conventional systems, interrupts and faults have been handled as follows: if an interrupt or fault occurs, the computer system recognizes the occurrence before it executes the next machine instruction and executes an interrupt-handling microroutine or Procedure 602 instead of the next machine instruction. If the interrupt or fault cannot be handled by the Process 610 in which it occurs, the interrupt or fault results in a process swap. When the interrupt handling routine is finished, Process 610 which faulted or was interrupted can be returned to the CPU if it was removed and the next machine instruction executed.

While the above method works well with faults, the fact that interrupts are asynchronous causes several problems:

— Machine instructions cannot require an indefinite amount of time to execute, since interrupts cannot be handled until the machine instruction during which they occur is finished.
— It must be possible to remove a Process 610 from the CPU at any time, since the occurrence of an interrupt is not predictable. This requirement greatly increases the difficulty of process management.

The method used for interrupt and fault handling in a present embodiment of CS 10110 is described below.

2. Hardware Interrupt and Fault Handling in CS 10110

In CS 10110, there are two levels of interrupts: those which may created and dealt with completely by software, and those which may created by hardware signals. The former class of interrupts is dealt with in the discussion of Processes 610; the latter, termed hardware interrupts, is discussed below.

In CS 10110, hardware interrupts and faults begin as invocations of microroutines in FU 10120. The invocations may be the result of Event signals or may be made by microprograms. For example, when IOS 10116 places data in MEM 10112 for JP 10114, an Inter-processor Message Event signal results, and the signal causes the invocation of Inter-processor Message Interrupt handler microcode. On the other hand, a Page Fault begins as an invocation of Page Fault microcode by LAT microcode. The actions taken by the microcode which begins handling the fault or interrupt depend on whether the fault or interrupt is handled by the Process 610 which was being executed when the fault or Event occurred or by a special KOS Process 610.

In the first case, the Event microcode may perform a Microcode-to-Software Call to a high-level language procedure which handles the Event. An example of an Event handled in this fashion is a floating point overflow: when FU 10120 microcode determines that a floating point overflow has occurred, it invokes microcode which may invoke a floating point overflow procedure provided by the high-level language whose S-Language was being executed when the overflow occurred. In alternate embodiments of CS 10110, the overflow procedure may also be in microcode.

In the second case, the microcode handling the fault or interrupt puts information in tables used by a KOS Process 610 which handles the fault or interrupt and then causes the KOS Process 610 to run at some later time by advancing an Event Counter awaited by the Process 610. Event Counters and the operations on them are explained in detail in a following description of Processes 610. Since the tables and Event Counters manipulated by microcode are always present in MEM 10112, these operations do not cause page faults, and can be performed in monitor mode. For example, when IOS 10116 transmits an IPM Event signal to JP 10114 after IOS 10116 has loaded data into MEM 10112, the Event resulting from the Event signal invokes microcode which examines a queue containing messages from IOS 10116. The messages in the queue contain Event Counter locations, and the microcode which examines the queue advances those Event counters, thereby causing Processes 610 which were waiting for the data returned by the I/O operation to recommence execution.

3. The Monitor Mode, Differential Masking and Hardware Interrupt Handling

FU 10120 micromachine's monitor mode and differential masking facilities allow a method of hardware interrupt handling which overcomes two problems associated with conventional hardware interrupt handling: an interrupt can be handled in a predictable amount of time regardless of the amount of time required to execute an SIN, and if the microcode which handles the interrupt executes in monitor mode, the interrupt may be handled at any time without unpredictable consequences. There are two sources of hardware interrupts in CS 10110: an Inter-Processor Message (IPM) and an Interval Timer 25410 Runout. An IPM occurs when IOS 10116 completes an I/O task for JP 10114 and signals completion of the task via IOJP Bus 10132. An Interval Timer Runout occurs when a preset time at which CS 10110 must take some action is reached. For example, a given Process 610 may have a limit placed on the amount of time it may execute on JP 10114. As is explained in a following description of process synchronization, the virtual processor management system sets Interval Timer 25412 to run out when Process 610 has used all of the time available to it.

Petitioner Apple Inc. - Exhibit 1004, p. 5298

Both IPMs and Interval Timer Runouts begin as Event signals. The immediate effect of the Event signal is to set a bit in EP Field 27309 of MCW1. In principle, the set bit can cause invocation of the event microcode for the Event on the next M0 cycle in which the FU 10120 micromachine is in virtual mode. Since microroutines running in monitor mode are guaranteed to return the micromachine to virtual mode within a reasonable length of time, and the Event invocation will occur when this happens, the Event is guaranteed to be serviced in a reasonable period of time. The microroutines invoked by the Events themselves execute in monitor mode, thereby guaranteeing that no page faults will occur while they are executing and that Process 610 which is executing on JP 10114 when the hardware interrupt occurs need not be removed from JP 10114.

While hardware interrupts are serviced in principle as described above, considerations of efficiency require that as many hardware interrupts as possible be serviced when the size of the FU micromachine's stack is at a minimum, i.e., at the beginning of an SIN's execution. This requirement is achieved by means of Egg TMR 25412 and ET Flag 27311 in MCW1 20290. As described above, when an IPM interrupt or an Interval Timer 25410 Runout interrupt occurs, Field 27317 or 27313 respectively is set in MCW1 20290. At the same time, Egg TMR 25412 begins running. If the current SIN's execution ends before Egg TMR 25412 runs out, the set Field in MCW1 20290 causes the Interval Timer Runout or Inter-processor Message Event invocations to occur on the first microinstruction for the next SIN. If, on the other hand, the current SIN's execution does not end before Egg TMR 25412 runs out, the Egg Timer Runout causes an Event signal. The immediate result of this signal is the setting of ET bit 27311 in MCW1 20290, and the setting of ET bit 27311 in turn causes the Interval Timer Runout Event invocation and/or IPM Event invocation to take place on the next M0 cycle to occur while the micromachine is in virtual mode. The above mechanism thus guarantees that most hardware interrupts will be handled at the beginning of an SIN, but that hardware interrupts will always be handled within a certain amount of time regardless of the length of time required to execute an SIN.

g. FU Micromachine and CS 10110 Subsystems

The subsystems of CS 10110, such as the object subsystem, the process subsystem, the S-Interpreter subsystem, and the Name Interpreter subsystem, are implemented all or in part in the micromachine. The description of the micromachine therefore closes with an overview of the relationship between these subsystems and the micromachine. Detailed descriptions of the operation of the subsystems have been presented previously.

The subsystems fall into three main groups: KOS subsystems, the Name Interpreter subsystem, and the S-Interpreter subsystem. The relationship between the three is to some extent hierarchical: the KOS subsystems provide the environment required by the Name Interpreter subsystem, and the Name Interpreter subsystem provides the environment required by the S-Interpreter subsystem. For example, the S-Interpreter subsystem interprets SINs consisting of SOPs and Names; the Name Interpreter subsystem translates Names into logical descriptors, using values called ABPs to calculate the locations contained in the logical descriptors. The KOS subsystems calculate the values of the ABPs, translate Logical Descriptors 27116 into physical MEM 10112 addresses, and check whether a Process 610 has access to an object which it is referencing.

In a present embodiment of CS 10110, the Name Interpreter subsystem and the S-Interpreter subsystem are implemented completely in the micromachine; in other embodiments, they could be implemented in high-level languages or in hardware. The KOS subsystems are implemented in both the micromachine and in high-level language routines. In alternate embodiments of CS 10110, KOS subsystems may be embodied entirely in microcode, or in high-level language routines. Some high-level language routines may execute in any Process 610, while others are executed only by special KOS Processes 610. The KOS subsystems also differ from the others in the manner in which the user has access: with the S-Interpreter subsystem and the Name Interpreter subsystem, the subsystems come into play only when SINs are executed; the subsystems are not directly visible to users of the system. Portions of the KOS subsystems, on the other hand, may be explicitly invoked in high-level language programs. For example, an invocation in a high-level language program may cause KOS to bind a Process 610 to a Virtual Processor 612.

The following will first list the functions performed by the subsystems, and then relate the subsystems to the monitor and virtual micromachine modes and specific micromachine devices. KOS subsystems perform the following functions:

- Virtual memory management;
- Virtual processor management;
- Inter-processor communication;
- Access Control;
- Object management; and,
- Process management.

The Name Interpreter performs the following functions:

- Fetching and parsing SOPs, and
- Interpreting Names.

The S-Interpreter, finally, dispatches SOPs, i.e., locates the FU 10120 and EU 10122 microcode which

Petitioner Apple Inc. - Exhibit 1004, p. 5299

executes the operation corresponding to a given SOP for a given S-Language.

Of these subsystems, the S-Interpreter, the Name Interpreter, and the microcode components of the KOS process and object manager subsystems execute on the virtual micromachine; the microcode components of the remaining KOS subsystems execute on the monitor micromachine. As will be seen in the discussions of these subsystems, subsystems which execute on the virtual micromachine may cause Page Faults, and may therefore reference data located anywhere in memory; subsystems which execute on the monitor micromachine may not cause Page Faults, and the data bases which these subsystems manipulate must therefore always be present at known locations in MEM 10112.

The relationship between subsystems and FU 10120 micromachine devices is the following: Microcode for all subsystems uses DESP 20210, Microcode Addressing 27013, and Register Addressing 27011, and may use EU Interface 27007. S-Interpreter microcode uses SOP Decoder 27003, and Name Interpreter Microcode uses Instruction Stream Reader 27001, Parsing Unit 27005, and Name Translation Unit 27015. KOS virtual memory management microcode uses Memory Reference Unit 27017, and Protection Microcode uses Protection Unit 27019.

Having described in detail the structure and operation of CS 10110's major subsystems, MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118, and the CS 10110 micromachine, CS 10110 operation will be described in further detail next below. First, operation of CS 10110's Namespace, S-Interpreter, and Pointer Systems will be described. Then, operation of CS 10110 will be described in further detail with respect to CS 10110's Kernel Operating System.

3. Namespace, S-Interpreters, and Pointers (Figs. 301—307, 274)

The preceding chapters have presented an overview of CS 10110, examined its hardware in detail, and explained how the FU 10120 hardware functions as a micromachine which controls the activities of other CS 10110 components. In the remaining portions of the specification, the means are presented by which certain key features of CS 10110 are implemented using the hardware, the micromachine, tables in memory, and high-level language programs. The present chapter presents three of these features: the Pointer Resolution System, Namespace, and the S-interpreters.

The Pointer Resolution System translates pointers, i.e., data items which contain location information, into UID-offset addresses. Namespace has three main functions:

— It locates SINs and fetches them from CS 10110's memory into FU 10120.
— It parses SINs into SOPs and Names.
— It translates Names into Logical Descriptors 27116 or values.

The S-interpreters decode S-operations received from namespace into locations in microcode contained in FUSITT 11012 and EUSITT 20344 and then execute that microcode. If the S-operations require operands, the S-interpreters use Namespace to translate the operands into Logical Descriptors 27116 or values as required by the operations.

Since Namespace depends on the Pointer Resolution System and the S-interpreters depend on Namespace, the discussion of the systems begins with pointers and then deals with namespace and S-interpreters.

A. Pointers and Pointer Resolution (Figs. 301, 302)

A pointer is a data item which represents an address, i.e., in CS 10110, a UID-offset address. CS 10110 has two large classes of pointers: resolved pointers and unresolved pointers. Resolved pointers are pointers whose values may be immediately interpreted as UID-offset addresses; unresolved pointers are pointers whose values must be interpreted by high level language routines or microcode routines to yield UID-offset addresses. The act of interpreting an unresolved pointer is called resolving it. Since the manner in which an unresolved pointer is resolved may be determined by a high-level language routine written by a system user, unresolved pointers provide a means by which users of the system may define their own pointer types.

Both resolved and unresolved pointers have subclasses. The subclasses of resolved pointers are UID pointers and object relative pointers. UID pointers contain a UID and offset, and can thus represent any CS 10110 address; object-relative pointers contain only an offset; the address's UID is assumed to be the same as that of the object containing the object-relative pointer. An object-relative pointer can therefore only represent addresses in the object which contains the pointer.

The subclasses of unresolved pointers are ordinary unresolved pointers and associative pointers. The difference between the two kinds of unresolved pointers is the manner in which they are resolved. Ordinary unresolved pointers are always resolved by high-level language routines, while associative pointers are resolved the first time they are used in a Process 610 and a domain by high-level language routines, but are subsequently resolved by means of a table called the Associated Address Table (AAT). This table is accessible to microcode, and associative pointers may therefore be more quickly resolved than ordinary unresolved pointers.

The following discussion will first explain the formats used by all CS 10110 pointers, and will then explain how pointers are processed in FU 10120.

Petitioner Apple Inc. - Exhibit 1004, p. 5300

a. Pointer Formats (Fig. 301)

Figure 301 represents a CS 10110 pointer. The figure has two parts: a representation of General Pointer Format 30101, which gives an overview of the fields which appear in all CS 10110 pointers, and a detailed presentation of Flags and Format Field 30105, which contains the information by which the kinds of CS 10110 pointers are distinguished.

Turning first to General Pointer Format 30101, all CS 10110 pointers contain 128 bits and are divided into three main fields:

- Offset Field 30103 contains the offset portion of a UID-offset address in resolved pointers and in associative pointers; in other unresolved pointers, it may contain an offset from some point in an object or other information as defined by the user.
- Flags and Format Field 30105 contains flags and format codes which distinguish between kinds of pointers. These flags and format codes are explained in detail below.
- UID field 30115 contains a UID in UID pointers and in some associative pointers; in objectrelative pointers, and other associative pointers, its meaning is undefined, and in ordinary unresolved pointers, it may contain information as defined by the user.

Flags and Format Field 30105 contains four subfields:

- Fields 30107 and 30111 are reserved and must be set to 0.
- NR Field 30109 indicates whether a pointer is resolved or unresolved. In resolved pointers, the field is set to 0, and in unresolved pointers, it is set to 1.
- Format Code Field 30113 indicates the kind of resolved or unresolved pointers. Format codes for the present embodiment are explained below.

The values of Format Code Field 30113 may range from 0 to 31. If Format Code Field 30113 has the value 0, the pointer is a null pointer, i.e., a pointer which neither directly nor indirectly indicates an address. The meanings of the other format codes depend on the value of NR Field 30109:

NR Field Value	Format Code Value	Meaning
0	1	UID pointer
0	2	Object-relative pointer
0	all other codes	Illegal
1	1	UID associative pointer
1	2	Object-relative associative pointer
1	all other codes	Ordinary unresolved pointer

As indicated by the above table, the present embodiment has two kinds of associative pointer, UID associative pointers and object-relative associative pointers. Like a UID pointer, a UID associative pointer contains a UID and an offset, and like an object-relative pointer, an object-relative associative pointer contains an offset and takes the value of the UID from the object to which it belongs. However, as will be explained in detail later, the UID and offset which the associative pointers contain or represent are not used as addresses. Instead, the UID and offset are used as tags to locate entries in the AAT, which associates an associative pointer with a resolved pointer.

b. Pointers in FU 10120 (Fig. 302)

When a pointer is used as an address in FU 10120, the address information in the pointer must be translated into a Logical Descriptor 27116 consisting of an AON, an offset, and a length field of 0; when a Logical Descriptor 27116 in FU 10120 is used to form a pointer value in memory, the AON must be converted back to a UID. The first conversion is termed pointer-to-descriptor conversion, and the second descriptor-to-pointer conversion. Both conversions are accomplished by microcodes executing in FU 10120.

What is involved in the translation depends on the kind of pointer: if the pointer is a UID pointer, the UID must be translated into an AON; if the pointer is an object-relative pointer, the AON required to fetch the pointer is the pointer's AON, so no translation is necessary. If the pointer is an unresolved pointer, it must first be translated into a resolved pointer and then into a Logical Descriptor 27116. If the pointer is associative, the translation to a resolved pointer may be performed by means of the ATT.

In the present embodiment, when other FU 10120 microcode calls pointer-to-descriptor microcode, the calling microcode passes Logical Descriptor 27116 for the location of the pointer which is to be translated as an argument to the pointer-to-description translation microcode. The pointer-to-descriptor microcode returns a Logical Descriptor 27116 produced from the value of the pointer at the location specified by

Petitioner Apple Inc. - Exhibit 1004, p. 5301

Logical Descriptor 27116 which the pointer-to-descriptor microcode received as an argument.

The pointer-to-descriptor microcode first uses Logical Descriptor 27116 given it as an argument to fetch the value of the pointer's Offset Field 30103 from memory. It then saves Logical Descriptor 27116's offset in the output register belonging to OFFALU 20242 and places the value of the pointer's Offset Field 30103 in the offset field of Logical Descriptor 27116 which it received as an argument. The pointer-to-descriptor microcode then saves Logical Descriptor 27116 indicating the pointer's location by storing Logical Descriptor 27116's AON and offset (obtained from OFFALU 20242) in a register in the GRF 10354 frame being used by the invocation of the pointer-to-descriptor microcode. Next, the microcode adds 40 to the offset stored in OFFALU 20242, thereby obtaining the address of NR Field 30109, and uses the address to fetch and read NR Field 30109 and Format Code Field 30113. The course of further processing is determined by the values of these fields. If NR Field 30109 indicates a resolved pointer, there are four cases, as determined by the value of Format Code Field 30113:

- Format code field = 0: The pointer is a null pointer.
- Format code field = 1: The pointer is a UID pointer.
- Format code field = 2: The pointer is an intra-object pointer.
- Any other value of the format code field: The pointer is invalid.

In the first case, the microcode sets all fields of the argument to 0; in the second, it fetches the value of UID Field 30115 from memory and invokes LAR microcode (explained in the discussion of objects), which translates the UID to the AON associated with it. The AON is then loaded into the argument's AON field. In the third case, the AON of Logical Descriptor 27116 for the pointer's location and the pointer's AON are the same, so the argument already contains the translated pointer. In the fourth case, the microcode performs a call to a pointer fault-handling Procedure 602 which handles invalid pointer faults, passing saved Logical Descriptor 27116 for the pointer as an argument. Procedure 602 which handles the fault must return a resolved pointer to the microcode, which then converts it to a Logical Descriptor 27116 as described above.

c. Descriptor to Pointer Conversion

Descriptor to pointer conversion is the reverse of pointer to descriptor conversion with resolved pointers. The operation must be performed whenever a resolved pointer is moved from an FU 10120 register into MEM 10112. The operation takes two arguments: a Logical Descriptor 27116 which specifies the address to which the pointer is to be written, and a Logical Descriptor 27116 whose AON and offset fields specify the location contained in the pointer. There are two cases: intra-object pointers and UID pointers. Both kinds of pointers have values in Offset Field 30103, so the descriptor-to-pointer microcode first writes the second argument's offset to location specified by the first argument's Logical Descriptor 27116. The next step is to determine whether the pointer is an intra-object pointer or a UID pointer. To do so, the microcode compares the arguments' AONs. If they are the same, the pointer points to a location in the object which contains it, and is therefore an intra-object pointer. Since UID Field 30115 of an intra-object pointer is meaningless, the only step remaining for intra-object pointers is to set Flags and Format Field 30105 to the binary representation of 2, which sets all bits but bit 46 to 0, and thereby identifies the pointer as a resolved intra-object pointer.

With UID pointers, the descriptor-to-pointer microcode sets Flags and Format Field 30105 to 1, thereby identifying the pointer as a resolved UID pointer, and calls a KOS LAR microroutine (explained in detail in the discussion of objects) which converts the first argument's AON to a UID and places the result UID in the current frame. When the KOS AON to UID conversion microroutine returns, the descriptor-to-pointer microcode writes the UID to the converted pointer's UID Field 30115.

B. Namespace and the S-Interpreters (Figs. 303—307)

Namespace and the S-Interpreter both interpret information contained in Procedure Objects 608. Consequently, the discussion of these components of CS 10110 begins with an overview of those parts of Procedure Object 606 relevant to Namespace and the S-interpreters, and then explains Namespace and the S-interpreters in detail.

a. Procedure Object 606 Overview (Fig. 303)

Figure 303 represents those portions of Procedure Object 608. Fig. 303 expands information contained in Fig. 103; Fields which appear in both Figures have the number of Fig. 103. Portions of Procedure Object 608 which are not discussed here are dealt with later in the discussion of Calls and Returns. The most important part of a Procedure Object 608 for these systems is Procedure Environment Descriptor (PED) 30303. A Procedure 602's PED 30303 contains the information required by Namespace and the S-interpreter to locate and parse Procedure 602's code and interpret its Names. A number of Procedures 602 in a Procedure Object 608 may share a PED 30303. As will be seen in the discussion of Calls, the fact that a Procedure 602 shares a PED 30303 with the Procedure 602 that invokes it affects the manner in which the Call is executed.

The fields of PED 30303 which are important to the present discussion are three fields in Header 30304: K Field 30305, LN Field 30307, and SIP Field 30309, and three of the remaining fields: NTP Field 30311, SDPP Field 30313, and PBP Field 30315.

Petitioner Apple Inc. - Exhibit 1004, p. 5302

— K Field 30305 indicates whether the Names in the SINs of Procedures 602 which share PED 30303 have 8, 12, or 16 bits.
— LN Field 30307 contains the Name which has the largest index of any in Procedure 602's Name Table 10350.
— SIP Field 30309 is a UID pointer to the object which contains the S-interpreter for Procedure 602's S-Language.
— NTP Field 30311 is an object-relative pointer to the beginning of Procedure 602's Name Table 10350.
— SDPP Field 30313 is a pointer which is resolved to the location of static data used by Procedures 602 to which PED 30303 belongs when one of Procedures 602 is invoked by a given Process 610. The resolved pointer corresponding to SDPP 30313 is the SDP ABP.
— PBP Field 30315 contains the PBP ABP for invocations of Procedures 602 to which PED 30303 belongs. The PBP ABP is used to calculate locations inside Procedure Object 608.

Other areas of interest in Procedure Object 608 are Literals 30301 and Static Data Prototype (SDPR) 30317. Literals 30301 contains literal values, i.e., values in Procedure 602 which are known at compile time and will not change during program execution. SDPR 30317 may contain any of the following: pointers to external routines and to static data contained in other objects, information required to create static data for a Procedure 602, and in some cases, the static data itself. Pointers in SDPR 30317 may be either resolved or non-resolved.

In the present embodiment, Binder Area 30323 is also important. Binder Area 30323 contains information which allows unresolved pointers contained in Procedure Object 608 to be resolved. Unresolved pointers other than SDPP 30313 in Procedure Object 608 all contain locations in Binder Area 30323, and the specified location contains the information required to resolve the pointer.

Fig. 303 contains arrows showing the locations in Procedure Object 608 pointed to by NTP Field 30311, SDPP Field 30313, and PBP Field 30315. NTP Field 30311 points to the beginning of Name Tables 10350, and thus a Name's Name Table Entry can be located by adding the Name's value to NTP Field 30311. PBP Field 30315 points to the beginning of Literals 30301, and consequently, the locations of Literals and the locations of SINs may be expressed as offsets from the value of PBP Field 30315. SDPP Field 30313 points to the beginning of SDPR 30317. As will be explained in detail in the discussion of Calls, when a procedure 602 has static data, the SDP ABP is derived from SDPP Field 30313.

b. Namespace

The Namespace component of CS 10110 locates SINs belonging to a procedure and fetches them from memory to FU 10120, parses SINs into SOPs and Names, and performs Resolve and Evaluation operations on Names. The Resolve operation translates a Name into a Logical Descriptor 27116 for the data represented by the Name, while the Evaluation operation obtains the data itself. The Evaluation operation does so by performing a Resolve operation and then using the resulting Logical Descriptor 27116 to fetch the data. Since the Evaluation and Resolve operations are the most complicated, the discussion begins with them.

1. Name Resolution and Evaluation

Name Resolution and Evaluation translate Names into Logical Descriptors 27116 by means of information contained in the Names' NTEs, and the NTEs define locations in terms of Architectural Base Registers. Consequently, the following discussion will first describe Name Table Entries and Architectural Base Pointers and then the means by which Namespace translates the information contained in the Name Table Entries and Architectural Base Pointers into Logical Descriptors 27116.

2. The Name Table (Fig. 304)

As previously mentioned, Name Tables 10350 are contained in Procedure Objects 608. Name Tables 10350 contain the information required to translate Names into Logical Descriptors 27116 for the operands represented by the Names. Each Name has as its value the number of a Name Table Entry. A Name's Name Table Entry is located by multiplying the Name's value by the size of a short Name Table Entry and adding the product to the value in NTP Field 30311 of PED 30303 belonging to Procedure 602 which contains the SIN.

The Name Table Entry contains length and type information for the data item specified by the Name, and represents the data item's location as a displacement from a known location, termed the base. The base may be a location specified by an ABP, a location specified by another Name, or a location specified by a pointer. In the latter case, the pointer's location may be specified in terms of an ABP or as a Name.

Fig. 304 is a detailed representation of a Name Table Entry (NTE) 30401. There are two kinds of NTEs 30401: Short NTEs 30403 and Long NTEs 30405. Short NTEs 30403 contain 64 bits; Long NTEs 30405 contain 128 bits. Names that represent scaler data items whose displacements may be expressed in 16 bits have Short NTEs 30403; Names that represent scaler data items whose displacements require more than 16 bits and Names that represent array elements have Long NTEs 30405.

A Short NTE 30403 has four main fields, each 16 bits in length:

— Flags and Format Field 30407 contains flags and format information which specify how Namespace is to interpret NTE 30401.

Petitioner Apple Inc. - Exhibit 1004, p. 5303

— Base Field 30425 indicates the base to which the displacement is to be added to obtain the location of the data represented by the Name. Base Field 30425 may represent the location in four ways: by means of an ABP by means of a Name, by means of a pointer located by means of an ABP, and by means of a pointer located by means of a Name.
— Length Field 30435 represents the length of the data. The length may be a literal value or a Name. If it is a Name, the Name resolves to a location which contains the data item's length.
— Displacement Field 30437 contains the displacement of the beginning of the data from the base specified in Field 30425. The displacement is a signed integer value.

Long NTEs 30405 have four additional fields, each 16 bits long: Two of the fields, Index Name Field 30441 and IES Field 30445 are used only in NTEs 30401 for Names that represent arrays.

— Displacement Extension Field 30439 is used in all Long NTEs 30405. If the displacement value in Field 30437 has less than 16 bits, Displacement Extension Field 30439 contains sign bits, i.e., the bits in the field are set to 0 when the displacement is positive and 1 when the displacement is negative. When the displacement value has more than 16 bits, Displacement Extension Field 30439 contains the most significant bits of the displacement value as well as sign bits.
— Index Name Field 30441 contains a Name that represents a value used to index an element of an array.
— Field 30443 is reserved.

IES Field 30445 contains a Name or Literal that specifies the size of an element in an array. The value represented by this field is used together with the value represented by Index Name Field 30441 to locate an element of an array.

As may be seen from the above, the following fields may contain names: Base Field 30425, Length Field 30435, Index Name Field 30441, and IES Field 30445.

Two fields in NTE 30401 require further consideration: Flags and Format Field 30407 and Base Field 30425. Flags and Format Field 30407 has three subfields: Flags Field 30408, FM Field 30421, and Type Field 30423. Turning first to Flags Field 30408, the six flags in the field indicate how Namespace is to interpret NTE 30401. The flags have the following meanings when they are set:

— Long NTE Flag 30409: NTE 30401 is a Long NTE 30405.
— Length is a Name Flag 30411: Length Field 30435 contains a Name.
— Base is a Name Flag 30413: Base Field 30425 contains a Name instead of the number of an ABP.
— Base Indirect Flag 30415: Base Field 30425 represents a pointer, and the location represented by NTE 30401 is to be calculated by obtaining the pointer's value and adding the value contained in Displacement Field 30437 and Displacement Extension Field 30439 to the pointer's offset.
— Array Flag 30417: NTE 30401 represents an array.
— IES is a Name Flag 30419: IES Field 30445 contains a Name that represents the IES value.

Several of these flags may be set in a given NTE 30401. For example, an entry for an array element that was referenced via a pointer to the array which in turn was represented by a Name, and whose IES value was represented by a Name, would have Flags 30409, 30413, 30415, 30417, and 30419 set.

FM Field 30421 indicates how the data represented by the Name is to be formated when it is fetched from memory. The value of FM Field 30421 is placed in FIU Field 27107 of Logical Descriptor 27116 produced from NTE 30401. The two bits allow for four possibilities:

Setting	Meaning
00	right justify, zero fill
01	right justify, sign fill
10	left justify, zero fill
11	left justify, ASCII space fill

The four bits in Type Field 30423 are used by compilers for language-specific type information. The value of Type Field 30423 is placed in Type Field 27109 of Logical Descriptor 27116 produced from NTE 30401.

Base Field 30425 may have either Base is an ABP Format 30427 or Base is a Name Format 30432. The manner in which Base Field 30425 is interpreted depends on the setting of Base is a Name Flag 30413 and Base Indirect Flag 30415. There are four possibilities:

Petitioner Apple Inc. - Exhibit 1004, p. 5304

Field Settings

Base is a Name	Base Indirect	Meaning
0	0	ABP Format locates base directly.
0	1	ABP Format locates a pointer which is the base.
1	0	Base is Name Format locates base when Name is resolved.
1	1	Base is Name Format locates a pointer when Name is resolve and the pointer is the base.

As indicated by the above table, Base Field 30425 is interpreted as having Base is ABP Format 30427 when Base is a Name Flag 30411 is not set. In Base is ABP Format 30427, Base Field 30425 has two subfields: ABP Field 30429 and Pointer Locator Field 30431. The latter field has meaning only when Base Indirect Flag 30415 is set. ABP Field 30429 is a two-bit code which indicates the ABP. The settings and their meanings are the following:

Setting	APB
00	FP
01	Unused
10	SDP
11	PBP

The ABPs are discussed below. When Base Indirect Flag 30415 is set to 1 and Base is a Name Flag 30413 is set to 0, the remaining 14 bits of the Base Field in ABP Format are interpreted as Pointer Locator Field 30413. When so interpreted, Pointer Locator Field 30413 contains a signed integer, which, when multiplied by 128, gives the displacement of a pointer from the ABP specified in ABP Field 30429. The value of this pointer is then the base to which the displacement is added.

Base Field 30425 is interpreted as having Base is a Name Format 30432 when Base is a Name Flag 30413 is set to 1. In Base is a Name Format 30432, Base Field 30425 contains a Name. If Base Indirect Flag 30415 is not set, the Name is resolved to obtain the Base. If Base Indirect Flag 30415 is set, the name is evaluated to obtain a pointer value, and that pointer value is the Base.

3. Architectural Base Pointers (Figs. 305, 306)

If Base is a Name Flag 30413 belonging to a NTE 30401 is not set, Base Field 30425 specifies one of the three ABPs in CS 10110:

- PBP specifies a location in Procedure Object 608 to which displacements may be added to obtain the locations of Literals and SINs.
- SDP specifies a location in a Static Data Block for an invocation of a Procedure 602 to which displacements may be added to obtain the locations of static data and linkage pointers to Procedures 602 contained in other Procedure Objects 608 and static data.
- FP specifies a location in the MAS frame belonging to Procedure 602's current invocation to which displacements may be added to obtain the location of local data and linkage pointers to arguments.

Each time a Process 610 invokes a Procedure 602, Call microcode saves the current values of the ABPs on Secure Stack 10336, calculates the values of the ABPs for the new invocation, and places the resulting Logical Descriptors 27116 in FU 10120 registers, where they are accessible to Namespace microcode.

Call microcode calculates the ABPs as follows: PBP is obtained directly from PBP Field 30315 in PED 30303 belonging to the Procedure 602 being executed. All that is required to make it into a Logical Descriptor 27116 is the addition of the AON for Procedure Object 608's UID.

SDP is obtained by performing a pointer-to-descriptor translation on SDPP Field 30313. FP, finally, is provided by the portion of Call microcode which creates the new MAS 502 frame for the invocation. As is described in detail in the discussion of Call, the Call microcode copies linkage pointers to the invocation's actual arguments onto MAS 502, sets FP to point to the location following the last actual argument, and then allocates storage for the invocation's local data. Positive displacements from FP thus specify locations

Petitioner Apple Inc. - Exhibit 1004, p. 5305

in the local data, while negative offsets specify linkage pointers.

a.a. Resolving and Evaluating Names (Fig. 305)

The primary operations performed by Namespace are resolving names and evaluating them. A Name has been resolved when Namespace has used the ABPs and information contained in the Name's NTE 30401 to produce a Logical Descriptor 27116 for the Name; a name has been evaluated when Namespace has resolved the Name, presented the resulting Logical Descriptor 27116 for the Name to memory, and obtained the value of the data represented by the Name from memory.

The resolve operation has three parts, which may be performed in any order:

— Obtaining the Base from Base Field 30425 of the Name's NTE 30401.
— Obtaining the displacement.
— Obtaining the length from Length Field 30435.

Obtaining the length is the simplest of the operations: if Length in a Name Flag 30411 is set, the length is the value obtained by evaluating the Name contained in Length Field 30435; otherwise, Length Field 30435 contains a literal value and the length is that literal's value.

There are four ways in which the Base may be calculated. Which is used depends on the settings of Base is a Name Flag 30413 and Base Indirect Flag 30415:

— Both Flags 0: the ABP specified in ABP Field 30429 is the Base.
— Base is a Name Flag 30413 0 and Base Indirect Flag 30415 1: The Base is the location contained in the pointer specified by ABP Field 30429 and pointer Locator Field 30431.
— Base is a Name Flag 30413 1 and Base Indirect Flag 30415 0: The Base is the location obtained by resolving the Name in Base Field 30425.
— Both Flags 1: The Base is the location obtained by evaluating the Name in Base Field 30425.

The manner in which Namespace calculates the displacement depends on whether NTE 30401 represents a scaler data item or an array data item. In the first case, Namespace adds the value contained in Displacement Field 30437 and Displacement Extension Field 30439 to the location obtained for the Base; in the second case, Namespace evaluates Index Name Field 30441 and IES Field 30445, multiplies the resulting values together, and adds the product to the value in Displacement Field 30437 in order to obtain the displacement.

If any field of a NTE 30401 contains a Name, Namespace obtains the value or location represented by the Name by performing a Resolve or Evaluation operation on it as required. As mentioned in the discussion of NTEs 30401, flags in Flags Field 30408 indicate which fields of an NTE 30401 contain Names. Since the NTE 30401 for a Name used in another NTE 30401 may itself contain Names, Namespace performs the Resolve and Evaluation operations recursively.

b.b. Implementation of Name Evaluation and Name Resolve in CS 10110

In the present embodiment, the Name Evaluation and Resolve operations are carried out by FU 10120 microcode Eval and Resolve commands. Both commands require two pieces of information: a register in the current frame of SR portion 10362 of GRF 10354 for receiving Logical Descriptor 27116 produced by the operation, and the source of the Name which is to be resolved or evaluated. Both Resolve and Eval may choose between three sources: Parser 20264, Name Trap 20254, and the low-order 16 bits of the output register for OFFALU 20242. Resolve may specify current frame registers 0, 1, or 2 for Logical Descriptor 27116, and Eval may specify current frame registers 0 or 1. At the end of the Resolve operation, Logical Descriptor 27116 for the data represented by the Name is in the specified SR 10362 register and at the end of the Evaluation operation, Logical Descriptor 27116 is in the specified SR 10362 register and the data's value has been transferred via MOD Bus 10114 to EU 10122's OPB 20322.

The execution of both Resolve and Eval commands always begin with the presentation of the Name to Name Cache 10226. The Name presented to Name Cache 10226 is latched into Name Trap 20254, where it is available for subsequent use by Name Resolve microcode.

If there is an entry for the Name in Name Cache 10226, a name cache hit occurs. For Names with NTEs 30401 fulfilling three conditions, the Name Cache 10226 entry for the Name is a Logical Descriptor 27116 for the data item represented by the Name. The conditions are the following:

— NTE 30401 contains no Names.
— Length Field of NTE 30401 specifies a length of less than 256 bits.
— If Base is Indirect Flag 30415 is set, Pointer Displacement Field 30431 must have a negative value, indicating that the base is a linkage pointer.

Logical Descriptor 27116 can be encached in this case because neither the location nor the length of the data represented by the Name can change during the life of an invocation of Procedure 602 to which the Name belongs. If the Name Cache 10226 entry for the Name is a Logical Descriptor 27116, the hit causes Name Cache 10226 to place Logical Descriptor 27116 in the specified SR 10362 register. In all other cases, the Name Cache 10226 entry for the Name does not contain a Logical Descriptor 27116, and a hit causes Name Cache 10226 to emit a JAM signal. The JAM signal invokes microcode which uses information stored in Name Cache 10226 to construct Logical Descriptor 27116 for the data item represented by the Name. JAMS are explained in detail below.

If there is no entry for the Name in Name Cache 10226, a Name Cache Miss occurs, and Name Cache 10226 emits a cache miss JAM signal. The Name Resolve microroutine invoked by the cache miss JAM

Petitioner Apple Inc. - Exhibit 1004, p. 5306

signal constructs an entry in Name Cache 10226 from the Name's NTE 30401, using FU 10120's DESP 20210 to perform the necessary calculations. When it is finished, the cache miss microcode leaves a Logical Descriptor 27116 for the Name in the specified SR 10362 register and returns.

The Resolve operation is over when Logical Descriptor 27116 has been placed in the specified GRF 10354 register; the Evaluation operation continues by presenting Logical Descriptor 27116 to Memory Reference Unit 27017, which reads the data represented by Logical Descriptor 27116 from memory and places it on OPB 20322. The memory reference may result in Protection Cache 10234 misses and ATU 10228 misses, as well as protection faults and page faults, but these are handled by means of event signals and are therefore invisible to the Evaluation operation.

Name Cache 10226 produces 15 different JAM signals. The signal produced by a JAM depends on the following: whether the operation is a Resolve or an Eval, which register Logical Descriptor 27116 is to be placed in, whether a miss occurred, and in the case of a hit, which register in the Name Cache 10226 entry for the Name was loaded last. From the point of view of the behavior of the microcode invoked by the JAM, the last two factors are the most important. Their relation to the microcode is explained in detail below.

In the present embodiment, all entries in Name Cache 10226 are invalidated when a Procedure 602 calls another Procedure 602. The invalidation is required because Calls always change the value of FP and may also change the values of SDP and PBP, thereby changing the meaning of NTEs 30401 using displacements from ABPs. Entries for Names in invoked Procedure 602 are created and loaded into Name Cache 10226 when the Names are evaluated or resolved and a cache miss occurs.

The following discussion will first present Name Cache 10226 as it appears to the microprogrammer and then explain in detail how Name Cache 10226 is used to evaluate and resolve Names, how it is loaded, and how it is flushed.

c.c. Name Cache 10226 Entries (Fig. 306)

The structure and the physical behavior of Name Cache 10226 was presented in the discussion of FU 10120 hardware; here, the logical structure of Name Cache 10226 entries as they appear to the microprogrammer is presented. To the microprogrammer, Name Cache 10226 appears as a device which, when presented a Name on NAME Bus 20224, always provides the microprogrammer with a Name Cache 10226 entry for the Name consisting of four registers. The microprogrammer may read from or write to any one of the four registers. When the microprogrammer writes to the four registers, the action taken by Name Cache 10226 when a hit occurs on the Name associated with the four registers depends on which of the registers has most recently been loaded. The means by which Name Cache 10226 associates a Name with the four registers, and the means by which Name Cache 10226 provides registers when it is full are invisible to the microprogrammer.

Fig. 306 illustrates Name Cache Entry 30601 for a Name. The four Registers 30602 in Name Cache Entry 30601 are numbered 0 through 3, and each Register 30602 has an AON, offset, and length field like those in GRF 10354 registers, except that some flag bits in GRF 10354 register AON fields are not included in Register 30602 fields, and the length field in Register 30602 is 8 bits long. As is the case with GRF 10354 registers, the microprogrammer can read or write individual fields of Register 30602 or entire Register 30602. Name Cache Entry 30601 is connected via DB 27021 to DESP 20210, and consequently, the contents of a GRF 10354 register may be obtained from or transferred to a Register 30602 or viceversa. When the contents of a Register 30602 have been transfered to a GRF 10354 register, the contents may be processed using OFFALU 20242 and other arithmetic-logical devices in DESP 20210.

d.d. Name Cache 10226 Hits

When a Name is presented to Name Cache 10226 and Name Cache 10226 has a Name Cache Entry 30601 containing information about the Name, a name cache hit occurs. On a hit, Name Cache 10226 hardware always loads the contents of Register 30602 0 of the Name's Name Cache Entry 30601 into the GRF 10354 register specified in the Resolve or Eval microcommand. In addition, a hit may result in the invocation of microcode via a JAM:

- The JAM may invoke special microcode for resolving Names of array elements whose NTEs 30401 allow certain hardware accelerations of index calculations.
- The JAM may invoke general name resolution microcode which produces a Logical Descriptor 27116 from the contents of Name Cache Entry 30601.

Whether the hit produces a JAM, and the kind of JAM it produces, are determined by the last Register 30602 to be loaded when Name Cache Entry 30601 was created by Name Cache Miss microcode. If Register 30602 0 was the last to be loaded, no JAM occurs; if Register 30602 1 was loaded last, the JAM for special array Name resolution occurs; if Register 30602 2 or 3 was loaded last, the JAM for general Name resolution occurs.

As may be inferred from the above, Name Cache 10226 hardware defines the manner in which Name Cache Entries 30601 are loaded for the first two cases. In the first case, Name Cache Register 30602 O must contain Logical Descriptor 27116 for the Name's data. As already mentioned, the Name's NTE 30401 must therefore describe data whose location and length does not change during an invocation and whose length is less than 256 bits. Name Cache 10226 hardware also determines the form of Name Cache Entries 30601 for encachable arrays. An encachable array NTE 30401 is an array NTE 30401 which fills the following

Petitioner Apple Inc. - Exhibit 1004, p. 5307

conditions:

- The only Name contained in array NTE 30401 is in Index Name Field 30441.
- NTE 30401 for the index Name fills the conditions for scaler NTEs 30401 for which Logical Descriptors 27116 may be encached.
- The value in IES Field 30445 is no greater than 128 and a power of 2.
- Array NTE 30401 otherwise fills the conditions for scaler NTEs 30401 for which Logical Descriptors 27116 may be encached.

In the present embodiment, the encachable array entry uses registers 0, 1, and 2 of Name Cache Entry 30601 for the name:

Register	Contents		
	AON	OFFSET	LENGTH
0	Logical Descriptor 27116 for the index Name		
1	0	IES power of 2	unused
2	Logical Descriptor 27116 for the array		

When a hit for this type of entry occurs, the resulting JAM signal does two things: it invokes encachable array resolve microcode and it causes the index Name's Logical Descriptor 27116 to be presented to Memory Reference Unit 27017 for a read operation which returns the value of the data represented by the index Name to an accumulator in OFFALU 20242. The encachable array resolve microroutine then uses the Name that caused the JAM, latched into Name Trap 20254, to locate Register 30602 2 of Name Cache Entry 30601 for the Name, writes the contents of Register 30602 2 into the GRF register specified by the Resolve or Eval microcommand, obtains the product of the IES value and the index value by shifting the index value left the number of times specified by the IES exponent in Register 30602 1, adds the result to the offset field of the GRF 10354 register containing the array's Logical Descriptor 27116, thus obtaining Logical Descriptor 27116 for the desired array element, and returns.

For the other cases, the manner in which Name Cache Entries 30601 are loaded and processed to obtain Logical Descriptors 27116 is determined by the microprogrammer. The JAM signal which results if a Name Cache Entry 30601 is neither a Logical Descriptor 27116 nor an encachable array entry merely invokes a microroutine. The microroutine uses the Name latched into Name Trap 20254 to locate the Name's Name Cache Entry 30601 and then reads tag values in Name Cache Entry 30601 to determine how the information in Name Cache Entry 30601 is to be translated into a Logical Descriptor 27116. The contents of Name Cache Entries 30601 for the other cases have two general forms: one for NTEs 30401 with Base is Indirect Flag 30415 set, and one for NTEs in which it is not set. The first general form looks like this:

Register	Contents		
	AON	OFFSET	LENGTH
0	ABP AON	tag/length	unused
1	0	index name/IES	unused
2	0	unused	unused
3	0	data displacement from loc. specified by pointer	unused

Register 30602 0 contains the AON of the ABP. Register 30602 0's offset field contains two items: the tag, which contains Flags Field 30408 of NTE 30401 along with other information, and which determines how Name Resolve microcode interprets the contents of Name Cache Entry 30601, and a value or Name for the length of the data item. Register 30602 1 is used only if the Name represents a data item in an array. It then contains the Name from Index Field 30441 and the Name or value from IES Field 30445. The offset field of Register 30602 3 contains the sum of the offset indicated by NTE 30401's ABP and of the displacement indicated by NTE 30401.

The second format, used for NTEs 30401 whose bases are obtained from pointers or by resolving a Name, looks like this:

Petitioner Apple Inc. - Exhibit 1004, p. 5308

Registers	Contents		
	AON	OFFSET	LENGTH
0	0	tag/length	unused
1	0	index name/IES	unused
2	0	FM and type bits/ base field	unused
3	0	data displacement from loc. specified by pointer or name	unused

In this form, the location of the Base must be obtained either by evaluating a pointer or resolving a Name. Hence, there is no field specifying the Base's AON. Otherwise, Registers 30602 0 and 1 have the same contents as in the previous format. In Register 30602 2, the offset field contains Name Table Entry 30401's FM Field 30421 and Type Field 30423 and Base Field 30425. The Offset Field of Register 30602 2 contains the value of Name Table Entry 30401 Displacement Fields 30437 and 30439.

As in Name Table Entries 30401, the index must be represented by a Name, and length, IES, and Base may be represented by Names. If a field of Name Cache Entry 30601 contains a Name, a flag in the tag indicates that fact, and Name Resolve microcode performs an Eval or Resolve operation on it as required to obtain the value or location represented by the name.

The microcode which resolves Name Cache Entries 30601 of the types just described uses the general algorithms described in the discussion of Name Table Entries 30401, and is therefore not discussed further here.

e.e. Name Cache 10226 Misses

When a Name is presented to Name Cache 10226 and there is no Name Cache Entry 30601 for the Name, a name cache miss occurs. On a miss Name Cache 10226 hardware emits a JAM signal which invokes name cache miss microcode. The microcode obtains the Name which caused the miss from Name Trap 20254 and locates the Name's NTE 30401 by adding the Name to the value of NTP 30311 from PED 30303 for Procedure 602 being executed. As will be explained in detail later, when a Procedure 602 is called, the Call microcode places the AON and offset specifying the NTP's location in a register in GR's 10360. Using the information contained in the Name's NTE 30401, the Cache Miss microcode resolves the Name and constructs a Name Cache Entry 30601 for it. As described above, the microcode determines the method by which it resolves the Name and the form of the Name's Name Cache Entry 30601 by reading Flags Field 30408 in the Name's NTE 30401. Since the descriptions of the Resolve operation, the micromachine, Name Cache 10226, and the formats of Name Cache Entries 30601 are sufficient to allow those skilled in the art to understand the operations performed by Cache Miss microcode, no further description of the microcode is provided.

f.f. Flushing Name Cache 10226

As described in the discussion of Name Cache 10226 hardware, hardware means, namely VALS 24068, exist which allow Name Cache Entries 30601 to be invalidated. Name Cache Entries 30601 may be invalidated singly, or all entries in Name Cache 10226 may be invalidated by means of a single microcommand. The latter operation is termed name cache flushing. In the present embodiment, Name Cache 10226 must be flushed when Process 610 whose Virtual Processor 612 is bound to JP 10114 executes a Call or a Return and whenever Virtual Processor 612 NO is unbound from JP 10114. Flushing is required on Call and Return because Calls and Returns change the values of the ABPs and other pointers needed to resolve Names. At a minimum, a Call produces a new MAS Frame 10412, and a Return returns to a previous Frame 10412, thereby changing the value of FP. If the called Procedure 602 has a different PED 30303 from that of the calling Procedure 602, the Call or Return may also change PBP, SDP, and NTP. Flushing is required when a Virtual Processor 612 is unbound from JP 10114 because Virtual Processor 612 which is next bound to JP 10114 is bound to a different Process 610, and therefore cannot use any information belonging to Process 610 bound to the Previous Virtual Processor 612.

g.g. Fetching the I-Stream

As explained in the discussion of FU 10120 hardware, SINs are fetched from memory by Prefetcher 20260. PREF 20260 contains a Logical Descriptor 27116 for a location in Code 10344 belonging to Procedure 602 which is currently being executed. On any MO cycle, PREF 20260 can place Logical Descriptor 27116 on DB 27021, cause Memory Reference Unit 27017 to fetch 32 bits at the location specified by Logical

Petitioner Apple Inc. - Exhibit 1004, p. 5309

Descriptor 27116, and write them into INSTB 20262. When INSTB 20262 is full, PREF 20260 stops fetching SINs until Namespace parsing operations, described below, have processed part of the contents of INSTB 20262, thereby creating space for more SINs.

The fetching operation is automatic, and requires intervention from Namespace only when a SIN causes a branch, i.e., causes the next SIN to be executed to be some other SIN than the one immediately following the current SIN. On a branch, Namespace must load PREF 20260 with the location of the next SIN to be executed and cause PREF 20260 to begin fetching SINs at that location. The operation which does this is specified by the load-prefetch-for-branch microcommand. The microcommand specifies a source for a Logical Descriptor 27116 and transfers that Logical Descriptor 27116 via DB 27021 to PREF 20260. After PREF 20260 has thus been loaded, it begins fetching SINs at the specified location. Since any SINs still in INSTB 20262 have been rendered meaningless by the branch operation, the first SINs loaded into INSTB 20262 are simply written over INSTB 20262's prior contents. Fig. 274 contains an example of the use of the load-prefetch-for-branch microcommand.

h.h. Parsing the I-Stream

The I-stream as fetched from MEM 10112 and stored in INSTB 20262 is a sequence of SOPs and Names. As already mentioned, the I-stream has a fixed format: in the present embodiment, SOPs are always 8 bits long, and Names may be 8, 12, or 16 bits long. The length of Names used in a given procedure is fixed, and is indicated by the value in K Field 30305 in the Procedure 602's PED 30303. The Namespace parsing operations obtain the SOPs and Names from the I-stream and place them on NAME Bus 20224. The SOPs are transferred via this bus to the devices in SOP Decoder 27003, while the Names are transferred to Name Trap 20254 and Name Cache 10226 for Resolve and Evaluation operations as described above. As the parsing operations obtain SOPs and Names, they also update the three program counters CPC 20270, EPC 20274, and IPC 20272. The values in these three counters are offsets from PBP which point to locations in Code 10344 belonging to Procedure 602 being executed. CPC 20270 points to the I-stream syllable currently being parsed, so it is updated on every parsing operation. EPC 20274 points to the beginning of the last SIN executed by JP 10114, and IPC 20272 points to the beginning of the current SIN, so these program counters are changed only at the beginning of the execution of an SIN, i.e., when a SOP is parsed.

As described in the discussion of FU 10120 hardware, in the current implementation, parsing consists physically of reading 8 or 16 bits of data from a location in INSTB 20262 identified by a pointer for INSTB 20262 which is accessible only to the hardware. As data is read, the hardware increments the pointer by the number of bits read, wrapping around and returning to the beginning of INSTB 20262 if it reaches the end. At the same time that the hardware increments the pointer, it increments CPC 20270 by the same number of bits. As previously mentioned, CPC 20270 contains the offset from PBP of the SOP or Name being currently parsed, thus coordinating the reading of INSTB 20262 with the reading of Procedure 602's Code 10344.

The number of bits read depends on whether Parser 20264 is reading an SOP or a Name, and in the latter case, by the syllable size specified for the Name. The syllable size is contained in CSSR 24112. On a Call to a Procedure 602 which has a different PED 30303 from that of the calling procedure, the Call microcode loads the value contained in K Field 30305 into CSSR 24112.

Namespace's parsing operations are performed by separate microcommands for parsing SOPs and Names. There is a single microcommand for parsing S-operations: parse-op-stage. The microcommand obtains the next eight bits from INSTB 20262, places the bits onto NAME Bus 20224, and latches them into LOPCODE Register 24212. It also updates EPC 20274 and IPC 20272 as required at the beginning of an SIN: EPC 20274 is set to IPC 20272's former value, and IPC 20272 is set to CPC 20270's value. At the end of the operation, CPC 20270 is incremented by 8. Since the parsing of an SOP always occurs as the first operation in the interpretation of an SIN, the parse-op-stage command is generally combined with a dispatch fetch command. As will be explained below, the latter command interprets the S-operation as an address in FDISP 24218, and FDISP 24218 in turn produces an address in FUSITT 11012. The latter address is the location of the beginning of the SIN microcode for the SIN.

There are two microcommands for parsing Names:
parse_k_load_epc and parse_k_dispatch_ebox. Both commands obtain a number of bits from INSTB 20262 and place them on NAME Bus 20214. With both microcommands, the syllable size, K, stored in CSSR 24112, determines the number of bits obtained from INSTB 20262. Both commands also increment CPC by the value stored in CSSR 24112. In addition, parse_k_load_epc sets EPC to IPC's value, while parse_k_dispatch_ebox also dispatches EU 10122, i.e., interprets the SOP saved in LOPCODE 24210 as an address in EDISP 24222, which in turn contains an address in EU EUSITT 20344. The EU EUSITT 20344 address is passed via EUDIS Bus 20206 to COMQ 20342 in EU 10122.

c. The S-Interpreters (Fig. 307)

CS 10110 does not assign fixed meanings to SOPs. While all SOPs are 8 bits long, a given 8 bit SOP may have one meaning in one S-Language and a completely different meaning in another S-Language. The semantics of an S-Language's S-operations are determined completely by the S-interpreter for the S-Language. Thus, in order to correctly interpret an S-operation, CS 10110 must know what S-interpreter it is to use. The S-interpreter is identified by a UID pointer with offset 0 in SIP Field 30309 of PED 30303 for

Petitioner Apple Inc. - Exhibit 1004, p. 5310

Procedure 602 that CS 10110 is currently executing. In the present embodiment, the UID is the UID of a microcode object which contains FU 10120 microcode. When loaded into FUSITT 11012, the microcode interprets SOPs as defined by the S-Language to which the SOP belongs. In other embodiments, the UID may be the UID of a Procedure Object 608 containing Procedures 602 which interpret the S-Language's SOPs, and in still others, the S-interpreter may be contained in a PROM and the S-interpreter UID may not specify an object, but may serve solely to identify the S-interpreter.

When a Procedure 602 executes an SIN on JP 10114, CS 10110 must translate the value of SIP Pointer 30309 for Procedure 602 and the S-instruction's SOP into a location in the microcode or high-level language code which makes up the S-interpreter. The location obtained by the translation is the beginning of the microcode or high-level language code which implements the SIN. The translation of an SOP together with SIP Pointer 30309 into a location in the S-interpreter is termed dispatching. Dispatching in the present embodiment involves two primary components: a table in memory which translates the value of SIP Pointer 30309 into a small integer called the Dialect Number, and S-operation Decoder Portion 27003 of the FU 10120 micromachine. The following discussion will first present the table and explain how an SIP Pointer 30309 is translated into a Dialect Number, and then explain how the Dialect Number and the SOP together are translated into locations in FUSITT 11012 and EUSITT 20344.

1. Translating SIP into a Dialect Number (Fig. 307)

In the present embodiment, all S-interpreters in CS 10110 are loaded into FUSITT 11012 when CS 10110 begins operation and each S-interpreter is always placed in the same location. Which S-interpreter is used to interpret an S-Language is determined by a value stored in dialect register RDIAL 24212. Consequently, in the present embodiment, a Call to a Procedure 602 whose S-interpreter differs from that of the calling Procedure 602 must translate the UID pointer contained in SIP Field 30309 into a Dialect Number.

Fig. 307 represents the table and microcode which performs this translation in the present embodiment. S-interpreter Translation Table (STT) 30701 is a table which is indexed by small AONs. Each STT Entry (STTE) 30703 has two fields: an AON Field 30705 and a Dialect Number Field 30709. Dialect Number Field 30709 contains the Dialect Number for the S-interpreter object whose AON is in AON Field 30705.

When CS 10110 begins operation, each S-interpreter object is wired active and assigned an AON small enough to serve as an index in STT 30701. By convention, a given S-interpreter object is always assigned the same AON and the same Dialect Number. The AON is placed in AON Field 30705 of STTE 30703 indexed by the AON, and the Dialect Number is placed in Dialect Number Field 30709. Since the S-interpreter objects are wired active, these AONs will never be reassigned to other objects.

On a Call which requires a new S-interpreter, Call microcode obtains the new SIP from SIP Field 30309, calls KOS LAR microcode to translate its UID to its AON, uses the AON to locate the S-interpreter's STTE 30703, and places the value of Dialect Number Field 30709 into RDIAL 21242.

Other embodiments may allow S-interpreters to be loaded into FUSITT 11012 at times other than system initialization, and allow S-interpreters to occupy different locations in FUSITT 11012 at different times. In these embodiments, STT 30701 may be implemented in a manner similar to the implementations of AST 10914 or MHT 10716 in the present embodiment.

2. Dispatching

Dispatching is accomplished by Dispatch Files 27004. These files translate the values provided by RDIAL 24212 and the SOP of the S-instruction being executed into the location of microcode for the SIN specified by the S-operation in the S-interpreter specified by the value of RDIAL 24212. The present embodiment has three dispatch files: FDISP 24218, FALG 24220, and EDISP 24222. FDISP 24218 and FALG 24220 translate S-operations into locations of microcode which executes on FU 10120; EDISP 24222 translates S-operations into locations of microcode which executes on EU 10122. The difference between FDISP 24218 and FALG 24220 is one of speed: FDISP 24218 can translate an SOP in the same microinstruction which performs a parse_op_stage command to load the SOP into LOPCODE 24210. FALG 24220 must perform the translation on a cycle following the one in which the SOP is loaded into LOPCODE 24210. Typically, the location of the first portion of the microcode to execute an S-operation is contained in an FDISP 24218 register, the location of portions executed later is contained in an FALG 24220 register, and the location of microcode for the S-operation which executes on EU 10122 is contained in EDISP 24222.

In the present embodiment, the registers accomplish the translation from S-operation to microcode location as follows: As mentioned in the discussion of FU 10120 hardware, each Dispatch File contains 1024 registers. Each register may contain an address in an S-interpreter. As will be seen in detail later, the address may be an address in an S-interpreter's object, or it may be the address in FUSITT 11012 or EUSITT 20344 of a copy of microcode stored at an S-interpreter address. The registers in the Dispatch Files may be divided into sets of 128 or 256 registers. Each set of registers translates the SOPs for a single S-Language into locations in microcode. Which set of registers is used to interpret a given S-operation is decided by the value of RDIAL 24212; which register in the set is used is determined by the value of the S-operation. The value contained in the specified register is then the location of microcode which executes the S-instruction specified by the S-operation in the S-Language specified by RDIAL 24212.

Logically, the register addressed by the concatenated value in turn contains a 15 bit address which is

Petitioner Apple Inc. - Exhibit 1004, p. 5311

the location in the S-interpreter of the first microinstruction of microcode used to execute the S-instruction specified by the S-operation in the S-Language specified by the contents of RDIAL 24212. In the present embodiment, the microcode referred to by the address may have been loaded into FUSITT 11012 and EUSITT 20344 or it may be available only in memory. Addresses of microcode located in FUSITT 11012 and EUSITT 20344 are only eight bits long. Consequently, if a Dispatch File 27004 contains an address which requires more bits than that, the microcode specified by the address is in memory. As described in the discussion of FU 10112 hardware, addresses larger than 8 bits produce an Event Signal, and microcode invoked by the event signal fetches the microinstruction at the specified address in the S-interpreter from memory and loads it into location 0 of FUSITT 11012. The event microcode then returns, and the microinstruction at location 0 is executed. If the next microinstruction also has an address larger than 8 bits, the event signal occurs again and the process described above is repeated.

As previously mentioned, FDISP 24218 is faster than FALG 24220. The reason for the difference in speed is that FDISP registers contain only 6 bits for addressing the S-interpreter. The present embodiment assumes that all microcode addressed via FDISP 24218 is contained in FUSITT 11012. It concatenates 2 zero bits with the six bits in the FDISP 24218 register to produce an 8 bit address for FUSITT 11012. FDISP 24218 registers can thus contain the location of every fourth FUSITT 11012 register between FUSITT register 256 and FUSITT register 448. The microcode loaded into these locations in FUSITT 11012 is microcode for operations which are performed at the start of the SIN by many different SINs. For example, all SINs which perform operations on 2 operands and assign the result to a location specified by a third operand must parse and evaluate the first two operands and parse and resolve the third operand. Only after these operations are done are SINs-specific operations performed. In the present embodiment, the microcode which parses, resolves, and evaluates the operands is contained in a part of FUSITT 11012 which is addressable by FDISP 24218.

As previously mentioned, in the present embodiment, FUSITT 11012 and EUSITT 20344 may be loaded only when CS 10110 is initialized. The microcode loaded into FUSITT 11012 and EUSITT 20344 is produced by the microbinder from the microcode for the various SINs. To achieve efficient use of FUSITT 11012 and EUSITT 20344, microcode for operations shared by various S-interpreters appears only once in FUSITT 11012 and EUSITT 20344. While the SINs in different S-Languages which share the microcode have different registers in FDISP 24218, FALG 24220, or EDISP 24222 as the case may be, the registers for each of the S-instructions contain the same location in FUSITT 11012 or EUSITT 20344.

4. The Kernel Operating System

A. Introduction

Many of the unique properties of CS 10110 are produced by the manipulation of tables in MEM 10112 and Secondary Storage 10124 by programs executing on JP 10114. These programs and tables together make up the Kernel Operating System (KOS). Having described CS 10110's components and the means by which they cooperate to execute computer programs, this specification now presents a detailed account of KOS and of the properties of CS 10110 which it produces. The discussion begins with a general introduction to operating systems, then presents an overview of CS 10110's operating systems, an overview of the KOS, and detailed discussions of the implementation of objects, access control, and Processes 610.

a. Operating Systems (Fig. 401)

In CS 10110, as in other computer systems, the operating system has two functions:

- It controls the use of CS 10110 resources such as JP 10114, MEM 10112, and devices in IOS 10116 by programs being executed on CS 10110.
- It defines how CS 10110 resources appear to users of CS 10110.

The second function is a consequence of the first: By controlling the manner in which executing programs use system resources, the operating system in fact determines how the system appears to its users. Figure 401 is a schematic representation of the relationship between User 40101, Operating System 40102, and System Resources 40103. When User 40101 wishes to use a System Resource 40103, User 40101 requests the use of System Resource 40103 from Operating System 40102, and Operating System 40102 in turn commands CS 10110 to provide the requested Resources 40103. For example, when a user program wishes to use a peripheral device, it does not deal with the device directly, but instead calls the Operating System 40102 procedure 602 that controls the device. While Operating System 40102 must take into account the device's complicated physical properties, the user program that requested the device need know nothing about the physical properties, but must only know what information the Operating System 40102 Procedure 602 requires to perform the operation requested by the user program. For example, while the peripheral device may require that a precise pattern of data be presented to it, the Operating System 40102 procedure 602 may only require the data itself from the user program, and may format the data as required by the peripheral device. The Operating System 40102 Procedure 602 that controls the peripheral device thus transforms a complicated physical interface to the device into a much simpler logical interface.

1. Resources Controlled by Operating Systems (Fig. 402)

Operating Systems 40102 control two kinds of resources: physical resources and virtual resources. The physical resources in the present embodiment of CS 10110 are JP 10114, IOS 10116 and the peripheral

Petitioner Apple Inc. - Exhibit 1004, p. 5312

devices associated with IOS 10116, MEM 10112, and Secondary Storage 10124. Virtual resources are resources that the operating system itself defines for users of CS 10110. As was explained above, in controlling how CS 10110's resources are used, Operating System 40102 defines how CS 10110 appears to the users. Instead of the physical resources controlled by Operating System 40102, the user sees a far simpler set of virtual resources. The logical I/O device interface that Operating System 40102 gives the user of a physical I/O device is such a virtual resource. Often, an Operating System 40102 will define sets of virtual resources and multiplex the physical resources among these virtual resources. For instance, Operating System 40102 may define a set of Virtual Processors 612 that correspond to a smaller group of physical processors, and a set of virtual memories that correspond to a smaller group of physical resources. When a user executes a program, it runs on a Virtual Processor 612 and uses virtual memory. It seems to the user of the virtual processor and the virtual memory that he has sole access to a physical processor and physical memory, but in fact, Operating System 40102 is multiplexing the physical processors and memories among the Virtual Processors 612 and virtual memories.

Operating System 40102, too, uses virtual resources. For instance, the memory management portion of an Operating System 40102 may use I/O devices; when it does so, it uses the virtual I/O devices defined by the portion of the Operating System 40102 that manages the I/O devices. One part of Operating System 40102 may also redefine virtual resources defined by other parts of Operating System 40102. For instance, one part of Operating System 40102 may define a set of primitive virtual I/O devices and another part may use these primitive virtual I/O devices to define a set of high-level user-oriented I/O devices. Operating System 40102 thus turns the physical CS 10110 into a hierarchy of virtual resources. How a user of CS 10110 perceives CS 10110 depends entirely on the level at which he is dealing with the virtual resources.

The entity that uses the resources defined by Operating System 40102 is the process. A Process 610 may be defined as the activity resulting from the execution of a program with its data by a sequential processor. Whenever a user requests the execution of a program on CS 10110, Operating System 40102 creates a Process 610 which then executes the Procedures 602 making up the user's program. In physical terms, a process 610 is a set of data bases in memory that contain the current state of the program execution that the process represents. Operating System 40102 causes Process 610 to execute the program by giving Process 610 access to the virtual resources which it requires to execute the program, by giving the virtual resources access to those parts of Process 610's state which they require to perform their operations, and by giving these virtual resources access to the physical resources. The temporary relationship of one resource to another or of a Process 610 to a resource is called a binding. When a Process 610 has access to a given Virtual Processor 612 and Virtual processor 612 has access to process 610's state, process 610 is bound to Virtual Processor 612, and when Virtual Processor 612 has access to JP 10114 and Virtual Processor 612's state is loaded into JP 10114 registers, Virtual processor 612 is bound to JP 10114, and JP 10114 can execute SINs contained in Procedures 602 in the program being executed by Process 610 bound to Virtual Processor 612. Binding and unbinding may occur many times in the course of the execution of a program by a Process 610. For instance, if a Process 610 executes a reference to data and the data is not present in MEM 10112, then Operating System 40102 unbinds Process 610's Virtual Processor 612 from JP 10114 until the data is available in MEM 10112. If the data is not available for an extended period of time, or if the user for whom Process 610 is executing the program wishes to stop the execution of the program for a while, Operating System 40102 may unbind process 610 from its Virtual Processor 612. Virtual Processor 612 is then available for use by other Processes 610.

As mentioned above, the binding process involves giving a first resource access to a second resource, and using the first resource's state in the second resource. To permit binding and unbinding, Operating System 40102 maintains data bases that contain the current state of each resource and each Process 610. State may be defined as the information that the operating system must have to use the resource or execute the Process 610. The state of a line printer, for instance, may be variables that indicate whether the line printer is busy, free, off line, or out of order. A Process 610's state is more involved, since it must contain enough information to allow Operating System 40102 to bind Process 610 to a Virtual Processor 612, execute Process 610 for a while, unbind Process 610, and then rebind it and continue execution where it was halted. A process 610's state thus includes all of the data used by Process 610 up to the time that it was unbound from a Virtual Processor 612, along with information indicating whether Process 610 is ready to begin executing again.

Figure 402 shows the relationship between Processes 610, virtual, and physical resources in an operating system. The figure shows a multi-process Operating System 40102, that is, one that can multiplex CS 10110 resources among several Processes 610. The Processes 610 thus appear to be executing concurrently. The solid arrows in Figure 402 indicate bindings between virtual resources or between virtual and physical resources. Each Process 610 is created by Operating System 40102 to execute a user program. The program consists of Procedures 602, and Process 610 executes Procedures 602 in the order prescribed by the program. Processes 610 are created and managed by a component of Operating System 40102 called the Process Manager. Process Manager 40203 executes a Process 610 by binding it to a Virtual Processor 612. There may be more Processes 610 than there are Virtual Processors 612. In this case, Operating System 40102 multiplexes Virtual Processors 612 among Processes 610.

Virtual Processors 612 are created and made available by another component of Operating System 40102, Virtual Processor Manager 40205. Virtual Processor Manager 40205 also multiplexes JP 10114

Petitioner Apple Inc. - Exhibit 1004, p. 5313

among Virtual Processors 612. If a Virtual Processor 612 is ready to run, Virtual Processor Manager 40205 binds it to JP 10114. When Virtual Processor 612 can run no longer, or when another Virtual Processor 612 requires JP 10114, Virtual Processor Manager 40205 unbinds running Virtual Processor 612 from JP 10114 and binds another Virtual Processor 612 to it.

Virtual Processors 612 use virtual memory and I/O resources to perform memory access and input-output. Virtual Memory 40206 is created and managed by Virtual Memory Manager 40207, and Virtual I/O Devices 40208 are created and managed by Virtual I/O Manager 40209. Like Virtual Processor Manager 40205, Components 40207 and 40209 of Operating System 40102 multiplex physical resources among the virtual resources. As described above, one set of virtual resources may use another set. One way in which this can happen is indicated by the broken arrows in Figure 402. These arrows show a binding between Virtual Memory 40206 and Virtual I/O Device 40208. This binding occurs when Virtual Memory 40206 must handle a reference to data contained on a peripheral device such as a disk drive. To the user of Virtual Memory 40206, all data appears to be available in MEM 10110. In fact, however, the data is stored on peripheral devices such as disk drives, and copied into MEM 10112 when required. When a Process 610 references data that has not been copied into MEM 10112, Virtual Memory 40206 must use IOS 10116 to copy the data into MEM 10112. In order to do this, it uses a Virtual I/O Device 40208 provided by Virtual I/O Manager 40209.

b. The Operating System in CS 10110

For the sake of clarity, Operating System 40102 has been described as though it existed outside of CS 10110. In fact, however, Operating System 40102 itself uses the resources it controls. In the present embodiment, parts of Operating System 40102 are embodied in JP 10114 hardware devices, parts are embodied in microcode which executes on JP 10114, and parts are embodied in Procedures 602. These Procedures 602 are sometimes called by Processes 610 executing user programs, and sometimes by special Operating System Processes 610 which do nothing but execute operations for Operating System 40102.

The manner in which the components of Operating System 40102 interact may be illustrated by the way in which CS 10110 handles a page fault, i.e., a reference to data which is not available in MEM 10110. The first indication that there may be a page fault is an ATU Miss Event Signal. This Event Signal is generated by ATU 10228 in FU 10120 when there is no entry in ATU 10228 for a Logical Descriptor 27116 used in a read or write operation. The Event Signal invokes Operating System 40102 microcode, which examines a table in MEM 10112 in order to find whether the data described by Logical Descriptor 27116 has a copy in MEM 10112. If the table indicates that there is no copy, Operating System 40102 microcode communicates the fact of the page fault to an Operating System 40102 Virtual Memory Manager process 610 and removes Virtual Processor 612 bound to the Process 610 which was executing when the page fault occurred from JP 10114. Some time later, Virtual Memory Manager Process 610 is bound to JP 10114. Procedures 602 executed by Virtual Memory Manager Process 610 then initiate the I/O operations required to locate the desired data in Secondary Storage 10124 and copy it into MEM 10112. When the data is available in MEM 10112, Operating System 40102 allows Virtual Processor 612 bound to Process 610 which was executing when the page fault occurred to return to JP 10114. Virtual Processor 612 repeats the memory reference which caused the page fault, and since the data is now in MEM 10112, the reference succeeds and execution of Process 610 continues.

c. Extended Operating System and the Kernel Operating System (Fig. 403)

In CS 10110, Operating System 40102 is made up of two component operating systems, the Extended Operating System (EOS) and the Kernel Operating System (KOS). The KOS has direct access to the physical resources. It defines a set of primitive virtual resources and multiplexes the physical resources among the primitive virtual resources. The EOS has access to the primitive virtual resources defined by KOS, but not to the physical resources. The EOS defines a set of user-level virtual resources and multiplexes the primitive virtual resources defined by KOS among the user level virtual resources. For example, KOS provides EOS with Processes 610 and Virtual processors 612 and binds Virtual Processors 612 to JP 10114, but EOS decides when a Process 610 is to be created and when a process 610 is to be bound to a Virtual processor 612.

Figure 403 shows the relationship between a user Process 610, EOS, KOS, and the physical resources in CS 10110. Figure 403 shows three levels of interface between executing user Process 610 and JP 10114. The highest level of interface is Procedure Level 40302. At this level, Process 610 interacts with CS 10110 by calling Procedures 602 as specified by the program Process 610 is executing. The calls may be either calls to User Procedures 40306 or calls to EOS Procedures 40307. When Process 610 is executing a procedure 602, Process 610 produces a stream of SINs. The stream contains two kinds of SINs, S-language SINs 40310 and KOS SINs 40311. Both kinds of SINs interact with CS 10110 at the next level of interface, SIN-level Interface 40309. SINs 40310 and 40311 are interpreted by Microcode 40312 and 40313, and Microinstructions 40315 interact with CS 10110 at the lowest level of interface, JP 10114 Interface 40316. As already explained in the discussion of the FU 10120 micromachine, certain conditions in JP 10114 result in

Petitioner Apple Inc. - Exhibit 1004, p. 5314

Event Signals 40314 which invoke microroutines in S-interpreter Microcode 40312 or KOS Microcode 40313. Only Procedure-Level Interface 40302 and SIN-level Interface 40309 are visible to users. Procedure-level Interface 40309 appears as calls in user Procedures 602 or as statements in user Procedures 602 which compilers translate into calls to EOS procedures 602. SIN-level Interface 40309 appears as the Name Tables 10335 and SINs in Procedure Objects 608 generated by compilers.

As Figure 403 indicates, EOS exists only at Procedural Level 40302, while KOS exists at Procedural Level 40302, and SIN Level 40304, and within the microcode beneath SIN Level 40309. The only portion of the operating system that is directly available to user Processes 610 is EOS Procedures 40307. EOS Procedures 40307 may in turn call KOS procedures 40308. In many cases, an EOS Procedure 40307 will contain nothing more than the call to a KOS Procedure 40308.

User Procedures 40306, EOS Procedures 40307, and KOS Procedures 40308 all contain S-language SINs 40310. In addition, KOS Procedures 40308 only may contain special KOS SINs 40311. Special KOS SINs 40311 control functions that are not available to EOS Procedures 40307 or User Procedures 40306, and KOS SINs 40311 may therefore not appear in Procedures 40306 or 40307. S-language SINs 40310 are interpreted by S-interpreter Microcode 40312, while KOS SINs 40311 are interpreted by KOS Microcode 40313. KOS Microcode 40313 may also be called by S-interpreter Microcode 40313. Depending on the hardware conditions that cause Event Signals 40314, Signals 40314 may cause the execution of either S-interpreter Microcode 40312 or KOS Microcode 40313.

Figure 403 shows the system as it is executing a user Process 610. There are in addition special Processes 610 reserved for KOS and EOS use. These Processes 610 work like user Processes 610, but carry out operating system functions such as process management and virtual memory management. With one exception, EOS Processes 610 call EOS Procedures 40307 and KOS Procedures 40308, while KOS Processes 610 call only KOS Procedures 40308. The exception is the beginning of Process 610 execution: KOS performs the KOS-level functions required to begin executing a Process 610 and then calls EOS. EOS performs the required EOS level functions and then calls the first User Procedure 40306 in the program Process 610 is executing.

A description of how KOS handles page faults can serve to show how the parts of the system at the JP 10114—, SIN—, and procedure Levels work together. A page fault occurs when a Process 610 references a data item that has no copy in MEM 10112. The page fault begins as an Event Signal from ATU 10228. The Event Signal invokes a microroutine in KOS Microcode 40313. If the microroutine confirms that the referenced data item is not in MEM 10112, it records the fact of the page fault in some KOS tables in MEM 10112 and calls another KOS microroutine that unbinds Virtual Processor 612 bound to Process 610 that caused the page fault from JP 10114 and allows another Process 610's Virtual Processor 612 to run. Some time after the page fault, a special operating system Process 610, the Virtual Memory Manager Process 610, runs and executes KOS Procedures 40309. Virtual Memory Manager Process 610 initiates the I/O operation that reads the data from Secondary Storage 10124 into MEM 10112. When IOS 10116 has finished the operation, Process 610 that caused the page fault can run again and Virtual Memory Manager Process 610 performs an operation which causes Process 610's Virtual Processor 612 to again be bound to JP 10114. When Process 610 resumes execution, it again attempts to reference the data. The data is now in MEM 10112 and consequently, the page fault does not recur.

The division of Operating System 40102 into two hierarchically-related operating systems is characteristic for CS 10110. Several advantages are gained by such a division:

- Each of the two operating systems is simpler than a single operating system would be. EOS can concern itself mainly with resource allocation policy and high-level virtual resources, while KOS can concern itself with low-level virtual resources and hardware control.
- Because each operating system is simpler, it is easier to verify that each system's components are performing correctly, and the two systems are therefore more dependable than a single system.
- Dividing Operating System 40102 makes it easier to implement different embodiments of CS 10110. Only the interface provided by EOS is visible to the user, and consequently, the user interface to the system can be changed without altering KOS. In fact, a single CS 10110 may have a number of EOSs, and thereby present different interfaces to different users. Similarly, changes in the hardware affect the implementation of the KOS, but not the interface that KOS provides EOS. A given EOS can therefore run on more than one embodiment of CS 10110.
- A divided operating system is more secure than a single operating system. Physical access to JP 10114 is provided solely by KOS, and consequently, KOS can ensure that users manipulate only those resources to which they have access rights.

All CSs 10110 will have the virtual resources defined by KOS, while the resources defined by EOS will vary from one CS 10110 to another and even within a single CS 10110. Consequently, the remainder of the discussion will concern itself with KOS.

The relationship between the KOS and the rest of CS 10110 is governed by four principles:

- Only the KOS has access to the resources it controls. User calls to EOS may result in EOS calls to KOS, and S-language SINs may result in invocations of KOS microcode routines, but neither EOS nor user programs may directly manipulate resources controlled by KOS.
- The KOS is passive. It responds to calls from the EOS, to microcode invocations, and to Event Signals, but it initiates no action on its own.

Petitioner Apple Inc. - Exhibit 1004, p. 5315

— The KOS is invisible to all system users but the EOS. KOS does not affect the logical behavior of a Process 610 and is noticeable to users only with regard to the speed with which a Process 610 executes on CS 10110.

As discussed above, KOS manages both physical and virtual resources. The physical resources and some of the virtual resources are visible only within KOS; others of the virtual resources are provided to EOS. Each virtual resource has two main parts: a set of data bases that contain the virtual resource's state, and a set of routines that manipulate the virtual resource. The set of routines for a virtual resource are termed the resource's manager. The routines may be KOS Procedures 40308, or they may be KOS Microcode 40313. As mentioned, in some cases, KOS uses separate Processes 610 to manage the resources.

For the purposes of this specification, the resources managed by KOS fall into two main groups: those associated with objects, and those associated with Processes 610. In the following, first those resources associated with objects, and then those associated with Processes 610 are discussed.

B. Objects and Object Management (Fig. 404)

The virtual resources termed objects are defined by KOS and manipulated by EOS and KOS. Objects as seen by EOS have five properties:

— A single UID that identifies the object throughout the object's life and specifies what Logical Allocation Unit (LAU) the object belongs to.
— A set of attributes that describe the object and limit access to it.
— Bit-addressable contents. I the present embodiment, the contents may range from 0 to (2**32) — 1 bits in length. Any bit in the contents may be addressed by an offset.
— Objects may be created.
— Objects may be destroyed.

All of the data and Procedures 602 in a CS 10110 are contained in objects. Any process 610 executing on a CS 10110 may use a UID-off set address to attempt to access data or Procedures 602 in certain objects on any CS 10110 accessible to the CS 10110 on which Process 610 is executing. The objects which may be thus accessed by any Process 610 are those having UIDs which are guaranteed unique for all present and future CS 10110. Objects with such unique UIDs thus form a single address space which is at least potentially accessible to any process 610 executing on any CS 10110. As will be explained in detail later, whether a Process 610 can in fact access an object in this single address space depends on whether Process 610 has access rights to the object. Other objects, whose UIDs are not unique, may be accessed only by Processes 610 executing on CSs 10110 or groups of CSs 10110 for which the non-unique UID is in fact unique. No two objects accessible to a CS 10110 at a given time may have identical UIDs.

The following discussion of objects will first deal with objects as they are seen directly by EOS and indirectly by user programs, and then deal with objects as they appear to KOS.

Figure 404 illustrates how objects appear to EOS. The object has three parts: the UID 40401, the Attributes 40404, and the Contents, 40406. The object's contents reside in a Logical Allocation Unit (LAU), 40405. UID 40401 has two parts: a LAU Identifier (LAUID) 40402 that indicates what LAU 40405 the object is on, and the Object Serial Number (OSN) 40403, which specifies the object in LAU 40405.

The EOS can create an object on a LAU 40405, and given the object's UID 40401, can destroy the object. In addition, EOS can read and change an object's Attributes 40404. Any Process 610 executing on a CS 10110 may reference information in an object by specifying the object's UID 40401 and the bit in the object at which the information begins. At the highest level, addresses in CS 10110 thus consist of a UID 40401 specifying an object and an offset specifying the number of bits into the object at which the information begins. As will be explained in detail below, KOS translates such UID-offset addresses into intermediate forms called AON-offset addresses for use in JP 10114 and into page number-displacement addresses for use in referencing information which has been copied into MEM 10112.

The physical implementation and manipulation of objects is restricted solely to KOS. For instance, objects and their attributes are in fact stored in Secondary Storage 10124. When a program references a portion of an object, KOS copies that portion of the object from Secondary Storage 10124 into MEM 10112, and if the portion in MEM 10112 is changed, updates the copy of the object in Secondary Storage 10124. EOS and user programs cannot control the location of an object in Secondary Storage 10124 or the location of the copy of a portion of an object in MEM 10112, and therefore can access the object only by means of KOS.

While EOS cannot control the physical implementation of an object, it can provide KOS with information that allows KOS to manage objects more effectively. Such information is termed hints. For instance, KOS generally copies a portion of an object into MEM 10112 only if a Process 610 references information in the object. However, EOS schedules Process 610 execution, and therefore can predict that certain objects will be required in the near future. EOS can pass this information on to KOS, and KOS can use the information to decide what portions of objects to copy into MEM 10112.

a. Objects and User Programs (fig. 405)

As stated above, user programs manipulate objects, but the objects are generally not directly visible to user programs. Instead, user programs use symbols such as variable names or other references to refer to

Petitioner Apple Inc. - Exhibit 1004, p. 5316

data stored in objects or file names to refer to the objects themselves. The discussion of Namespace has already illustrated how CS 10110 compilers translate variable names appearing in statements in Procedures 602 into Names, i.e., indexes of NTEs 30401, how Name Resolve microcode resolves NTE 30401 into Logical Descriptors 27116, and how ATU 10228 translates Logical Descriptors 27116 into locations in MEM 10112 containing copies of the portions of the objects in which the data represented by the variables resides.

The translation of filenames to UIDs 40401 is accomplished by EOS. EOS maintains a filename translation table which establishes a relationship between a system filename called a pathname and the UID 40401 of the object containing the file's data, and thereby associates the pathname with the object. A Pathname is a sequence of ASCII characters which identifies a file to a user of CS 10110. Each pathname in a given CS 10110 must be unique. Figure 405 shows the filename translation table. Referring to that figure, when a user gives pathname 40501 to the EOS, EOS uses Filename Translation Table 40503 to translate pathname 40501 into UID 40401 for object 40504 containing the file. An object in CS 10110 may thus be identified in two ways: by means of its UID 40401 or by means of a Pathname 40501. While an object has only a single UID 40401 throughout its life, the object may have many Pathnames 40501. All that is required to change an object's pathname 40501 is the substitution of one Pathname 40501 for another in the object's Entry 40502 in Filename Translation Table 40503. One consequence of the fact that an object may have different Pathnames 40501 during its life is that when a program uses a Pathname 40501 to identify an object, a user of CS 10110 may make the program process a different object simply by giving the object which formerly had Pathname 40501 which appears in the program a new Pathname 40501 and giving the next object to be processed the Pathname 40501 which appears in the program.

In the present embodiment, an object may contain only a single file, and consequently, a Pathname 40501 always refers to an entire object. In other embodiments, a Pathname 40501 may refer to a portion of an object, and in such embodiments, Filename Translation Table 40503 will associate a Pathname 40501 with a UID-offset address specifying the beginning of the file.

b. UIDs 40401 (Fig. 406)

UIDs 40401 may identify objects and other entities in CS 10110. Any entity identified by a UID 40401 has only a single UID throughout its life. Figure 406 is a detailed representation of a CS 10110 UID 40401. UID 40401 is 80 bits long, and has two fields. Field 40402, 32 bits long, is the Logical Allocation Unit Identifier (LAUID). It specifies LAU 40405 containing the object. LAUID 40402 is further subdivided into two subfields: LAU Group Number (LAUGN) 40607 and LAU Serial Number (LAUSN) 40605. LAUGN 40607 specifies a group of LAUs 40405, and LAUSN 40605 specifies a LAU 40405 in that group. Purchasers of CS 10110 may obtain LAUGNs 40607 from the manufacturer. The manufacturer guarantees that he will assign LAUGN 40607 given the purchaser to no other CS 10110, and thus these LAUGNs 40607 may be used to form UIDs 40401 which will be unique for all CSs 10110. Field 40604, 48 bits long, is the Object Serial Number (OSN). It specifies the object in LAU 40405.

UIDs 40401 are generated by KOS Procedures 602.

There are two such procedures 602, one which generates UIDs 40401 which identify objects, and another which generates UIDs 40401 which identify other entities in CS 10110. The former Procedure 602 is called Generate Object UID, and the latter Generate Non-object UID. The Generate Object UID Procedure 602 is called only by the KOS Create Object Procedure 602. Create Object Procedure 602 provides Generate Object UID Procedure 602 with a LAUID 40402, and Generate Object UID Procedure 602 returns a UID 40401 for the object. In the present embodiment, UID 40401 is formed by taking the current value of the architectural clock, contained in a location in MEM 10112, forming an OSN 40403 from the architectural clock's current value, and concatenating OSN 40403 to LAUID 40402.

Generate Non-object UID Procedure 602 may be invoked by EOS to provide a UID 40401 which does not specify an object. Non-object UIDs 40401 may be used in CS 10110 wherever a unique label is required. For example, as will be explained in detail later, all Virtual processors 612 which are available to CS 10110 have non-object UIDs 40401. All such non-object UIDs 40401 have a single LAUSN 40607, and thus, EOS need only provide a LAUGN 40605 as an argument. Generate Non-object UID Procedure 602 concatenates LAUGN 40605 with the special LAUSN 40607, and LAUID 40402 thus produced with an OSN 40403 obtained from the architectural clock. In other embodiments, OSNs 40403 for both object and non-object UIDs 40401 may be generated by other means, such as counters.

CS 10110 also has a special UID 40401 called the Null UID 40401. The Null UID 40401 contains nothing but 0 bits, and is used in situations which require a UID value which cannot represent an entity in CS 10110.

c. Object Attributes

What a program can do with an object is determined by the object's Attributes 40404. There are two kinds of Attributes 40404: Object Attributes and Control Attributes. Object Attributes describe the object's contents; Control Attributes control access to the object. Objects may have Attributes 40404 even though they have no Contents 40406, and in some cases, objects may even exist solely for their Attributes 40404.

For the purposes of this discussion, there are two kinds of Object Attributes: the Size Attribute and the Type Attributes.

An object's Size Attribute indicates the number of bits that the object currently contains. On each

Petitioner Apple Inc. - Exhibit 1004, p. 5317

reference to an object's Contents 40406, KOS checks to make sure that the data accessed does not extend beyond the end of the object. If it does, the reference is aborted.

The Type Attributes indicate what kind of information the object contains and how that information may be used. There are three categories of Type Attributes: the Primitive Type Attributes, the Extended Type Attribute, and the Domain of Execution attribute. An object's Primitive Type Attribute indicates whether the object is a data object, a Procedure Object 608, an Extended Type Manager, or an S-interpreter. As their names imply, data objects contain data and Procedure Objects 608 contain Procedures 602. Extended Type Managers (ETMs) are a special type of Procedure Object 608 whose Procedures 608 may perform operations solely on objects called Extended Type Objects. Extended Type Objects (ETOs) are objects which have an Extended Type Attribute in addition to their Primitive Type Attribute; for details, see the discussion of the Extended Type Attribute below. S-interpreters are objects that contain interpreters for S-languages. In the present embodiment, the interpreters consist of dispatch tables and microcode, but in other embodiments, the interpreters may themselves be written in high-level languages. Like the Length Attribute, the Primitive Type Attributes allow KOS to ensure that a program is using an object correctly. For instance, when the KOS executes a call for a Procedure 602 it checks whether the object specified by the call is a Procedure Object 608. If it is not, the call fails.

d. Attributes and Access Control

The remaining Object Attributes and the Control Attributes are all part of CS 10110's Access Control System. The Access Control System is discussed in detail later; here, it is dealt with only to the extent required for the discussion of objects. In CS 10110, an access of an object occurs when a Process 610 fetches SINs contained in a Procedure Object 608, reads data from an object, writes data to an object, or in some cases, when Process 610 transfers control to a Procedure 602. The Access Control System checks whether a Process 610 has the right to perform the access it is attempting. There are two kinds of access in CS 10110, Primitive Access and Extended Access. Primitive Access is access which the Access Control System checks on every reference to an object by a Process 610; Extended Access is access that is checked only on user request. Primitive access checks are performed on every object; extended access checks may be performed only on ETOs, and may be performed only by Procedures 602 contained in ETMs.

The means by which the Access Control System checks a Process 610's access to an object are Process 610's subject and the object's Access Control Lists (ACLs). Each Process 610 has a subject made up of four UIDs 40401. These UIDs 40401 specify the following:

- The user for whom Process 610 was created. This UID 40401 is termed the principal component of the subject.
- Process 610 itself. This UID 40401 is termed the process component.
- The domain in which Process 610 is currently executing. This UID 40401 is termed the domain component.
- A user-defined subgroup of subjects. This UID 40401 is termed the tag component.

A domain is a group of objects which may potentially be accessed by any Process 610 which is executing a Procedure 602 in one of a group of Procedure Objects 608 or ETMs. Each Procedure Object 608 or ETM has a Domain of Execution (DOE) Attribute. This attribute is a UID 40401, and while a Process 610 is executing a Procedure 602 in that Procedure Object 608 or ETM, the DOE attribute UID 40401 is the domain component in Process 610's subject. The DOE attribute thus defines a group of objects which may be accessed by a Process 610 executing Procedures 602 from Procedure Object 608. The group of objects is called Procedure Object 608's domain. As may be seen from the above definition, a subject's domain component may change on any call to or return from a Procedure 602. The tag component may change whenever the user desires. The principal component and the process component, on the other hand, do not change for the life of Process 610.

The ACLs which make up the other half of the Access Control System are attributes of objects. Each ACL consists of a series of Entries (ACLE), and each ACLE has two parts: a Subject Template and a set of Access Privileges. The Subject Template defines a group of subjects, and the set of Access Privileges define the kinds of access that subjects belonging to the group have to the object. To check whether an access to an object is legal, the KOS examines the ACLs. It allows access only if it finds an ACLE whose Subject Template matches the current subject of Process 610 which wishes to make the access and whose set of Access Privileges includes the kind of access desired by Process 610. For example, a Procedure Object 608 may have an ACL with two entries: one whose Subject Template allows any subject access, and whose set of Access Privileges allows only Execute Access, and another whose Subject Template allows only a single subject access and whose set of Access Privileges allows Read, Write, and Execute Access. Such an ACL allows any user of CS 10110 to execute the Procedures 602 in Procedure Object 608, but only a specified Process 610 belonging to a specified user and executing a specified group of Procedures 602 may examine or modify the Procedures 602 in the Procedure Object 608.

There are two kinds of ACLs. All objects have Primitive Access Control Lists (PACLs); ETOs may in addition have Extended Access Control Lists (EACLs). The subject portion of the ACLE is the same in all ACLs; the two kinds of list differ in the kinds of access they control. The access controlled by the PACL is defined by KOS and is checked by KOS on every attempt to gain such access; the access controlled by the EACL is defined by the user and is checked only when the user requests KOS to do so.

Petitioner Apple Inc. - Exhibit 1004, p. 5318

e. Implementation of Objects

1. Introduction (Fig. 407, 408)

The user of a CS 10110 need only concern himself with objects as they have just been described. In order for a Process 610 to reference an object, the object's LAU 40405 must be accessible from CS 10110 upon which Process 610 is running, Process 610 must know the object's UID 40401, and Process 610's current subject must have the right to access the object in the desired manner. Process 610 need know neither how the object's Contents 40406 and Attributes 40404 are stored on CS 10110's physical devices nor the methods CS 10110 uses to make the object's Contents 40406 and Attributes 40404 available to Process 610.

The KOS, on the other hand, must implement objects on the physical devices that make up CS 10110. In so doing, it must take into account two sets of physical limitations:

— In logical terms, all CSs 10110 have a single logical memory, but the physical implementation of memory in the system is hierarchical: a given CS 10110 has rapid access to a relatively small MEM 10112, much slower access to a relatively large amount of slow Secondary Storage 10124, and very slow access to LAUs 40405 on other accessible CSs 10110.

— UIDs 40401, and even more, subjects, are too large to be handled efficiently on JP 10114's internal data paths and in JP 10114's registers.

The means by which the KOS overcomes these physical limitations will vary from embodiment to embodiment. Here, there are presented first an overview and then a detailed discussion of the means used in the present embodiment.

The physical limitations of the memory are overcome by means of a Virtual Memory system. The Virtual Memory System creates a one-level logical memory by automatically bringing copies of those portions of objects required by executing Processes 610 into MEM 10112 and automatically copying altered portions of objects from MEM 10112 back to Secondary Storage 10124. Objects thus reside primarily in Secondary Storage 10124, but copies of portions of them are made available in MEM 10112 when a Process 610 makes a reference to them. Besides bringing portions of objects into MEM 10112, when required, the Virtual Memory System keeps track of where in MEM 10112 the portions are located, and when a Process 610 references a portion of an object that is in MEM 10112, the Virtual Memory System translates the reference into a physical location in MEM 10112.

JP 10114's need for smaller object identifiers and subject identifiers is satisfied by the use of internal identifiers called Active Object Numbers (AONs) and Active Subject Numbers (ASNs) inside JP 10114. Each time a UID 40401 is moved from MEM 10112 into JP 10114's registers, it is translated into an AON, and the reverse translation takes place each time an AON is moved from a JP 10114's registers to MEM 10112. Similarly, the current subjects of Processes 610 which are bound to Virtual Processors 612 are translated from four UIDs 40401 into small integer ASNs, and when Virtual Processor 612 is bound to JP 10114, the ASN for the subject belonging to Virtual Processor 612's process 610 is placed in a JP 10114 register. The translations from UID 40401 to AON and vice-versa, and from subject to ASN are performed by KOS.

When KOS translates UIDs 40401 to AONs and vice-versa, it uses AOT 10712. An AOT 10712 Entry (AOTE) for an object contains the object's UID 40401, and the AOTE's index in AOT 10712 is that object's AON. Thus, given an object's AON, KOS can use AOT 10712 to determine the object's UID 40401, and given an obect's UID 40401, KOS can use AOT 10712 to determine the object's AON. If the object has not been referenced recently, there may be no AOTE for the object, and thus no AON for the object's UID 40401. Objects that have no AONs are called inactive objects. If an attempt to convert a UID 40401 to an AON reveals that the object is inactive, an Inactive Object Fault results and KOS must activate the object, that is, it must assign the object an AON and make an AOTE for it.

KOS uses AST 10914 to translate subjects into ASN's. When a Process 610's subject changes, AST 10914 provides Process 610 with the new subject's ASN. A subject may presently have no ASN associated with it. Such subjects are termed inactive subjects. If a subject is inactive, an attempt to translate the subject to an ASN causes KOS to activate the subject, that is, to assign the subject an ASN and make an entry for the subject in AST 10914.

In order to achieve efficient execution of programs by Processes 610, KOS accelerates information that is frequently used by executing processes 610. There are two stages of acceleration:

— Tables that contain the information are wired into MEM 10112, that is, the Virtual Memory System never uses MEM 10112 space reserved for the tables for other purposes.

— Special hardware devices in JP 10114 contain portions of the information in the tables.

MHT 10716, AOT 10712, and AST 10914 are examples of the first stage of acceleration. As previously mentioned, these tables are always present in MEM 10112. Address Translation Unit (ATU) 10228 is an example of the second stage. As previously explained, ATU 10228 is a hardware cache that contains copies of the most recently used MHT 10716 entries. Like MHT 10716, it translates AON offset addresses into the MEM 10112 locations that contain copies of the data that the UID-offset address corresponding to the AON-offset address refers to ATU 10228 is maintained by KOS Logical Address Translation (LAT) microcode.

Figure 407 shows the relationship between ATU 10228, MEM 10112, MHT 10716, and KOS LAT microcode 40704. When JP 10114 makes a memory reference, it passes AON-offset Address 40705 to ATU 10228. If ATU 10228 contains a copy of MHT 10716's entry for Address 40705, it immediately produces the corresponding MEM 10112 Address 40706 and transmits the address to MEM 10112. If there is no copy,

Petitioner Apple Inc. - Exhibit 1004, p. 5319

ATU 10228 produces an ATU Miss Event Signal which invokes LAT microcode 40704 in JP 10114. LAT microcode 40704 obtains the MHT entry that corresponds to the AON-offset address from MHT 10716, places the entry in ATU 10228, and returns. JP 10114 then repeats the reference. This time, there is an entry for the reference, and ATU 10228 translates the AON address into the address of the copy of the data contained in MEM 10112.

The relationship between KOS table, hardware cache, and microcode just described is typical for the present embodiment of CS 10110. The table (in this case, MHT 10716), is the primary source of information and is maintained by the Virtual Memory Manager Process, while the cache accelerates portions of the table and is maintained by KOS microcode that is invoked by event signals from the cache.

AOT 10712, AST 10914, and MHT 10716 share another characteristic that is typical of the present embodiment of CS 10110: the tables are constructed in such a fashion that the table entry that performs the desired translation is located by means of a hash function and a hash table. The hash function translates the large UID 40401, subject, or AON into a small integer. This integer is the index of an entry in the hash table. The contents of the hash table entry is an index into AOT 10712, AST 10914, or MHT 10716, as the case may be, and these tables are maintained in such a fashion that the entry corresponding to the index provided by the hash table is either the entry that can perform the desired translation or contains information that allows KOS to find the desired entry. The entries in the tables furthermore contain the values they translate. Consequently, KOS can hash the value, find the entry, and then check whether the entry is the one for the hashed value. If it is not, KOS can quickly go from the entry located by the hash table to the correct entry.

Figure 408 shows how hashing works in AST 10914 in the present embodiment. In the present embodiment, Subject 40801, i.e., the principal, process, and domain components of the current subject, are input into Hash Function 40802. Hash Function 40802 produces the index of an entry in ASTHT 10710. ASTHT Entry 40504 in turn contains the index of an Entry (ASTE) 40806 in AST 10914 These ASTE 40806 indexes are ASNs. ASTE 40806 contains the principal, process, and domain components of some subject and a link field pointing to ASTE 40806'. ASTE 40806' has 0 in its link field, which indicates that it is the last link in the chain of ASTES begining with ASTE 40806. If the hashing of a subject yields ASTE 40806, KOS compares the subject in ASTE 40806 with the hashed subject; if they are identical, ASTE 40806's index in AST 10914 is the subject's ASN. If they are not identical, KOS uses the link in ASTE 40806 to find ASTE 40806'. It compares the subject in ASTE 40806' with the hashed subject; if they are identical, ASTE 40806''s AST index is the subject's ASN; otherwise, ASTE 40806' is the last entry in the chain, and consequently, there is no ASTE 40806 and no ASN for the hashed subject.

In the following, we will discuss the implementation of objects in the present embodiment in detail, beginning with the implementation of objects in Secondary Storage 10124 and proceeding then to CS 10110's Active Object Management System, the Access Control System, and the Virtual Memory System.

2. Objects in Secondary Storage 10124 (Figs. 409, 410)

As described above, objects are collected into LAUs 40405. The objects belonging to a LAU 40405 are stored in Secondary Storage 10124. Each LAU 40405 contains an object whose contents are a table called the Logical Allocation Unit Directory (LAUD). As its name implies, the LAUD is a directory of the objects in LAU 40405. Each object in LAU 40405, including the object containing the LAUD, has an entry in the LAUD. Figure 409 shows the relationship between Secondary Storage 10124, LAU 40405, the LAUD, and objects. LAU 40405 resides on a number of Storage Devices 40904. LAUD Object 40902' in LAU 40405 contains LAUD 40903. Two LAUDEs 40906 are shown. One contains the attributes of LAUD Object 40902 and the location of its contents, and the other contains the attributes of LAUD Object 40902' containing LAUD 40903 and the location of its contents.

KOS uses a table called the Active LAU Table (ALAUT) to locate the LAUD belonging to LAU 40405. Figure 410 illustrates the relationship between ALAUT 41001, ALAUT Entries 41002, LAUs 40405, and LAUD Objects 40902'. Each LAU 40405 accessible to CS 10110 has an Entry (ALAUTE) 41002 in ALAUT 41001. ALAUTE 41002 for LAU 40405 includes LAU 40405's LAUID 40402 and UID 40401 of LAU 40705's LAUD Object 40902'. Hence, given an object's UID 40401, KOS can use UID 40401's LAUID 40402 to locate ALAUTE 41002 for the object's LAU 40405, and can use ALAUTE 41002 to locate LAU 40405's LAUD 40903. Once LAUD 40903 has been found, OSN portion 40402 of the object's UID 40401 provides the proper LAUDE 40906, and LAUDE 40906 contains object's attributes and the location of its contents.

LAUD 40903 and the Procedures 602 that manipulate it belong to a part of KOS termed the Inactive Object Manager. The following discussion of the Inactive Object Manager will begin with the manner in which an object's contents are represented on Secondary Storage 10124, will then discuss LAUD 40903 in detail, and conclude by discussing the operations performed by Inactive Object Manager Procedures 602.

a.a. Representation of an Object's Contents on Secondary Storage 10124

In general, the manner in which an object's contents are represented on Secondary Storage 10124 depends completely on the Secondary Storage 10124. If a LAU 40405 is made up of disks, then the object's contents will be stored in disk blocks. As long as KOS can locate the object's contents, it makes no difference whether the storage is contiguous or non-contiguous.

In the present embodiment, the objects' contents are stored in files created by the Data General

Petitioner Apple Inc. - Exhibit 1004, p. 5320

Advance Operating System (AOS) procedures executing on IOS 10116 These procedures manage files that contain objects' contents for KOS. In future CSs 10110, the representation of an object's contents on Secondary Storage 10124 will be managed by a portion of KOS.

b.b. LAUD 40903 (fig. 411, 412)

Figure 411 is a conceptual illustration of LAUD 40903. LAUD 40903 has three parts: LAUD Header 41102, Master Directory 41105, and LAUD Entries (LAUDEs) 40906. LAUD Header 41102 and Master Directory 41105 occupy fixed locations in LAUD 40903, and can therefore always be located from the UID 40401 of LAUD 40903 given in ALAUT 41001. The locations of LAUDEs 40906 are not fixed, but the entry for an individual object can be located from Master Directory 41105.

Turning first to LAUD Header 41102, LAUD Header 41102 contains LAUID 40402 belonging to LAU 40405 to which LAUD 40903 belongs and OSN 40403 of LAUD 40903. As will be explained in greater detail below, KOS can use OSN 40403 to find LAUDE 40906 for LAUD 40903.

Turning now to Master Directory 41105, Master Directory 41105 translates an object's OSN 40403 into the location of the object's LAUDE 40906. Master Directory 41105 contains one Entry 41108 for each object in LAU 40505. Each Entry has two fields: OSN Field 41106 and Offset Field 41107. OSN Field 41106 contains OSN 40403 for the object to which Entry 41108 belongs; Offset Field 41107 contains the offset of the object's LAUDE 40906 in LAUD 40903. KOS orders Entries 41108 by increasing OSN 40403, and can therefore use binary search means to find Entry 41108 containing a given OSN 40403. Once Entry 41108 has been located, Entry 41108's Offset Field 41107, combined with LAUD 40903's OSN 40403, yields the UID offset address of the object's LAUDE 40906.

Once KOS knows the location of LAUDE 40906 it can determine an object's Attributes 40404 and the location of its Contents 40406. Figure 411 gives only an overview of LAUDE 40906's general structure. LAUDE 40906 has three components: a group of fields of fixed size 41109 that are present in every LAUDE 40906, and two variable sized components, one, 41139, containing entries belonging to the object's PACL, and another, 41141, containing the object's EACL.

As the preceding descriptions of the LAUD's components imply, the number of LAUDEs 40906 and Master Directory Entries 41108 varies with the number of objects in LAU 40405. Furthermore, the amount of space required for an object's EACL and PACL varies from object to object. KOS deals with this problem by including Free Space 41123 in each LAUD 40903. When an object is created, or when an object's ACLs are expanded, the Inactive Object Manager expands LAUD 40903 only if there is no available Free Space 41123; if there is Free Space 41123, the Inactive Object Manager takes the necessary space from Free Space 41123; when an object is deleted or an object's ACLs shortened, the Inactive Object Manager returns the unneeded space to Free Space 41123.

Figure 412 is a detailed representation of a single LAUDE 40906. Figure 412 presents those fields of LAUDE 40906 which are common to all embodiments of CS 10110; fields which may vary from embodiment to embodiment are ignored. Starting at the top of Figure 412, Structure Version Field 41209 contains information by which KOS can determine which version of LAUDE 40906 it is dealing with. Size Field 41211 contains the Size Attribute of the object to which LAUDE 40906 belongs. The Size Attribute specifies the number of bits currently contained in the object. Lock Field 41213 is a KOS lock. As will be explained in detail in the discussion of Processes 610, Lock Field 41213 allows only one Process 610 to read or write LAUDE 40906 at a time, and therefore keeps one Process 610 from altering LAUDE 40906 while another Process 610 is reading LAUDE 40906. File Identifier 41215 contains a system identifier for the file which contains the Contents 40406 of the object to which LAUDE 40906 belongs. The form of File Identifier 41215 may vary from embodiment to embodiment; in the present embodiment, it is an AOS system file identifier. UID Field 41217 contains UID 40401 belonging to LAUDE 40906's object. Primitive Type Field 41219 contains a value which specifies the object's Primitive Type. The object may be a data object, a Procedure Object 608, an ETM, or an S-interpreter object. AON Field 41221 contains a valid value only when LAUDE 40906's object is active, i.e., has an entry in AOT 10712. AON Field 41221 then contains the object's AON. If the object is an ETO, Extended Type Attribute Field 41223 contains the UID 40401 of the ETO's ETM. Otherwise, it contains a Null UID 40401. Similarly, if the object is a Procedure Object 608 or an ETM, Domain of Execution Attribute Field 41225 contains the object's Domain of Execution Attribute.

The remaining parts of LAUDE 40906 belong to the Access Control System and will be explained in detail in that discussion. Attribute Version Number Field 41227 contains a value indicating which version of ACLEs this LAUDE 40906 contains, PACL Size Field 41229 and EACL Size Field 41231 contain the sizes of the respective ACLs, PACL Offset Field 41233 and EACL Offset Field 41235 contain the offsets in LAUD 40903 of additional PACLEs 41139 and EACLEs 41141, and fixed PACLEs 41237 contains the portion of the PACL which is always included in LAUDE 40906.

3. Active Objects (fig. 413)

An active object is an object whose UID 40401 has an AON associated with it. In the present embodiment, each CS 10110 has a set of AONs' KOS associates these AONs with UIDs 40401 in such fashion that at any given moment, an AON in a CS 10110 represents a single UID 40401. Inside FU 10120, AONs are used to represent UIDs CS 10110. In the present embodiment, the AON is represented by 14 bits. A 112-bit UID-offset address (80 bits for UID 40401 and 32 for the offset) is thus represented inside FU 10120

Petitioner Apple Inc. - Exhibit 1004, p. 5321

by a 46-bit AON-offset address (14 bits for the AON and 32 bits for the offset).

A CS 10110 has far fewer AONs than there are UIDs 40401. KOS multiplexes a CS 10110's AONs among those objects that are being referenced by CS 10110 and therefore require AONs as well as UIDs 40401. While a given AON represents only a single UID 40401 at any given time, at different times, a UID 40401 may have different AONs associated with it.

Figure 413 provides a conceptual representation of the relationship between AONs and UIDs 40401. Each CS 10110 has potential access to 2**80 UIDs 40401. Some of these UIDs, however, represent entities other than objects, and others are never associated with any entity. Each CS 10110 also has a set of AONs 41303 available to it. In the present embodiment, this set may have up to 2**14 values. Since the AONS are only used internally, each CS 10110 may have the same set of AONs 41303. Any AON 41304 in set of AONs 41303 may be associated with a single UID 40401 in set of object UIDs 41301. At different times, an AON 41304 may be associated with different UIDs 40401.

As mentioned above, KOS associates AONs 41304 with UIDs 40401. It does so by means of AOT 10712. Each AOT entry (AOTE) 41306 in AOT 10712 associates a UID 40401 with an AON 41304. AON 41304 is the index of AOTE 41306 which contains UID 40401. Until AOTE 41306 is changed, the AON 41304 which is the index of AOTE 41306 containing UID 40401 represents UID 40401. AOT 10712 also allows UIDs 40401 to be translated into AONs 41303 and vice-versa. Figure 413 illustrates the process for UID-offset Address 41308 and AON-offset Address 41309. AOTE 41306 associates AON 41304 in AON-offset Address 41309 with UID 40401 in UID-offset Address 41308, and Addresses 41308 and 41309 have the same Offset 41307. Consequently, AON-offset Address 41309 represents UID-offset Address 41308 inside JP 10114. Since both addresses use the same Offset, Address 41309 can be translated into address 41308 by translating Address 41309's AON 41304 into Address 41308's UID 40401, and Address 41308 can be translated into Address 41309 by the reverse process. In both cases, the translation is performed by finding the proper AOTE 41306.

The process by which an object becomes active is called object activation. A UID-offset Address 41308 cannot be translated into an AON-offset Address 41309 unless the object to which UID 40401 of UID-offset Address 41308 belongs is active. If a Process 610 attempts to perform such a translation using a UID 40401 belonging to an inactive object, an Inactive Object Fault occurs. KOS handles the fault by removing Process 610 that attempted the translation from JP 10114 until a special KOS Process called the Object Manager Process has activated the object. After the object has been activated, Process 610 may return to JP 10114 and complete the UID 40401 to AON 41304 translation.

The portion of KOS that manages active objects is called the Active Object Manager (AOM). Parts of the AOM are Procedures 602, and parts of it are microcode routines. The high-level language components of the AOM may be invoked only by KOS processes 610. KOS Active Object Manager Process 610 performs most of the functions involved in active object management.

a.a. UID 40401 to AON 41304 Translation

Generally speaking, in CS 10110, addresses stored in MEM 10112 and Secondary Memory 10124 are stored as UID offset addresses. The only form of address that FU 10120 can translate into a location in MEM 10112 is the AON-offset form. Consequently, each time an address is loaded from MEM 10112 into a FU 10120 register, the address must be translated from a UID-offset address to an AON-offset address. The reverse translation must be performed each time an address is moved from a FU 10120 register back into memory.

Such translations may occur at any time. For example, a running Virtual Processor 612 performs such a translation when the Process 610 being executed by Virtual Processor 612 carries out an indirect memory reference. An indirect memory reference is a reference which first fetches a pointer, that is, a data item whose value is the address of another data item, and then uses the address contained in the pointer to fetch the data itself. In CS 10110, pointers represent UID-offset addresses. Virtual Processor 612 performs the indirect memory reference by fetching the pointer from MEM 10112, placing it in FU 10120 registers, translating UID 40401 represented by the pointer into AON 41304 associated with it, and using the resulting AON-offset address to access the data at the location specified by the address.

Most such translations, however, occur when Virtual Processor 612 state is saved or restored. For instance, when one Process 610's Virtual Processor 612 is removed from JP 10114 and another Process 610's Virtual Processor 612 is bound to JP 10114, the state of Virtual Processor 612 being removed from JP 10114 is stored in memory, and the state of Virtual Processor 612 being bound to JP 10114 is moved into JP 10114's registers. Because only UID-offset addresses may be stored in memory, all of the AON-offset addresses in the state of Virtual Processor 612 which is being removed from JP 10114 must be translated into UID-offset addresses. Similarly, all of the UID-offset addresses in the state of Virtual Processor 612 being bound to JP 10114 must be translated into AON-offset addresses before they can be loaded into FU 10120 registers.

C. The Access Control System

As mentioned in the introduction to objects, each time a process 610 accesses data or SINs in an object, the KOS Access Control System checks whether Process 610's current subject has the right to perform the kind of access that Process 610 is attempting. If Process 610's current subject does not have the proper access, the Access Control System aborts the memory operation which Process 610 was attempting to

Petitioner Apple Inc. - Exhibit 1004, p. 5322

carry out. The following discussion presents details of the implementation of the Access Control System, beginning with subjects, then proceeding to subject templates, and finally to the means used by KOS to accelerate access checking.

a. Subjects

A Process 610's subject is part of process 610's state and is contained along with other state belonging to Process 610 in an object called a Process Object. Process Objects are dealt with at length in the detailed discussion of Processes 610 which follows the discussion of objects. While a subject has, as mentioned above, four components, the principal component, the process component, the domain component, and the tag component, the Access Control System in the present embodiment of CS 10110 assigns values to only the first three components and ignores the tag component when checking access.

In the present embodiment, the UIDs 40401 which make up the components of a Process 610's subject are the UIDs 40401 of objects containing information about the entities represented by the UIDs 40401. The principal component's UID 40401 represents an object called the Principal Object. The Principal Object contains information about the user for whom Process 610 was created. For example, the information might concern what access rights the user had to the resources of CS 10110, or it might contain records of his use of CS 10110. The process component's UID 40401 represents the Process Object, while the domain component's UID 40401 represents an object called the Domain Object. The Domain Object contains information which must be accessible to any Process 610 whose subject has the Domain Object's UID 40401 as its domain component. Other embodiments of CS 10110 will use the tag component of the subject. In these embodiments, the tag component's UID 40401 is the UID 40401 of a Tag Object containing at least such information as a list of the subjects which make up the group of subjects represented by the tag component's UID.

b. Domains

As stated above, the subject's domain component is the domain of execution attribute belonging to the Procedure Object 608 or ETM whose code is being executed when the access request is made. The domain component of the subject thus gives Process 610 to which the subject belongs potential access to the group of objects whose ACLs have ACLEs with subject templates containing domain components that match the DOE attribute. This group of objects is the domain defined by the Procedure Object 608 or ETM's DOE attribute. When a Process 610 executes a Procedure 602 from a Procedure Object 608 or ETM with a given DOE attribute, Process 610 is said to be executing in the domain defined by that DOE attribute. As may be inferred from the above, different Procedure Objects 608 or ETMs may have the same DOE attribute, and objects may have ACLEs which make them members of many different domains.

In establishing a relationship between a group of Procedure Objects 608 and another group of objects, a domain allows a programmer using CS 10110 to ensure that a given object is read, executed, or modified only by a certain set of Procedures 602. Domains may thus be used to construct protected subsystems in CS 10110. One example of such a protected subsystem is KOS itself: the objects in CS 10110 which contain KOS tables all have ACLs whose domain template components match only the DOE which represents the KOS domain. The only Procedure Objects 608 and ETMs which have this DOE are those which contain KOS Procedures 602, and consequently, only KOS Procedures 602 may manipulate KOS tables.

Since an object may belong to more than one domain, a programmer may use domains to establish hierarchies of access. For example, if some of the objects in a first domain belong both to the first domain and a second domain, and the second domain's objects all also belong to the first domain, then Procedures 602 contained in Procedure Objects 608 whose DOEs define the first domain may access any object in the first domain, including those which also belong to the second domain, while those from Procedure Objects 608 whose DOEs define the second domain may access only those objects in the second domain.

c. Access Control Lists

As previously mentioned, the Access Control System compares the subject belonging to Process 610 making an access to an object and the kind of access Process 610 desires to make with the object's ACLs to determine whether the access is legal. The following discussion of the ACLs will first deal with Subject Templates, since they are common to all ACLs, and then with PACLs and EACLs.

1. Subject Templates (Fig. 416)

Figure 416 shows Subject Templates, PACL Entries (PACLEs), and EACL Entries (EACLEs). Turning first to the Subject Templates, Subject Template 41601 consists of four components, Principal Template 41606, Process Template 41607, Domain Template 41609, and Tag Template 41611. Each template has two fields, Flavor Field 41603, and UID Field 41605. Flavor Field 41603 indicates the way in which the template to which it belongs is to match the corresponding component of the subject for Process 610 attempting the access. Flavor Field 41603 may have one of three values: match any, match one, match group. If Flavor Field 41603 has the value match any, any subject component UID 40401 matches the template, and the Access Control System does not examine UID Field 41605. If Flavor Field 41603 has the value match one, then the corresponding subject component must have the same UID 40401 as the one contained in UID Field 41605. If Flavor Field 41603 has the value match group, finally, then UID Field 41605 contains a UID 40401 of an

Petitioner Apple Inc. - Exhibit 1004, p. 5323

object containing information about the group of subject components which the given subject component may match.

2. Primitive Access Control Lists (PACLs)

PACLs are made up of PACLEs 41613 as illustrated in Figure 416. Each PACLE 41613 has two parts: a subject template 41601 and an Access Mode Bits Field 41615. The values in Access Mode Bits Field 41615 define 11 kinds of access. The eleven kinds fall into two groups: Primitive Data Access and Primitive Non-data Access. Primitive Data Access controls what the subject may do with the object's Contents 40406; Primitive Non-data Access controls what the subject may do with the object's Attributes 40404.

There are three kinds of Primitive Data Access: Read Access, Write Access, and Execute Access. If a subject has Read Access, it can examine the data contained in the object; if the subject has Write Access, it can alter the data contained in the object; if it has Execute Access, it can treat the data in the object as a Procedure 602 and attempt to execute it. A subject may have none of these kinds of access, or any combination of the kinds. On every reference to an object, the KOS checks whether the subject performing the reference has the required Primitive Data Access.

Primitive Non-data Access to an object is required only to set or read an object's Attributes 40404, and is checked only when these operations are performed. The kinds of Non-data Access correspond to the kinds of Attributes 40404:

Attributes	Kind of Access
Object Attributes	get object attributes set object attributes
Primitive Control Attributes	get primitive control attributes set primitive control attributes
Extended Control Attributes	get extended control attributes set extended control attributes
ETM Access	use as ETM create ETO

The access rights for object attributes allow a subject to get and set the object attributes described previously. The access rights for primitive and extended control attributes allow a subject to get and set an object's PACL and EACL respectively.

An object may have any number of PACLEs 41613 in its PACL. The first five PACLEs 41613 in an object's PACL are contained in fixed PACLE Field 41237 of LAUDE 40906 for the object; the remainder are stored in LAUD 40903 at the location specified in PACL Offset Field 41233 of LAUDE 40906.

3. APAM 10918 and Protection Cache 10234 (Fig. 421)

Primitive non-data access rights are checked only when users invoke KOS routines that require such access rights, and extended access rights are checked only when users request such checks. Primitive data access rights, on the other hand, are checked every time a Virtual Processor 612 makes a memory reference while executing a Process 610. The KOS implementation of primitive data access right checking therefore emphasizes speed and efficiency. There are two parts to the implementation: APAM 10918 in MEM 10112, and Protection Cache 10234 in JP 10114. APAM 10918 is in a location in MEM 10112 known to KOS microcode. APAM 10918 contains primitive data access information copied from PACLEs 41613 which belong to active objects and whose Subject Template 41601 matches an active subject. Protection Cache 10234, in turn, contain copies of the information in APAM 10918 for the active subject of Process 610 whose Virtual Processor 612 is currently bound to JP 10114 and active objects referenced by Process 610. A primitive data access check in CS 10110 begins with Protection Cache 10234, and if the information is not contained in Protection Cache 10234, proceeds to APAM 10918, and if it is not there, finally, to the object's PACL. The discussion which follows begins with APAM 10918.

Figure 421 shows APAM 10918. APAM 10918 is organized as a two-dimensional array. The array's row indexes are AONs 41304, and its column indexes are ASNs. There is a row for each AON 41304 in CS 10110, and a column for each ASN. In Figure 421, only a single row and column are shown. Any primitive data access information in APAM 10918 for the object represented by AON 41304 j is contained in Row 42104, while Column 42105 contains any primitive data access information in APAM 10918 for the subject

Petitioner Apple Inc. - Exhibit 1004, p. 5324

represented by ASN k. APAM Entry (APAME) 42106 is at the intersection of Row 42104 and Column 42105, and thus contains the primitive data access information from that PACLE 41613 belonging to the object represented by AON 41304 j whose Subject Template 41601 matches the subject represented by ASN k.

An expanded view of APAME 42106 is presented beneath the epresentation of APAM 10918. APAME 42106 contains four 1-bit fields. The bits represent the kinds of primitive data access that the subject represented by APAME 42106's column index has to the object represented by APAME 42106's row index.

- Field 42107 is the Valid Bit. If the Valid Bit is set, APAME 42106 contains whatever primitive data access information is available for the subject represented by the column and the object represented by the row. The remaining fields in APAME 42106 are meaningful only if Valid Bit 42107 is set.
- Field 42109 is the Execute Bit. If it is set, APAME 42106's subject has Execute Access to APAME 42106's object.
- Field 42111 is the Read Bit. If it is set, APAME 42106's subject has Read Access to APAME 42106's object.
- Field 42113 is the Write Bit. If it is set, APAME 42106's subject has Write Access to APAME 42106's object.

Any combination of bits in Fields 42109 through 42113 may be set. If all of these fields are set to 0, APAME 42106 indicates that the subject it represents has no access to the object it represents.

KOS sets APAME 42106 for an ASN and an AON 41304 the first time the subject represented by the ASN references the object represented by AON 41304. Until APAME 42106 is set, Valid Bit 42107 is set to 0. When APAME 42106 is set, Valid Bit 42107 is set to 1 and Fields 42109 through 42113 are set according to the primitive data access information in the object's PACLE 41613 whose Subject Template 41601 matches the subject. When an object is deactivated, Valid Bits 42107 in all APAMEs 42106 in the row belonging to the object's AON 41304 are set to 0; similarly, when a subject is deactivated, Valid Bits 42107 in all APAMEs 42106 in the column belonging to the subject's ASN are set to 0.

4. Protection Cache 10234 and Protection Checking (Fig. 422)

The final stage in the acceleration of protection information is Protection Cache 10234 in JP 10114. The details of the way in which Protection Cache 10234 functions are presented in the discussion of the hardware; here, there are discussed the manner in which Protection Cache 10234 performs access checks, the relationship between protection Cache 10234, APAM 10918, and AOT 10712, and the manner in which KOS protection cache microcode maintains Protection Cache 10234.

Figure 422 is a block diagram of Protection Cache 10234, AOTE 10712, APAM 10918, and KOS Microcode 42207 which maintains Protection Cache 10234. Each time JP 10114 makes a memory reference using a Logical Descriptor 27116, it simultaneously presents Logical Descriptor 27116 and a Signal 42208 indicating the kind of memory operation to Protection Cache 10234 and ATU 10228. Entries 42215 in Protection Cache 10234 contain primitive data access and length information for objects previously referenced by the current subject of Process 610 whose Virtual Processor 612 is currently bound to JP 10114. On every memory reference, Protection Cache 10234 emits a Valid/invalid Signal 42205 to MEM 10112. If Protection Cache 10234 contains no Entry 42215 for AON 41304 contained in Logical Descriptor 27116's AON field 27111, if Entry 42215 indicates that the subject does not have the type of access required by process 610, or if the sum of Logical Descriptor 27116's OFF field 27113 and LEN field 27115 exceed the object's current size, Protection Cache 10234 emits an Invalid Signal 42205. This signal causes MEM 10112 to abort the memory reference. Otherwise, Protection Cache 10234 emits a Valid Signal 42205 and MEM 10112 executes the memory reference.

When Protection Cache 10234 emits an Invalid Signal 42205, it latches Logical Descriptor 27116 used to make the reference into Descriptor Trap 20256, the memory command into Command Trap 27018, and if it was a write operation, the data into Data Trap 20258, and at the same time emits one of two Event Signals to KOS microcode. Illegal Access Event Signal 42208 occurs when Process 610 making the reference does not have the proper access rights or the data referenced extends beyond the end of the object. Illegal Access Event Signal 42208 invokes KOS microcode 42215 which performs a Microcode to Software Call 42217 (described in the discussion of Calls) to KOS Access Control System Procedures 602 and passes the contents of Descriptor Trap 20256, Command Trap 27018, the ASN of Process 610 (contained in a register MGR's 10360), and if necessary, Data Trap 20258 to these Procedures 602. These procedures 602 inform EOS of the protection violation, and EOS can then remedy it.

Cache Miss Event Signal 42206 occurs when there is no Entry 42215 for AON 41304 in protection Cache 10234. Cache Miss Event Signal 42206 invokes KOS Protection Cache Miss Microcode 42207, which constructs missing Protection Cache Entry 42215 from information obtained from AOT 10712 and APAM 10918. If APAM 10918 contains no entry for the current subject's ASN and the AON of the object being referenced, protection Cache Miss Microcode 42207 performs a Microcode-to-software Call to KOS Access Control System Procedures 602 which go to LAUDE 40906 for the object and copy the required primitive data access information from the PACLE 41613 belonging to the object whose Subject Template 41601 matches the subject attempting the reference into APAM 10918. The KOS Access Control System Procedures 602 then return to Cache Miss Microcode 42207, which itself returns. Since Cache Miss Microcode 41107 was invoked by an Event Signal, the return causes JP 10114 to reexecute the memory reference which caused the protection cache miss. If protection Cache 10234 was loaded as a result of the

Petitioner Apple Inc. - Exhibit 1004, p. 5325

last protection cache miss, the miss does not recur; if Protection Cache 10234 was not loaded because the required information was not in APAM 10918, the miss recurs, but since the information was placed in APAM 10918 as a result of the previous miss, Cache Miss Microcode 42207 can now construct an Entry 42215 in Protection Cache 10234. When Cache Miss Microcode 42207 returns, the memory reference is again attempted, but this time Protection Cache 10234 contains the information and the miss does not recur.

Cache Miss Microcode 42207 creates a new Protection Cache Entry 42215 and loads it into Protection Cache 10234 as follows: Using AON 41304 from Logical Descriptor 27116 latched into Descriptor Trap 20256 when the memory reference which caused the miss was executed and the current subject's ASN, contained in GR's 10360, Cache Miss Microcode locates APAME 42106 for the subject represented by the ASN and the object represented by AON 41304 and copies the contents of APAME 42106 into a JP 10114 register which may serve as a source for JPD Bus 10142. It also uses AON 41304 to locate AOTE 41306 for the object and copies the contents of Size Field 41519 into another JP 10114 register which is a source for JPD Bus 10142. It then uses three special microcommands, executed in successive microinstructions, to load Protection Cache Entry 42215. The first microcommand loads Protection Cache Entry 42215's TS 24010 with AON 41304 of Logical Descriptor 27116 latched into Descriptor Trap 20256; the second loads the object's size into Protection Cache 10234's EXTENT field, and the third loads the contents of APAME 42106 in the same fashion.

Another microcommand invalidates all Entries 42215 in Protection Cache 10234. This operation, called flushing, is performed when an object is deactivated or when the current subject changes. The current subject changes whenever a Virtual Processor 612 is unbound from JP 10114, and whenever a Process 610 performs a call to or a return from a Procedure 602 executing in a domain different from that in which the calling Procedure 602 or the Procedure 602 being returned to executes in. In the cases of the Call and the unbinding of Virtual Processor 612, the cache flush is performed by KOS Call and dispatching microcode; in the case of object deactivation, it is performed by a KOS procedure using a special KOS SIN which invokes Cache Flush Microcode.

D. Processes

1. Synchronization of Processes 610 and Virtual Processors 612

Since Processes 610 and the Virtual Processors 612 to which they are bound may execute concurrently on CS 10110, KOS must provide means for synchronizing Processes 610 which depend on each other. For example, if process 610 A cannot proceed until Process 610 B has performed some operation, there must be a mechanism for suspending A's execution until B is finished. Generally speaking, four kinds of synchronization are necessary:

- One Process 610 must be able to halt and wait for another Process 610 to finish a task before it proceeds.
- One Process 610 must be able to send another Process 610 a message and wait for a reply before it proceeds.
- When processes 610 share a data base, one Process 610 must be able to exclude other Processes 610 from the data base until the first Process 610 is finised using the data base.
- One Process 610 must be able to interrupt another Process 610, i.e., asynchronously cause the second Process 610 to perform some action.

KOS has internal mechanisms for each kind of synchronization, and in addition supplies synchronization mechanisms to EOS. KOS uses the internal mechanisms to synchronize Virtual Processors 612 and KOS Processes 610, while EOS uses the mechanisms supplied by KOS to synchronize all other Processes 610. The internal mechanisms are the following:

- Event counters, Await Entries, and Await Tables. As will be explained in detail below, Event Counters and Await Entries allow one Process 610 to halt and wait for another Process 610 to complete an operation. Event counters and Await Entries are also used to implement process interrupts. Await Entries are organized into Await Tables.
- Message Queues. Message Queues allow one Process 610 to send a message to another and wait for a reply. Message Queues are implemented with Event Counters and queue data structures.
- Locks. Locks allow one Process 610 to exclude other Processes 610 from a data base or a segment of code. Locks are implemented with Event Counters and devices called Sequencers.

KOS makes Event Counters, Await Entries, and Message Queues available to EOS. It does not provide Locks, but it does provide Sequencers, so that EOS can construct its own Locks. The following discussion will define and explain the logical properties of Event Counters, Await Entries, Message Queues, Sequencers, and Locks. Their implementation in the present embodiment will be described along with the implementation of Processes 610 and Virtual Processors 612.

a. Event Counters 44801, Await Entries 44804, and Await Tables (Fig. 448, 449)

Event Counters, Await Entries, and Await Tables are the fundamental components of the KOS Synchronization System. Figure 448 illustrates Event Counters and Await Entries in the present embodiment. Figure 449 gives a simplified representation of Process Event Table 44705, the present embodiment's Await Tables. Turning first to Figure 448, Event Counter 44801 is an area of memory which

Petitioner Apple Inc. - Exhibit 1004, p. 5326

contains a value that may only be increased. In one of the present embodiment, Event Counters 44801 for KOS systems which may not page fault are always present in MEM 10112; other Event Counters 44801 are stored in Secondary Storage 10124 unless a Process 610 has referenced them and thereby caused the VMM System to load them into MEM 10112. The value contained in an Event Counter 44801 is termed an Event Counter Value 44802. In the present embodiment, EventCounter 44801 contains 64 bits of data, of which 60 make up Event Counter Value 44802. Event Counter 44801 may be referred to either as a variable or by means of a 128-bit UID pointer which contains Event Counter 44801's location. The UID pointer is termed an Event Counter Name 44803.

Await Entry 44804 is a component of entries in Await Tables. In the present embodiment, there are two Await Tables: Process Event Table 44705 and Virtual Processor Await Table (VPAT) 45401. VPAT 45401 is always present in MEM 10112. As already mentioned, Figure 449 illustrates PET 44705. Both PET 44705 and UPAT 45401 will be described in detail later. Each Await Entry 44804 contains an Event Counter Name 44803, an Event Counter Value 44802, and a Back Link 44805 which identifies a Process 610 or a Virtual Processor 612. Await Entry 44804 thus establishes a relationship between an Event Counter 44801, an Event Counter Value 44802, and a Process 610 or Virtual processor 612.

Turning now to Figure 449, in the present embodiment, all Await Entries 44804 for user Processes 610 are contained in PET 44705. PET 44705 also contains other information. Figure 449 presents only those parts of PET 44705 which illustrate Await Entries 44804. PET 44705 is structured to allow rapid location of Await Entries 44804 belonging to a specific Event Counter 44801. PET entries (PETEs) 44909 contain links which allow them to be combined into lists in PETE 44705. There are four kinds of lists in PET 44705:

- Event counter lists: these lists link all PETEs 44909 for Event Counters 44801 whose Event Counter Names 44803 hash to a single value.
- Await lists: These lists link all PETEs 44909 for Event Counters 44801 which a given Process 610 is awaiting.
- Interrupt lists: These lists link all PETEs 44909 for Event Counters 44801 which will cause an interrupt to occur for a given Process 610.
- The Free list: PETEs 44909 which are not being used in one of the above lists are on a free list.

Each PETE 44909 which is on an await list or an interrupt List is also on an event counter list.

Turning first to the event counter lists, all PETEs 44909 on a given event counter list contain Event Counter Names 44803 which hash to a single value. The value is produced by Hash Function 44901, and then used as an index in PET Hash Table (PETHT) 44903. That entry in PETHT 44903 contains the index in PET 44705 of that PETE 44909 which is the head of the event counter list. PETE List 44904 represents one such event counter list. Thus, given an Event Counter Name 44803, KOS can quickly find all Await Entries 44804 belonging to Event Counter 44801.

In the present embodiment, the implementation of Event Counters 44801 and tables with Await Entries 44804 involves both Processes 610 and Virtual Processors 612 to which Processes 610 are bound. As will be explained later, a large number of Event Counters 44801 and Await Entries 44804 belonging to Processes 610 are multiplexed onto a small number of Event Counters 44801 and Await Entries 44804 belonging to the Processes' Virtual Processors 612. Await entries 44804 for Event Counters 44801 belonging to Virtual Processors 612 are contained in VPAT 45401.

b. Synchronization with Event Counters 44801 and Await Entries 44804

The simplest form of Process 610 synchronization provided by KOS uses only Event Counters 44801 and Await Entries 44804. Coordination takes place like this: A Process 610 A requests KOS to perform an Await Operation, i.e., to establish one or more Await Entries 44804 and to suspend Process 610 A until one of the Await Entries is satisfied. In requesting the Await Operation, Process 610 A defines what Event Counters 44801 it is awaiting and what Event Counter Values 44802 these Event Counters 44801 must have for their Await Entries 44804 to be satisfied. After KOS establishes Await Entries 44804, it suspends Process 610 A. While process 610 A is suspended, other Processes 610 request KOS to perform Advance Operations on the Event Counters 44801 specified in Process 610 A's Await Entries 44804. Each time a Process 610 requests an Advance Operation on an Event Counter 44801, KOS increments Event Counter 44801 and checks Event Counter 44801's Await Entries 44804. Eventually, one Event Counter 44801 satisfies one of Process 610 A's Await Entries 44804, i.e., reaches a value equal to or greater than the Event Counter Value 44802 specified in its Await Entry 44804 for process 610 A. At this point, KOS allows process 610 A to resume execution. As process 610 A resumes execution, it deletes all of its Await Entries 44804.

E. Virtual Processors 612 (fig. 453)

As previously stated, a Virtual processor 612 may be logically defined as the means by which a Process 610 gains access to JP 10114. In physical terms, a Virtual Processor is an area of MEM 10112 which contains the information that the KOS microcode which binds Virtual Processors 612 to JP 10114 and unbinds them from JP 10114 requires to perform the binding and unbinding operations. Figure 453 shows a Virtual Processor 612. The area of MEM 10112 belonging to a Virtual Processor 612 is Virtual processor 612's Virtual Processor State Block (VPSB) 614. Each Virtual Processor 612 in a CS 10110 has a VPSB 614. Together, the VPSBs 614 make up VPSB Array 45301. Within the Virtual Processor management system, each Virtual Processor 612 is known by its VP Number 45304, which is the index of the Virtual Processor

Petitioner Apple Inc. - Exhibit 1004, p. 5327

612's VPSB 614 in VPSB Array 45301. Virtual Processors 612 are managed by means of lists contained in Micro VP Lists (MVPL) 45309. Each Virtual processor 612 has an Entry (MVPLE) 45321 in MVPL 45309, and as Virtual Processor 612 changes state, virtual processor management microcode moves it from one list to another in MVPL 45309.

VPSB 614 contains two kinds of information:

information from Process Object 901 belonging to Process 610 which is bound to VPSB 614's Virtual Processor 612, and information used by the Virtual Processor Management System to manage Virtual Processor 612. The most important information from Process Object 901 is the following:

— Process 610's principal and process UIDs 40401.
— AONs 41304 for Process 610's Stack Objects 44703. (VPSB 614 uses AONs 41304 because KOS guarantees that AONs 41304 belonging to Stack Objects 44703 will not change as long as a Process 610 is bound to a Virtual Processor 612.)

Given AON 41304 of Process 610's SS object 10336, the Virtual Processor Management System can locate that portion of Process 610's state which is moved into registers belonging to JP 10114 when process 610's Virtual Processor 612 is bound to JP 10114. Similarly, when Virtual Processor 612 is unbound from JP 10114, the virtual processor management system can move the contents of JP 10114 registers into the proper location in SS Object 10336.

a. Virtual Processor Managment (Fig. 453)

EOS can perform six operations on Virtual Processors 612:

— Request VP allows EOS to request a Virtual Processor 612 from KOS.
— Release VP allows EOS to return a Virtual Processor 612 to KOS.
— Bind binds a Process 610 to a Virtual Processor 612.
— Unbind unbinds a process 610 from a Virtual Processor 612.
— Run allows KOS to bind Process 610's Virtual Processor 612 to JP 10114.
— Stop prevents KOS from binding process 610's Virtual Processor 612 to JP 10114.

As can be seen from the above list of operations, EOS has no direct influence over the actual binding of a Virtual Processor 612 to JP 10114. This operation is performed by a component of KOS microcode called the Dispatcher. Dispatcher microcode is executed whenever one of four things happens:

— Process 610 whose Virtual Processor 612 is currently bound to JP 10114 executes an Await Operation.
— Process 610 whose Virtual Processor 612 is currently bound to JP 10114 executes an Advance Operation which satisfies an Await Entry 44801 for some other Process 610.
— Either Interval Timer 25410 or Egg Timer 25412 overflows, causing an Event Signal which invokes Dispatcher microcode.
— IOJP Bus 10132 is activated, causing an Event Signal which invokes Dispatcher microcode. IOS 10116 activates IOJP bus 10132 when it loads data into MEM 10112 for JP 10114.

When Dispatcher microcode is invoked by one of these events, it examines lists in MVPL 45309 to determine which Virtual Processor 612 is to run next. For the purposes of the present discussion, only two lists are important: the running list and the eligible list. In the present embodiment, the running list, headed by Running List Head 45321, contains only a single MVPLE 45321, that representing Virtual Processor 612 currently bound to JP 10114. In embodiments with multiple JPs 10114, the running list may have more than one MVPLE 45321. The eligible list, headed by Eligible List Head 45313, contains MVPLEs 45321 representing those Virtual Processors 612 which may be bound to JP 10114. MVPLEs 45321 on the eligible list are ordered by priorities assigned Processes 610 by EOS. Whenever KOS Dispatcher microcode is invoked, it compares the priority of Process 610 whose Virtual Processor 612's MVPLE 45321 is on the running list with the priority of Process 610 whose Virtual Processor 612's MVPLE 45321 is at the head of the eligible list. If the latter Process 610 has a higher priority, KOS Dispatcher microcode places MVPLE 45321 belonging to the former Process 610's Virtual Processor 612 on the eligible list and MVPLE 45321 belonging to the latter Process 610's Virtual Processor 612 onto the running list. Dispatcher microcode then swaps Processes 610 by moving state in JP 10114 belonging to the former Process 610 onto the former Process 610's SS object 10336 and moving JP 10114 state belonging to the latter Process 610 from the latter Process 610's SS object 10336 into JP 10114.

b. Virtual Processors 612 and Synchronization (Fig. 454)

When a synchronization operation is performed on a Process 610, one of the consequences of the operation is a synchronization operation on a Virtual Processor 612. For example, an Advance Operation which satisfies an Await Entry 44804 for a Process 610 causes an Advance Operation which satisfies a second Await Entry 44804 for Process 610's Virtual Processor 612. Similarly, a synchronization operation performed on a Virtual Processor 612 may have a synchronization operation on Virtual Processor 612's Process 610 as a consequence. For example, if a Virtual Processor 612 performs an operation involving file I/O, Virtual Processor 612's Process 610 must await the completion of the I/O operation.

Figure 454 illustrates the means by which process level synchronization operations result in virtual processor-level synchronization operations and vice-versa. The discussion first describes the components which transmit process-level synchronization operations to Virtual Processors 612 and the manner in which these components operate. Then it describes the components which transmit virtual processor-level

Petitioner Apple Inc. - Exhibit 1004, p. 5328

synchronization operations to Processes 610 and the operation of these components.

The first set of components is made up of VPSBA 45301 and VPAT 45401. VPSBA 45301 is shown here with two VPSBs 614: one belonging to a Virtual Processor 612 bound to a user Process 610 and one belonging to a Virtual Processor 612 bound to the KOS Process Manager process 610. VPAT 45401 is a virtual processor-level table of Await Entries 44804. Each Await Entry 44804 is contained in a VPAT Entry (VPATE) 45403. Each Virtual Processor 612 bound to a Process 610 has a VPAT Chunk 45402 of four VPATEs 45403 in VPAT 45401, and can thus await up to four Event Counters 44801 at any given time. The location of a Virtual processor 612's VPAT Chunk 45402 is kept in Virtual Processor 612's VPSB 614. When an Advance Operation satisfies any of the Await Entries 44804 belonging to a Virtual Processor 612, all in Virtual Processor 612's VAT Chunk 45402's Await Entries 44804 are deleted. As in PET 44705, VPATES 45403 containing Await Entries 44804 which are awaiting a given Event Counter 44801 are linked together in a list.

VPATEs 45403 for Virtual Processors 612 bound to user Processes 610 may contain Await Entries 44804 for user Process 610's Private Event Counter 45405. Private Event Counter 45405 is contained in Process 610's Process Object 901. It is advanced each time an Await Entry 44804 in a PETE 44909 on a PET List belonging to Process 610 is satisfied.

The components operate as follows: When KOS performs an Await Operation on Process 610, it makes Await Entries 44804 in both PET 44705 and VPAT 45401 and puts Process 610's VP 612 on the suspended list in MVPL 45309. As previously described, an Await Entry 44804 in PET 44705 awaits an Event Counter 44801 specified in the Await Operation which created Await Entry 44804. Await Entry 44804 in VPAT 45401 awaits Process 610's Private Event Counter 45405. Each time an Await Entry 44804 belonging to Process 610 in PET 44705 is satisfied, Process 610's Private Event Counter 45405 is advanced. The advance of Private Event Counter 45405 satisfies Await Entry 44801 for Process 610's Virtual processor 612 in VPAT 45401, and consequently, KOS deletes Virtual Processor 612's VPATEs 45403 and moves Virtual Processor 612's MVPLE 45321 in MVPL 45309 from the suspended list to the eligible list.

The components which allow a Virtual Processor 612 to transmit a synchronization operation to a process 610 are the following: Outward Signals Object (OSO) 45409, Multiplexed Outward Signals Event Counter 45407, and PET 44705. OSO 45409 contains Event Counters 44801 which KOS FU 10120 microcode advances when it perfoms operations which user Processes 610 are awaiting. Event Counters 44801 in OSO 45409 are awaited by Await Entries 44804 in PET 44705. Each time KOS FU 10120 microcode advances an Event Counter 44801 in OSO 45409, it also advances Multiplexed Outward Signals Event Counter 45407. It is awaited by an Await Entry 44804 in VPAT 45401 belonging to Virtual Processor 612 bound to KOS Process Manager Process 610. When Virtual Processor 62 bound to KOS Process Manager Process 610 is again bound to JP 10114, KOS Process Manager Process 610 examines all PETEs 44909 belonging to the Event Counters 44801 in OSO 45423. If an advance of an Event Counter 44801 in OSO 44801 satisfied a PETE 44909 Process 610, that Process 610's Private Event Counter 45405 is advanced as previously described, and Process 610 may again execute.

A user I/O operation illustrates how the components work together. Each user I/O channel has an Event Counter 44801 in OSO 45409. When a Process 610 performs a user I/O operation on a channel, the EOS I/O routine establish an Await Entry 44804 in the PET 44705 list belonging to Process 610 for the channel's Event Counter 44801 in OSO 45409. When the I/O operation is complete, IOS 10116 places a message to JP 10114 in an area of MEM 10112 and activates IOJP Bus 10132. The activation of IOJP Bus 10132 causes an Event Signal which invokes KOS microcode. The microcode examines the message from IOS 10116 to determine which channel is involved, and then advances Event Counter 44801 for that channel in OSO 45409 and Multiplexed Outward Signals Event Counter 45407. The latter advance satisfies an Await Entry 44804 for Process Manager Process 610's Virtual Processor 612 in VPAT 45401, and Process Manager Process 610 begins executing. Process Manager Process 610 examines OSO 45409 to determine which Event Counters 44801 in OSO 45409 have been advanced since the last time process manager Process 610 executed, and when it finds such an Event Counter 44801, it examines the Event Counter Chain in PET 44705 for that Event Counter 44801. If it finds that the advance satisfied any Await Entries 44804 in the Event Counter Chain, it advances Private Event Counter 45405 belonging to Process 610 specified in Await Entry 44804, thereby causing that Process 610 to resume execution as previously described.

F. Process 610 Stack Manipulation

This section of the specification for CS 10110 describes the manner in which Process 610's MAS 502 and SS 504 are manipulated. As previously mentioned, in CS 10110, a Process 610's MAS 502 and SS 504 are contained in several objects. In the present embodiment, there are five objects, one for each domain's portion of the Macro Stack (MAS) (MAS Objects 10328 through 10324) and one for the Secure Stack (SS) (SS Object 10336). In other embodiments, a Process 610's MAS 502 may contain objects for user-defined domains as well. Though a Process 610's MAS 502 and SS 504 are contained in many objects, they function as a single logical stack. The division into several objects is a consequence of two things: the domain component of the protection system, which requires that an object referenced by a Procedure 602 have Procedure 602's domain of execution, and the need for a location inaccessible to user programs for micromachine state and state which may be manipulated only by KOS.

Stack manipulation takes place under the following circumstances:

— When a procedure 602 is invoked or a Return SIN is executed. Procedure 602 invocations are

Petitioner Apple Inc. - Exhibit 1004, p. 5329

performed by means of a Call SIN. Call causes a transfer of control to the first SIN in the invoked Procedure 602 and the Return SIN causes a transfer of control back to the SIN in the invoking Procedure 602 which follows the Call SIN.

— When a non-local Go To SIN is executed. The non-local Go To causes a transfer of control to an arbitrary position in some Procedure 602 which was previously invoked by Process 610 and whose invocation has not yet ended.
— When a condition arises, i.e., an execution of a statement in a program puts the executive Process 610 into a state which requires the execution of a previously established Handler Procedure 602.
— When a Process 610 is interrupted, i.e., when an Interrupt Entry 45718 for Process 610 is satisfied.

Most of the mechanisms involved in stack manipulation are used in Call and Return; these operations are therefore dealt with in detail and the other operations only as they differ from Call and Return. The discussion first introduces Call and Return, then explains the stacks in detail, and finally analyzes Call and Return and the other operations in detail.

1. Introduction to Call and Return

As a Process 610 executes a program, it executes Call and Return SINs. A Call SIN begins an invocation of a procedure 602, and a Return SIN ends the invocation. Generally speaking, a Call SIN does the following:

— It saves the state of Process 610's execution of Procedure 602 which contains the Call SIN. Included in this state is the information required to continue Procedure 602's execution after the Call SIN is finished. This portion of the state is termed calling Procedure 602's Macrostate.
— It creates the state which Process 610 requires to begin execution called Procedure 602.
— It transfers control to the first SIN in the called Procedure 602's code.

The Return SIN does the opposite: it releases the state of called Procedure 602, restores the saved state of calling Procedure 602, and transfers control to the SIN in the calling Procedure 602 following the Call SIN. An invocation of a Procedure 602 lasts from the execution of the Call SIN which transfers control to the Procedure 602 to the execution of the Return SIN which transfers control back to Procedure 602 which contained the Call SIN. The state belonging to a given invocation of a Procedure 602 by a Process 610 is called Procedure 602's invocation state.

While Calls and Returns may be implemented in many different fashions, it is advantageous to implement them using stacks. When a Call creates invocation state for a Procedure 602, that invocation state is added to the top of Process 610's stack. The area of a stack which contains the invocation state of a Procedure 602 is called a frame. Since a called Procedure 602 may call another procedure 602, and that another, a stack may have any number of frames, each frame containing the invocation state resulting from the invocation of a Procedure 602 by Process 610, and each frame lasting as long as the invocation it represents. When called Procedure 602 returns to its caller, the frame upon which it executes is released and the caller resumes execution on its frame. Procedure 602 being currently executed by a Process 610 thus always runs on the top frame of Process 610's MAS 502.

Calls and Returns in CS 10110 behave logically like those in other computer systems using stacks to preserve process 610 state. When a Process 610 executes a Call SIN, the SIN saves as Macrostate the current values of the ABPs, the location of the SIN at which the execution of calling Procedure 602 is to continue, and information such as a pointer to calling Procedure 602's Name Table 10350 and UID 40401 belonging to the S-interpreter object which contains the S-interpreter for Procedure 602's S-language. The Call SIN then creates a stack frame for called Procedure 602, obtains the proper ABP values, the location of called Procedure 602's Name Table 10350 and UID 40401 belonging to its S-interpreter object, and begins executing newly-invoked Procedure 602 on the newly-created stack frame. The Return SIN deletes the stack frame obtains the ABP values and name interpreter information from the Macrostate saved during the Call SIN and then transfers control to the SIN at which execution of calling Procedure 602 is to continue.

However the manner in which Call and Return are implemented is deeply affected by CS 10110's Access Control System Broadly speaking there are two classes of Calls and Returns in CS 0110: those which are mediated by KOS and those which are not. In the following discussion, the former class of Calls and Returns are termed Mediated Calls and Returns, and the latter are called Neighborhood Calls and Returns. Most Calls and Returns executed by CS 10110 are Neighborhood Calls and Returns; Mediated Calls and Returns are typically executed when a user procedure 602 calls EOS Procedures 602 and these in turn call KOS Procedures 602. The Mediated Call makes CS 10110 facilities available to user Processes 610 while protecting these CS 10110 facilities from misuse and therefore generally serves the same purpose as system calls in the present art. As will be seen in the ensuing discussion, Mediated Call requires more CS 10110 overhead than Neighborhood Call but the extra overhead is less than that generally required by system calls in the present art.

Mediated Calls and Returns involve S-interpreter, Namespace, and KOS microcode. S-interpreter and Namespace microcode interpret the Names involved in the call and only modifies those portions of Macrostate accessible to the S-interpreter. The remaining Macrostate is modified by KOS microroutines invoked in the course of the Call SIN. A Mediated Call may be made to any Procedure 602 contained in an object to which Process 610's subject has Execute Access at the time the invocation occurs. Mediated Calls and Returns must be made in the following situations:

Petitioner Apple Inc. - Exhibit 1004, p. 5330

— When called Procedure 602 has a different Procedure Environment Descriptor (PED) 30303 from that used by calling Procedure 602. Such Calls are termed Cross-PED Calls.
— When called Procedure 602 is in a different Procedure Object 608 from calling Procedure 602. Such Calls are termed Cross-Procedure Object Calls.
— When called Procedure 602's Procedure Object 608 has a different Domain of Execution (DOE) Attribute from that of calling Procedure 602's Procedure Object 608, and therefore must place its Invocation State on a different MAS object from that used by calling Procedure 602. Such Calls are termed Cross-Domain Calls.

In all of the above Calls, the information required to complete the Call is not available to the S-interpreter and consequently. KOS mediation is required to complete the Call. Neighborhood Calls and Returns only modify two components of Macrostate: the pointer to the current SIN and the FP ABP. Both of these components are available to the S-interpreter as long as called Procedure 602 has the same PED 30303 i.e., uses the same Name Tabe 10350 and S-interpreter or the calling Procedure 602 and has Names with the same syllable size as calling Procedure 602. The Call and Return SINs are specific to each S-language, but they resemble each other in their general behavior. The following discussion will deal exclusively with this general behavior and will concentrate on Mediated Calls and Returns. The discussion first describes MAS 502 and SS 504 belonging to a Process 610 and those parts of Procedure Object 608 involved in Calls and Returns, and then describes the Implementation of Calls and Returns.

2. Macro Stacks (MAS) 502 (Fig. 467)

Figure 467 gives an overview of an object belonging to a Process 610's MAS 502. The description of this Figure will be followed by descriptions of other Figures containing detailed representations of portions of MAS objects.

At a minimum MAS Object 46703 comprises KOS MAS Header 10410 together with Unused Storage 46727 reserved for the other elements comprising MAS Object 46703. If Process 610 has not yet returned from an invocation of a Procedure 602 contained in a Procedure Object 608 whose DOE is that required for access to MAS Object 46703. MAS object 46703 further comprises a Stack Base 46703 and at least one MAS Frame 46709.

Each MAS Frame 46709 represents one mediated invocation of a procedure 602 contained in a Procedure Objece 608 with the DOE attribute required by MAS 46703, and may in addition represent neighborhood invocations of Procedures 602 which share that Procedure 602's Procedure Object 608. The topmost MAS Frame 46709 represents the most recent group of invocations of Procedures 602 with the DOE attribute required by MAS Object 46703 and the bottom MAS Frame 46709 the earliest group of invocations from which Process 610 has not yet returned. Frames for invocations of Procedures 602 with other domains of execution are contained in other MAS Objects 46703. As will be explained in detail below MAS Frames 46709 in different MAS objects 46703 are linked by pointers.

MAS Domain Stack Base 46703 has two main parts: KOS MAS Header 10410 which contains information used by KOS microcode which manipulates MAS Object 46703, and Perdomain Information 46707, which contains information about 46703's domain and static information, i.e., information which lasts longer than an invocation used by Procedures 602 with MAS Frames 46709 on MAS Object 46703. MAS Frame 46709 also has two main parts, a KOS Frame Header 10414 which contains information used by KOS to manipulate Frame 46709 and S-interpreter Portion 46713 which contains information available to the S-interpreter when it executes the group of Procedures 602 whose invocations are represented by Frame 46709.

When making Calls and Returns, the S-interpreter and KOS microcode use a group of pointers to locations in MAS Object 46703. These pointers comprise the following:
— MAS Object UID 46715 the UID 40401 of AS Object 46703.
— First Frame Offset (FFO) 46719 which locates the beginning of KOS Frame Header 10414 belonging to the first MAS Frame 46709 in MAS Object 46703.
— Frame Header Pointer (FHP) 46702 which locates the beginning of the topmost KOS Frame Header 10414 in MAS Object 46703.
— Stack Top Offset (STO) 46704 a 32-bit offset from Stack UID 46715 which marks the first bit in Unused Storage 46727.

As will be seen presently all of these pointers are contained in fields in KOS MAS Header 46705.

a.a. MAS Base 10410 (Fig. 468)

Figure 468 is a detailed representation of MAS Domain Stack Base 10410 Turning first to the detailed representation of KOS MAS Header 46705 contained therein, there are the following fields:
— Format Information Field 46801 containing information about the format of KOS MAS Header 46705.
— Flags Field 46803. Of these flags, only one is of interest to the present discussion: Domain Active Flag 46804. This flag is set to TRUE when Process 610 to which MAS Object 46703 belongs is executing the Invocation of Procedure 602 whose invocation record makes up the topmost MAS Frame 46709 contained in MAS Object 46703 to which KOS MAS Header 46705 belongs.
— PFO Field 46805: All MAS Headers 46705 and Frame Headers 46709 have fields containing offsets locating the previous and following headers in MAS Object 46703. In a Stack Header 46705 there is no

Petitioner Apple Inc. - Exhibit 1004, p. 5331

previous header and this field is set to 0.

- FFO Field 46805: The field locating the following header In a Stack Header 46705 this field contains FFO 46719 since the next header is the first Frame Header in MAS Object 46703.
- STO Field 46807: the field containing STO offset 46704.
- Process ID Field 46809: UID 40401 belonging to Process Object 901 for Process 610 to which MAS Object 46703 belongs.
- Domain Environment Information pointer Field 46811: The pointer contained in the field locates an area which contains domain-specific information. In the present embodiment. the area is part of MAS Stack Base 10410; however, in other embodiments, it may be contained in a separate object.
- Signaller Pointer Field 46813: The pointer contained in the field locates a Procedure 602 which KOS invokes when a Process 610's execution causes a condition to arise while it is executing in the domain to which MAS object 46703 belongs.
- AAT Pointer Field 30211: The pointer in Field 30211 locates AAT 30201 for MAS Object 46703. AAT 30201 is described in detail in Chapter 3.
- Frame Label Sequencer Field 46819: This field contains a Sequencer 45102. Sequencer 45102 is used to generate labels used to locate MAS Frames 46709 when a non-local GOTO is executed.

Turning now to the detailed representation of Domain Environment Information 46821 located by Domain Environment Information Pointer Field 46811 there are the following fields:

- KOS Format Information Field 46823.
- Flags Field 46825 containing the following flags:
  - Pending Interrupt Flag 46827, set to TRUE when Process 610 has an interrupt pending for the domain to which MAS Object 46703 belongs.
  - Domain Dead Flag 46829, set to TRUE when Process 610 can no longer execute Procedures 602 with domains of execution equal to that to which MAS Object 46703 belongs.
  - Invoke Verify on Entry Flag 46833 and Invoke Verify on Exit Flag 46835. The former flag is set to TRUE when KOS is to invoke a Procedure 602 which checks the domain's data bases before a Procedure 602 is allowed to execute on the domain's MAS Object 46703; the latter is set to TRUE when KOS is to invoke such a Procedure 602 on exit from a Procedure 602 with the domain as its DOE.
  - Default Handler Non-null Flag 46835 is set to TRUE when there is a default clean-up handler for the domain. Clean-up handlers are described later.
  - Interrupt Mask Field 46839 determines what interrupts set for Process 610 in MAS object 46703's domain will be honored.
  - Domain UID Field 46841 contains UID 40401 for the domain to which MAS Object 46703 belongs.
  - Fields 46843 through 46849 are pointers to Procedures 602 or tables of pointers to Procedures 602. The Procedures 602 so located handle situations which arise as MASs 502 are manipulated. The use of these fields will become clear as the operations which require their use are explained.

b.b. Per-domain-Data Area 46853 (Fig. 468)

Per-domain Data Area 46853 contains data which cannot be kept in MAS Frames 46709 belonging to invocations of Procedures 602 executing in MAS Object 46703's domain, but which must be available to these invocations Per-Domain Data Area 46853 has two components: Storage Area 46854 and AAT 30201. Storage Area 46854 contains static data used by Procedures 602 with invocations on MAS Object 46703 and data used by S-interpreters which are used by such procedures 602. Associated Address Table (AAT) 30201 is used to locate data in Storage Area 46854. A detailed discussion of AAT 30201 is contained in Chapter 3.

Two kinds of data is stored in Storage Area 46854: static data and S-interpreter data.

Static data is stored in Static Data Block 46863. Static Data Block 46863 comprises two parts: Linkage Pointers 46865 and Static Data Storage 46867 Linkage Pointers 46865 are pointers to static data not contained in Static Data Storage 46867 for example, data which lasts longer than Process 610 and pointers to External Procedures 602 which the Procedure 602 to which Static Data Storage 46867 belongs invokes. Static Data Storage 46867 contains storage for static data used by the Procedure 602 which does not last longer than Process 610 executing the Procedure 602.

S-interpreter data is data required by S-interpreters used by Procedures 602 executing on MAS object 46703.

The S-interpreter data is stored in S-interpreter Environment Block (SEB) 46864 which, like Static Data Block 46864 is located via AAT 30201: The contents of SEB 46864 depend on the S-interpreter.

c.c. MAS Frame 46709 Detail (fig. 469)

Figure 469 represents a typical frame in MAS Object 46703. Each MAS Frame 46709 contains a Mediated Frame 46947 produced by a Mediated Call of a Procedure 602 contained in a Procedure Object 608 whose DOE attribute is the one required for execution on MAS object 46703. Mediated Frame 46947 may be followed by Neighborhood Frames 46945 produced by Neighborhood Calls of Procedures 602. Mediated Frame 46947 has two parts, a KOS Frame Header 10414 which is manipulated by KOS microcode, and an S-interpreter portion which is manipulated by S-interpreter and Namespace microcode. Neighborhood Frames 46945 have no KOS Frame Headers 10414. As will become clear upon closer

Petitioner Apple Inc. - Exhibit 1004, p. 5332

examination of Figure 469. Mediated Frames 46947 in the present embodiment contain no Macrostate. In the present embodiment, Macrostate for these frames is kept on SS Object 10336; however in other embodiments, Macrostate may be stored in Mediated Frames 46947. Neighborhood Frames 46945 contain those portions of the macrostate which may be manipulated by Neighborhood Call; the location of this macrostate depends on the Neighborhood Call SIN.

Turning now to KOS Frame Header 10414. there are the following fields:

- KOS Format Information Field 46901 containing information about MAS Frame 46709's format.
- Flags Field 46902. This field contains the following flags:
  - Result of Cross-domain Call Flag 46903. This Flag is TRUE if MAS Frame 46709 which precedes this MAS Frame 46709 is in another MAS Object 46703.
  - Is Signaller Flag 46905. This flag is TRUE if this MAS Frame 46709 was created by the invocation of a Signaller Procedure 602.
  - Do Not Return Flag 46907: This flag is TRUE if Process 610 is not to return to the invocation for which this MAS Frame 46709 was created.
  - Flags 46909 through 46915 indicate whether various lists used in condition handling and non-local GOTOs are present in the MAS Frame 46709.
- Previous Frame Offset Field 46917. Next Frame Offset Field 46919. and Frame Top Offset Field 46921 are offsets which give the location where Header 10414 for the previous MAS Frame 46709 in MAS Object 46703 begins. the location where the header for the next MAS Frame 46709 in MAS Object 46703 begins, and the location of the first bit beyond the top of MAS Frame 46709 respectively.
- Fields 46923 through 46927 are offsets which locate lists in S-interpreter portion 46713 of Frame 46709. KOS establishes such lists to handle conditions and non-local GOTOs. Their use will be explained in detail under those headings.
- Fields 46929 and 46933 contain information about Procedure 602 whose invocation is represented by MAS Frame 46709. Field 46929 contains the number of arguments required by procedure 602 and Field 46933 contains a resolvable pointer to Procedure 602's PED 30303. Both these fields are used primarily for debugging.
- Dynamic Back Pointer Field 46931 contains a resolvable pointer to the preceding MAS Frame 46709 belonging to Process 610's MAS 502 when that MAS Frame 46709 is contained in a different MAS Object 46703. In this case, Flag Field 46903 is set to TRUE. When the preceding MAS Frame 46709 is contained in the same MAS object 46703 field 46931 contains a pointer with a null UID 40401 and Flag Field 46903 is set to FALSE.
- Frame Label Field 46935 is for a Frame Label produced when a non-local GOTO is established which transfers control to the invocation represented by MAS Frame 46709. The label is generated by Frame Label Sequencer 46819 in KOS MAS Header 10410.

S-interpreter Portion 46713 of MAS Frame 46709 comprises those portions of MAS Frame 46709 which are under control of the S-interpreter. S-interpreter Portion 46713 in turn comprises two main subdivisions: those parts belonging to Mediated Frame 46947 and those belonging to Neighborhood Frames 46945.

The exact form of S-interpreter portion 46949 of KOS Frame 46947 and of S-Interpreter Frames 46945 depends on the Call SIN which created the frame in question. However all Neighborhood Frames 46945 and S-interpreter portions 46949 of Mediated Frames 46947 have the same arrangements for storing Linkage Pointers 10416 and local data in the frame. Linkage Pointers 10416 are pointers to the locations of actual arguments used in the invocation and Local Storage 10420 contains data which exists only during the invocation. In all Mediated Frames 46947 and Neighborhood Frames 46945. Linkage pointers 10416 precede Local Storage 10420. Furthermore, when a Mediated Frame 46947 or a Neighborhood Frame 46945 is the topmost frame of Process 610's MAS. i.e, when Process 610 is executing on that frame, the FP always points to the beginning of Local Storage 10420. and the beginning of Linkage Pointers 10416 is always at a known displacement from FP. References to Linkage Pointers 10416 may therefore be expressed as negative offsets from FP, and references to Local Storage 10420 as positive offsets.

In addition, S-interpreter Portion 46713 may contain lists of information used by KOS to execute non-local GOTOs and conditions, as well as S-interpreter frames for non-mediated calls. The lists of information used by KOS are contained in List Area 46943. The exact location of List Area 46943 is determined by the compiler which generates the SINs and Name Table for the Procedure 602 whose invocation is represented by Mediated Frame 46947. When Procedure 602's source text contains statements requiring storage in List Area 46943, the compiler generates SINs which place the required amount of storage in Local Storage 10420. KOS routines then build lists in Area 46943, and place the offsets of the heads of the lists in Fields 46923, 46925 or 46927, depending on the kind of list. The lists and their uses are described in detail later.

3. SS 504 (Fig. 470)

Figure 470 presents an overview of SS 504 belonging to a Process 610. SS 504 is contained in SS Object 10336. SS Object 10336 is manipulated only by KOS microcode routines. Neither Procedures 602 being executed by Process 610 nor S-interpreter or Namespace microcode may access information contained in SS Object 10336.

SS Object 10336 comprises two main components, SS Base 47001 and SS Frames 47003. Turning first to the general structure of SS Frames 47003, each time a Process 610 executes a Mediated Call KOS

Petitioner Apple Inc. - Exhibit 1004, p. 5333

microcode creates a new SS Frame 47003 on SS Object 10336 belonging to Process 610 and each time a Process 610 executes a Mediated Return, KOS microcode removes the current top SS Frame 47003 from SS Object 10336. There is thus one SS Frame 47003 on SS Object 10336 belonging to a process 610 for each Mediated Frame 46947 on Process 610's MAS 502.

SS Frames 47003 comprise two kinds of frames:

Ordinary Frames 10510 and Cross-domain Frames 47039. Cross-domain Frames 47039 are created whenever Process 610 executes a Cross-domain Call; for all other Mediated Calls. Ordinary Frames 10510 are created. Cross-domain Frames 47039 divide SS Frames 47003 into Groups 47037 of SS Frames 47003 belonging to sequences of invocations in a single domain. The first SS Frame 47003 in a Group 47037 is a Cross-domain Frame 47039 for the invocation which entered the domain, and the remainder of the SS Frames 47003 are Ordinary Frames 10510 for a sequence of invocations in that domain. These groups of SS Frames 47003 correspond to groups of Mediated Frames 46947 in a single MAS Object 46703.

a.a. SS Base 47001 (Fig. 471)

SS Base 47001 comprises four main parts: SS Header 10512 Process Microstate 47017, Storage Area 47033 for JP 10114 register contents, and Initialization Frame Header 47035. Secure Stack Header 10512 contains the following information:

- Fields 47001 and 47009 contain flag and format information; the exact contents of these fields are unimportant to the present discussion.
- Previous Frame Offset Value Field 47011 is a standard field in headers in SS Object 10336; here it is set to 0, since there is no previous frame.
- Secure Stack First Frame Offset Field 47013 contains the offset of the first SS Frame 47039 in SS object 10336, i.e., Initialization Frame Header 47035.
- Process UID field 47015 contains UID 40401 of Process 610 to which SS Object 10336 belongs.
- Number of Cross Domain Frames Field 47016 contains the number of Cross-domain Frames 47039 in SS Object 10336.

Process Microstate 47017 contains information used by KOS microcode when it executes Process 610 to which SS Object 10336 belongs Fields 47019, 47021 and 47022 contain the offsets of locations in SS Object 10336. Field 47019 contains the value of SSTO the location of the first free bit in SS Object 10336; Field 47021 contains the value of SSFO, the location of the topmost frame in SS object 10336; Field 47022 finally contains the value of XDFO, the location of the topmost Cross-domain Frame 47039 in SS Object 10336. All of these locations are marked in Figure 470.

Other fields of interest in Process Microstate 47017 comprise the following: Offsets in Storage Area Field 47023 contains offsets of locations in Storage Area 47033 of SS Object 10336; Domain Number Field 47025 contains the domain number for the DOE of Procedure 602 currently being executed by Process 610. The relationship between domain UIDs and domain numbers is explained in the discussion of domains. VPAT Offset Field 47027 contains the offset in VPAT 45401 of VPAT Chunk 45402 belonging to Virtual Processor 612 to which Process 610 is bound. Signal Pointer Field 47029 contains a resolved pointer to the Signaller (a Procedure 602 used in condition handling) belonging to the domain specified by Domain Number Field 47025 and Trace Information Field 47031 contains a resolved pointer to that domain's Trace Table, described later.

Storage Area for JP 10114 register Contents 47033 is used when a Virtual Processor 612 must be removed from JP 10114. When this occurs, either because Virtual Processor 612 is unbound from JP 10114, because CS 10110 is being halted, or because CS 10110 has failed, the contents of JP 10114 registers which contain information specific to Virtual Processor 612 are copied into Storage Area 47033. When Virtual Processor 612 is returned to JP 10114, these register contents are loaded back into the JP 10114 registers from whence they came. Initialization Frame Header 47035, finally, is a dummy frame header which is used in the creation of SS Object 10336.

b.b. SS Frames 47003 (Fig. 471)

Commencing the discussion of SS Frames 47039 and 10510. Figure 471 illustrates these structures in detail. Ordinary SS Frame 10510 comprises three main divisions: Ordinary SS Frame Header 10514, Macrostate 10516 and Microstate 10520. Ordinary SS Frame Header 10514 contains information used by KOS microcode to manipulate Ordinary SS Frame 10510 to which Header 10514 belongs. Macrostate 10516 contains the values of the ABPs for the frame's mediated invocation and other information required to resume execution of the invocation. Microstate 10520 contains micromachine state from FU 10120 and EU 10122 registers. The amount of micromachine state depends on the circumstances; in the present embodiment, some micromachine state is saved on all Mediated Calls; furthermore, if a Process 610 executes a microcode-to-software Call, the micromachine state that existed at the time of the call is saved; finally. Microstate 10520 belonging to the topmost SS Frame 47003 may contain information which was transferred from FU 10120 GRF registers 10354 or EU 10122 register and stack mechanism 10216 when their capacity was exceeded. For details about this portion of Microstate 10520 see the discussion of the FU 10120 micromachine in Chapter 2. The discussion of SS Object 10336 continues with details concerning SS Header 10514 and Macrostate 05163.

Petitioner Apple Inc. - Exhibit 1004, p. 5334

a.a.a. Ordinary SS Frame Headers 10514 (Fig. 741)

Fields of interest in Ordinary Secure Stack Frame Header 10514 are the following:

- Format Information 47103 which identifies the format of Header 10514.
- Flags Field 47105 which contains one flag of interest in this discussion: Frame Type Flag 47107: in Ordinary SS Frames 10510 this field is set to FALSE.
- Offset Fields 47109 through 47113: Field 47109 contains the offset of the previous SS Frame 47039 or 10510. Field 47111 contains the offset of the following SS Frame 47039 or 10510. and Field 47113 contains the offset of the last SS Frame 47039 or 10510 preceding the next Crossdomain Frame 47039
- Field 47117 contains the current domain number for the domain in which the mediated invocation represent SS Frame 47039 or 10510 is executing.
- Field 47119 contains the offset of the preceding Cross-domain Frame 47039.
- Field 47121 contains offsets for important locations in Microstate 10520.

b.b.b. Detailed Structure of Macrostate 10516 (Fig. 471)

These fields are of interest in Macrostate 10516:

- Syllable Size Field 47125 contains the value of K, i.e., the size of the Names in the SINs belonging to Procedure 602 which the invocation is executing.
- End of Name Table Field 47127 contains the location of the last Name in Name Table 10350 belonging to Procedure 602 which the invocation is executing.
- Fields 47129 through 47143 are resolved pointers to locations in Procedure Object 901 containing Procedure 602 being executed by the invocation and resolved pointers to locations containing data being used by Procedure 602. Field 47129 contains a pointer to Procedure 602's PED 30303; if Procedure 602 is an External Procedure 602. Field 47131 contains a pointer to Procedure 602's entry in Gates 10340; Field 47135 contains the UID-offset value of FP for the invocation; Field 47135 contains a pointer to SEB 46864 used by Procedure 602's S-interpreter. Field 47137 contains the UID-offset value of SDP and Field 47139 contains that of PBP. SIP Field 47141 contains a pointer to Procedure 602's S-interpreter object, and NTP, finally, is a pointer to Procedure 602's Name Table 10350.
- Field 47145 contains the PC for the SIN which is to be executed on return from the mediated invocation to which SS Frame 47003 belongs.

c.c.c. Cross domain SS Frames 47039 (Fig. 471)

Cross-domain SS Frames 47039 differ from Ordinary SS Frames 10510 in two respects: they have an additional component, Cross-domain State 10513, and fields in Cross domain Frame Header 47157 have different meanings from those in Ordinary Frame Header 10514.

Cross-domain State 10513 contains information which KOS Call microcode uses to verify that a return to a Procedure 602 whose DOE differs from that of Procedure 602 whose invocation has ended is returning to the proper domain. Fields of interest in Cross-domain State 10513 include GOTO Tag 47155 used for non-local GOTOs which cross domains, Stack Top Pointer Value 47153, which gives the location of the first free bit in the new domain's MAS Object 46703 and Frame Header Pointer Value 47151, which contains the location of the topmost Mediated Frame Header 46709 in new MAS Object 46703.

There are three fields in Cross-domain Frame Header 47157 which differ from those in Ordinary SS Frame Header 47101. These fields are Flag Field 47107 which in Cross-domain Frame Header 47157 always has the value TRUE, preceding Cross-domain Frame Offset Field 47161, which contains the offset of preceding Cross-domain Frame 47039 in SS Object 10336 and Next Cross domain Frame Offset Field 47159, which contains the location of the next Cross-domain Frame 47039. These last two fields occupy the same locations as Fields 47111 and 47109 respectively in Ordinary SS Frame Header 10514.

As will be noted from the above description of SS Frames 47003. Secure Stack Object 10336 in the present embodiment contains three kinds of information: macrostate cross-domain state and microstate. In other embodiments, the information in SS object 10336 may be stored in separate stack structures, for example, separate microstate and cross-domain stacks, or information presently stored in MAS Objects 46703 may be stored in SS Object 10336, and vice-versa.

4. Portion of Procedure Object 608 Relevant to Call and Return (Fig. 472)

The information which Process 610 requires to construct new frames on its MAS Objects 46703 and SS Object 10336 and to transfer control to invoked Procedure 602 is contained in invoked Procedure 602's Procedure Object 608. Figure 472 is an overview of Procedure Object 608 showing the information used in a Call. Figure 472 expands information contained in Figures 103 and 303; fields that appear in those Figures have the names and numbers used there.

Beginning with Procedure Object Header 10336, this area contains two items of information used in Calls: an offset in Field 47201 giving the location of Argument Information Array 10352 in Procedure Object 608 and a value in Field 47203 specifying the number of gates in Procedure Object 608. Gates allow the invocation of External Procedures 602 that is, Procedures 602 which may be invoked by Procedures 602 contained in other Procedure Objects 608. Procedure Object 608's gates are contained in External Entry Descriptor Area 10340. There are two kinds of gates: those for Procedures 602 contained in Procedure Object 608, and those for procedures 602 contained in other Procedure Objects 608, but callable via

Petitioner Apple Inc. - Exhibit 1004, p. 5335

Procedure Object 608. Gates for Procedures 602 contained in Procedure Object 608 are termed Local Gates 47205. Local Gates 47205 contain Internal Entry Offset (IEO) Field 47207 which contains the offset in Procedure Object 608 of Entry Descriptor 47227 for Procedure 602. If Procedure 602 is not contained in Procedure Object 472 its gate is a Link Gate 47206. Link Gates 47206 contain Binder Area Pointer (BAP) Fields 47208. A BAP Field 47208 contains the locations of an area in Binder Area 30323 which in turn contains a pointer to a Gate in another Procedure Object 608. The pointer in Binder Area 30323 may be either resolved or unresolved. If Procedure 602 is contained in that Procedure Object 608, the Gate is a Local Gate 47205; otherwise, it is another Link Gate 47206.

Procedure Environment Descriptors (PEDS) 10348 contains PEDs 30303 for Procedures 602 contained in Procedure Object 608. Most of the macrostate information for a Procedure 602 may be found in its PED 30303. PED 30303 has already been described, but for ease of understanding, its contents are reviewed here.

- K Field 30305 contains the size of Procedure 602's Names.
- Largest Name (LN) Field 30307 contains the i

Beginning with Procedure Object Header 10336, this area contains two items of information used in Calls: an Offset in Field 47201 giving the location of Argument Information Array 10352 in Procedure Object 608 and a value in Field 47203 specifying the number of gates in Procedure Object 608. Gates allow the invocation of External Procedures 602, that is, Procedures 602 which may be invoked by Procedures 602 contained in other Procedure Objects 6nter to Static Data Block 46863. Thus, for that invocation of Procedure 602 on invocation, the SDP ABP is derived via SDPP field 30313.

- PBP Field 30315 is the pointer from which the current PC is calculated. When Procedure 602 is invoked, this value becomes the PBP ABP.
- S-interpreter Environment Prototype Pointer (SEPP) Field 30316 contains the location of SEB Prototype Field 30317. When Procedure 602 is invoked, Field 30316 locates SEB 46864 via AAT 30201 in the same manner as SDPP field 30313 locates the invocation's static data.

A Procedure 602's PED 30303 may be located from its Internal Entry Descriptor 47227. A PED 30303 may be shared by several Procedures 602. Of course in this case, the values contained in shared PED 30303 are the same for all Procedures 602 sharing it. As will be explained in detail later in the present embodiment. if a calling Procedure 602 does not share a PED 30303 with called Procedure 602 the Call must be mediated. A calling Procedure 602 may make a Neighborhood Call only to Procedures 602 with which it shares a PED 30303.

The next portion of Procedure Object 608 which is of interest is Internal Entry Descriptors 10342. Each Procedure 602 contained in Procedure Object 608 has an Entry Descriptor 47227. Entry Descriptor 47227 contains four fields of interest:

- PBP Offset Field 47229 contains the offset from PBP at which the first SIN in Procedure 602's code is located.
- Flags Field 47230 contains flags which are checked when Procedure 602 is invoked. Four flags are of interest:
  - Argument Information Array Present Flag 47235 which is set to TRUE if Procedure 602 has entries in Argument Information Array 10352.
  - SEB Flag 47237 is set to TRUE if SEPP 47225 is non-null, i.e., if Procedure 602 has a SEB 46864 for its S-interpreter.
  - Do Not Check Access Flag 47239 is set to TRUE if KOS Call microcode is not to perform protection checking on the actual arguments used to invoke Procedure 602.
- PED Offset Field 47231 contains the offset of Procedure 602's PED 30303 from the beginning of Procedure Object 608.
- Frame Size Field 47233 contains the initial size of the Local Storage Portion 10420 of MAS Frame 46709 for an invocation of procedure 602.

Other areas of interest for Calls are SEB Prototype Area 47241, Static Data Area Prototype 30317, Binder Area 30323 and Argument Information Array 10352. SEB Prototype type Area 47241 and Static Data Area Prototype 30315 contain information used to create an SEB 46864 and Static Data Block 46863 respectively for Procedure 602. These areas are created on a per-MAS Object 46703 basis. The first time that a Process 610 executes a Procedure 602 in a domain, SEB 46864 and Static Data Block 46863 required for Procedure 602 are created either in MAS Object 46703 belonging to the domain or in another object accessible from MAS Object 46703. SEB 46864 and Static Data Block 46863 then remain as long as MAS Object 46703 exists.

Static Data Prototype 30317 contains two kinds of information: Static Data Links 30319 and Static Data Initialization Information 30321 Static Data Links 30319 contain locations in Binder Area 30323, which in turn contains pointers which may be resolved to yield the locations of data or External Procedures 602. When a Static Data Block 46863 is created for a Procedure 602, the information in Binder Area 30323 is used to create Linkage Pointers 46865. Static Data Initialization Information 30321 contains information required to create and initialize static data in Static Data Storage 46867.

As mentioned in the discussions of Link Gates 47206 and Static Data Links 30319 Binder Area 30323 contains pointers which may be resolved as described in Chapter 3 to yield locations of data and External Procedures 602.

Petitioner Apple Inc. - Exhibit 1004, p. 5336

Argument Information Array (AIA) 10352 contains information used by KOS Call microcode to check whether the subject which is invoking Procedure 602 has access to the actual arguments used in the invocation which allows the uses made of the arguments in Procedure 602. This so-called "Trojan horse check" is necessary because a Call may change the domain component of a subject. Thus, a subject which is lacking access of a specific kind to a data item could gain that access by passing the data item as an argument to a Procedure 602 whose DOE gives it access rights that the calling subject itself lacks.

Each Local Gate 47205 in Procedure Object 608 has an element in AIA 10352. Each of these Argument Information Array Elements (AIAEs) 60845 has fields indicating the following:

- The minimum number of arguments required to invoke Procedure 602 to which Local Gate 47205 belongs, in Field 47247.
- The maximum number of arguments which may be used to invoke Procedure 602 in Field 47249.
- The access rights that the invoking subject must have to the actual arguments in order to invoke Procedure 602 in Field 47251.

Field 47251 is itself an array which specifies the kinds of access that the invoking subject must have to the actual arguments it uses to invoke Procedure 602. Each formal argument for Procedure 602 has an Access Mode Array Entry (AMAE) 47255. The order of the AMAEs 47255 corresponds to the order of Procedure 602's formal arguments. The first formal argument has the first AMAE 47255, the second the second, and so forth. An AMAE 47253 is four bits long. There are two forms of AMAE 47253: Primitive Access Form 47255 and Extended Access Form 47257. In the former form, the leftmost bit is set to 0. The three remaining bits specify read, write, and execute access. If a bit is on, the subject performing the invocation must have that kind of primitive access to the object containing the data item used as an actual for the formal argument corresponding to that AMAE 47253. In the Extended Access Form 47257, the leftmost bit is set to 1 and the remaining bits are defined to represent extended access required for Procedure 602. The definition of these bits varies from Procedure 602 to Procedure 602.

5. Execution of Mediated Calls

Having described the portions of MAS Object 46703, SS Object 10336. and Procedure Object 608 which are involved in Calls, the discussion turns to the description of the Mediated Call Operation. First, there is presented an overview of the Mediated Call SIN and then the implementation of Mediated Calls in the present embodiment is discussed, beginning with a simple Mediated Call and continuing with Cross-Procedure Object Calls and Cross Domain Calls. The discussion closes with a description of software-to-microcode Calls.

a.a. Mediated Call SINs

While the exact form of a Mediated Call SIN is S-language specific, all Mediated Call SINs must contain four items of information:

- The SOP for the operation.
- A Name that evaluates to a pointer to the Procedure 602 to be invoked by the SIN.
- A literal (constant) specifying the number of actual arguments used in the invocation.
- A list of Names which evaluate to pointers to the actual arguments used in the invocation.

If Procedure 602 requires no arguments, the literal will be 0 and the list of Names representing the actual arguments will be empty.

In the present embodiment, Mediated Call and Return SINs are used whenever called Procedure 602 has a different PED 30303 from calling Procedure 602. In this case, the Call must save and recalculate macrostate other than FP and PC, and mediation by KOS Call microcode is required. The manner in which KOS Call microcode mediates the Call depends on whether the Call is a simple Mediated Call a Cross-procedure Object Call, or a Cross-Domain Call.

b.b. Simple Mediated Calls (Fig. 270, 468, 469, 470, 471, 472)

When the Mediated Call SIN is executed, S-interpreter microcode first evaluates the Name which represents the location of the called Procedure 602. The Name may evaluate to a pointer to a Gate 47205 or 4707 in another Procedure Object 608 or to a pointer to an Entry Descriptor 47227 in the present Procedure Object 608. When the Name has been evaluated, S-interpreter Call microcode invokes KOS Call microcode, using the evaluated Name as an argument. This microcode first fills in Macrostate Fields 10516, left empty until now, in the current invocation's SS Frame 47003. The microcode obtains the values for these fields from registers in FU 10120 where they are maintained while Virtual Processor 612 of Process 610 which is executing the Mediated Call is bound to JP 10114.

The next step to determine whether the pointer which KOS Call microcode received from S-interpreter Call microcode is a pointer to an External Procedure. To make this determination, KOS Call microcode compares the pointer's AON 41304 with that of Procedure Object 608 for Procedure 602 making the Call. If they are different, the Call is a Cross-Procedure Object Call, described below. In the case of the Simple Mediated Call, the format field indicates that the location is an Entry Descriptor 47227. KOS Call microcode continues by saving the location of Entry Descriptor 47227 and creating a new Mediated Frame 46947 on current MAS Object 46703 and a new Ordinary SS Frame 10510 on SS Object 10336 for called Procedure 602. As KOS Call microcode does so, it sets Fields 46917 and 46919 in Mediated Frame Header 10414 and

Petitioner Apple Inc. - Exhibit 1004, p. 5337

Fields 47109 and 47111 in Ordinary SS Frame Header 10514 to the values required by the addition of frames to MAS Object 46703 and SS Object 10336.

New Mediated Frame 46947 is now ready for Linkage Pointers 10416 to the actual arguments used in the Call, so KOS Call microcode returns to S-interpreter Call microcode, which parses the SIN to obtain the literal specifying the number of arguments and saves the literal value. S-interpreter Call microcode then parses each argument Name, evaluates it, and places the resulting value in Linkage Pointers Section 10416. When Linkage Pointers Section 10416 is complete, S-Interpreter Call Microcode calculates the new location of FP from the location of the top of Linkage Pointers Section 10416 and places a pointer for the location in the FU 10120 register reserved for FP. At this time, S-interpreter Call microcode also places the new location of the top of the stack in Stack Top Offset Field 46807.

S-interpreter Call microcode then invokes KOS Call microcode to place the value of the literal specifying the number of arguments in MAS Frame Field 46929, to calculate the new value of FHP 46702 and place it in the FU 10120 register reserved for that value, and finally to obtain the state necessary to execute called Procedure 602 from called Procedure 602's Entry Descriptor 47227 and PED 30303. As previously stated, S-interpreter Call microcode saved the location of Entry Descriptor 47227. Using this location, KOS Call Microcode obtains the size of the storage required for local data from Field 47233 and adds that amount of storage to the new MAS Frame 46709. Then KOS Call Microcode uses Field 47231 to locate PED 30303 for Procedure 602. PED 30303 contains the remainder of the necessary information about Procedure 602 and KOS Call microcode copies the location of PED 30303 into PED Pointer Field 46933 and then copies the values of K Field 30305. Last Name Field 30307, NTP Field 30311 and PBP Field 30315 into the relevant registers in FU 10120. KOS Call microcode next translates the pointer in SIP Field 30309 into a dialect number as explained in Chapter 3, and places it in register RDIAL 24212 of FU 10220 and thereupon derives SDP by resolving the pointer in SDPP Field 30313 and a pointer to SEB 46864 by resolving the pointer in SEPP Field 30316. Having performed these operations, KOS Call microcode returns to S-interpreter Call microcode, which finishes the Call by obtaining a new PC, that is, resetting registers in I-stream Reader 27001 in FU 10120 so that the next SIN to be fetched will be the first SIN of called procedure 602 S-interpreter Call microcode obtains the information required to change PC from Field 47229 in Entry Descriptor 47227 which contains the offset of the first SIN of called Procedure 602 from PBP.

In the present embodiment, some FU 10120 state produced by the Mediated Call SIN is retained on SS 504 throughout the duration of Procedure 602's invocation. The saved state allows Process 610 to reattempt the Mediated Call if the Call fails before the called Procedure 602 begins executing. When a Mediated Return SIN is executed, it resumes execution on the retained state from the CALL SINT. The Mediated Return is much simpler than the Call. Since all of the information required to resume execution of the invocation which perfomed the Call is contained in Macrostate 10516 in the calling invocation's SS Frame 47003, Return need only pop the called invocation's frames from current MAS Object 46703 and SS Object 10336, copy Macrostate 10516 47123 from the calling invocation's SS Frame 47003 into the proper FU 10120 registers, translate SIP Value 47141 into a dialect number, and resume executing the calling invocation. The pop operation involves nothing more than updating those pointers in MAS Object 46703 and SS Object 10336 which pointed to locations in the old topmost frame so that they now point to equivalent locations in the new topmost frame.

c.c. Invocations of Procedures 602 Requiring SEBs 46864 (Fig. 270, 468, 469, 470, 471, 472)

If a Procedure 602 requires a SEB 46864, this fact is indicated by Flag Field 47237 in Procedure 602's Entry Descriptor 47227. PED 30303 for such a Procedure 602 contains SEPP Field 47225, whose value is a non-resolvable pointer. The manner in which a SEB 46864 is created for Procedure 602 and SEPP field 47225 is translated into SEP, a pointer which contains the location of SEB 46864 and is saved as part of the invocation's macrostate on SS 10336, is similar to the manner in which a Static Data Block 46863 is created and the non-resolvable pointer contained in SDPP field 47225 is translated into SDP. The first time that a Procedure 602 requiring a SEB 46864 is invoked on a MAS Object 46703, a SEB 46864 is created for the Procedure 602 and an AATE 46857 is created which associates the nonresolvable pointer in SEPP field 47225 and the location of SEB 46864. That location is the value of SEP when the procedure is executing on MAS object 46703. On subsequent invocations of Procedure 602. AATE 46857 serves to translate the value in SEPP field 47225 into SEP.

d.d. Cross-Procedure Object Calls (Fig. 270, 468, 469, 470, 471, 472)

A Mediated Call which invokes an External Procedure 602 is called a Cross-Procedure Object Call. As previously mentioned. KOS Call microcode assumes that any time the Name representing the called Procedure 602 in a Mediated Call SIN resolves to the location of a Gate that the Call is to an External Procedure 602. As long as newly-called External procedure 602 has the same DOE as calling Procedure 602. Cross-Procedure Object Calls differ from the Simple Mediated Call only in the manner in which called Procedure 602's Entry Descriptor 47227 is located. Once KOS Call microcode has determined as described above that a Mediated Call is a Cross-Procedure Object Call it must next determine whether it is a Cross-Domain Call. To do so, KOS Call microcode compares the DOE Attribute of called Procedure 602's Procedure Object 608 with the domain component of the current subject. KOS Call microcode uses Procedure Object 608's AON 41304 to obtain Procedure Object 608's DOE from Field 41521 of its AOTE

Petitioner Apple Inc. - Exhibit 1004, p. 5338

41306 and it uses the ASN for the current subject, stored in an FU 10120 register, to obtain the current subject's domain component from AST 10914. If the DOE and the current subject's domain component differ, the Call is a Cross-domain Call, described below; otherwise, the Call locates the Gate 47205 or 47206 specified by the evaluated Name for called Procedure 602 in its Procedure Object 608. If the Gate is a Local Gate 47205, the Call uses Entry Descriptor Offset Field 47207 to locate Entry Descriptor 47227 belonging to Called Procedure 602 and then proceeds as described in the discussion of a Simple Mediated Call.

If the Gate is a Link Gate 47206, KOS Call microcode obtains the pointer corresponding to Link Gate 47206 from Binder Area 47245 and resolves it to obtain a pointer to another Gate 47205 or 47206, which KOS Call microcode uses to repeat the External Procedure 602 call described above. The repetitions continue until the newly-located gate is a Local Gate 47205, whereupon Call proceeds as described for Simple Mediated Calls.

e.e. Cross-domain Calls (Fig. 270, 408, 418, 468, 469, 470, 471, 472)

If a called Procedure 602's Procedure Object 608 has a DOE attribute differing from that of calling Procedure 602's Procedure Object 608, the Call is a Cross-domain Call. The means by which KOS Call microcode determines that a Mediated Call is a Cross-Domain Call have previously been described; If the Call is a Cross-Domain Call, KOS Call microcode must inactivate MAS Object 46703 for the domain from which the Call is made, perform trojan horse argument checks, switch subjects, place a Cross-domain Frame 47039 on SS object 10336, and locate and activate MAS Object 46703 for the new domain before it can make a Mediated Frame 46947 on new MAS Object 46703 and continue as described in the discussion of a Simple Mediated Call.

Cross-domain Call microcode first inactivates the current MAS Object 46703 by setting Domain Active Flag 46804 to FALSE. The next step is the trojan horse argument checks. In order to perform trojan horse argument checks, Cross-domain Call must have pointers to the actual arguments used in the cross-domain invocation. Consequently, Cross-domain Call first continues like a non-cross-domain Call: it creates a Mediated Frame Header 10414 on old MAS Object 46703 and returns to S-interpreter microcode, which evaluates the Names of the actual arguments, and places the pointers in Linkage Pointers 10416 above Mediated Frame Header 10414. However, the macrostate for the invocation performing the call was placed on SS Object 10336 before Mediated Frame Header 10414 and Linkage Pointers 10416 were placed on old MAS Object 46703. Consequently, when calling Procedure 602 resumes execution after a Return, it will resume on MAS Frame 46709 preceding the one built by Cross-domain Call microcode.

Once the pointers to the actual arguments are available, Cross-domain Call Microcode performs the trojan horse check. As described in the discussion of Procedure Object 608 and illustrated in Figure 472, the information required to perform the check is contained in AIA 10352. Each Local Gate 47205 in Procedure Object 608 has an AIAE 47245, each formal argument in Local Gate 47205's procedure has an entry in AIAE 47245's AMA 47251, and the formal argument's AMAE 47253 indicates what kind of access to the formal argument's actual argument is required in called Procedure 602.

Field AIA OFF 47201 contains the location of AIA 10352 in Procedure Object 608, and using this information and Local Gate 47205's offset in Procedure Object 608, Cross-domain Call microcode locates AIAE 47245 for Local Gate 47205. The first two fields in AIAE 47245 contain the minimum number of arguments in the invocation and the maximum number of arguments. Cross-domain Call microcode checks whether the number of actual arguments falls between these values. If it does, Cross-domain Call microcode begins checking the access allowed individual arguments. For each argument pointer. Cross-domain Call microcode calls LAR microcode to obtain the current AON 41304 for the pointer's UID and uses AON 41304 and the ASN for Process 610's current subject (i.e., the caller's subject) to locate an entry in either APAM 10918 or ANPAT 10920, depending on whether the argument's AIAE specifies primitive access (47255) or extended access (47257) respectively. If the information from APAM 10918 or ANPAT 10920 confirms that Process 610's current subject has the right to access the argument in the manner required in called Procedure 602, the Trojan Horse microcode goes on to the next argument. If the current subject has the required access to all arguments, the trojan horse check succeeds and the Cross-domain Call continues, Otherwise, it fails and Cross-domain Call performs a microcode-to-software Call as explained below.

Next, Cross-domain Call microcode places Cross domain State 10513 on SS Object 10336. As explained in the discussion of SS object 10336, Cross-domain State 10513 contains the information required to return to the caller's frame on former MAS Object 46703. Having done this, Cross-domain Call microcode changes subjects. Using the current subject's ASN, Cross-Domain Call microcode obtains the current subject from AST 10914 replaces the subject's domain component with DOE Attribute 41225 for called Procedure 602's Procedure Object 608 and uses AST 10914 to translate the new subject thus obtained into a new ASN. That ASN then is placed in the appropriate FU 10120 register.

After the subject has been changed, Cross-domain Call microcode uses Domain Table 41801 to translate the DOE of called Procedure 602 into a domain number. Cross-domain Call microcode then uses the domain number as an index into Array of MAS AONs 46211 in VPSB 614 for Virtual Processor 612 belonging to Process 610 making the cross-domain call. The entry corresponding to the domain number contains AON 41304 of MAS Object 46703 for that domain.

Having located the proper MAS Object 46703, Cross-domain Call microcode uses STO field 46807 in MAS Header 10410 belonging to the new domains MAS Object 46703 to locate the top of the last MAS

Petitioner Apple Inc. - Exhibit 1004, p. 5339

Frame 46709. It then saves the value of FHP 46702 used in the preceding invocation in a FU 10120 register, adds a Mediated Frame Header 10414 to the top of MAS Object 46703, and calculates a new FHP 46702 which points to new Mediated Frame Header 10414. KOS Cross-Domain Call microcode then places the old value of FHP 46702 in FHP Value Field 47151 of SS Object 10336 and the old value of STO 46704 (pointing to the top of the last complete MAS Frame 46709 on previous MAS Object 46703) in Field 47153 of Cross-Domain State 10513 and fills in Mediated Frame Header 10414 fields as follows: Result of Cross-domain Call Field 46903 is set to TRUE. Previous Frame Offset Field 46917 is set to 0, and Dynamic Back Pointer Field 46931 is set to the saved value of FHP 46702. Dynamic Back Pointer Field 46931 thus points to the header of the topmost Mediated Frame 46947 on the previous MAS Object 46703. The values of the remaining fields are copied from Mediated Frame Header 10414 which Cross-Domain Call created on previous MAS Object 46703.

Cross-domain Call microcode next copies the argument pointers for the formal arguments from the top of previous MAS Object 46703 to new Mediated Frame 46947 and calculates FP. Cross-domain Call Microcode finishes by returning to S-interpreter Call microcode, which completes the Call as described for Simple Mediated Calls.

Except for the work involved in transferring to a new MAS Object 46703, Cross-domain Return is like other Returns from Mediated Calls. Old FHP 46701 from Field 47151 of Cross-Domain State 10513 and old STO 46704, from Field 47153 of Cross-domain State are placed in FU 10120 registers. Then the frames belonging to the invocation that is ending are popped off of SS Object 10336 and off of MAS Object 46703 belonging to the domain of called Procedure 602 and MAS Object 46703 is inactivated by setting Domain-Active Flag 46804 to FALSE. Then KOS Cross-domain Return microcode uses old FHP 46701 and old STO 46704 to locate MAS Object 46703 being returned to and the topmost Mediated Frame 46947 on that MAS Object 46703. MAS Object 46703 being returned to is activated, and finally, the contents of Macrostate 10516 belonging to the invocation being returned to are placed in the appropriate registers of FU 10120 and execution of the invocation resumes.

f.f. Failed Cross-Doamin Calls (Fig. 270, 468, 469, 470, 471, 472)

A Cross-Domain Call as described above may fail at several points between the time that the calling invocation begins the call and called Procedure 602 begins executing. On failure, Cross-Domain Call microcode performs a microcode-to-software Call. KOS Procedures 602 invoked by this Call may remedy the reason for the Cross Domain Call's failure and reattempt the Cross-domain Call. This is possible because the implementation of Cross Domain Call in CS 10110 saves sufficient FU 10120 state to allow Process 610 executing the Cross-Domain Call to return to the invocation and the Mediated Call SIN from which the Cross-Domain Call began. On failure, the invocation's MAS Frame 46709 may be located from the values of STO Field 47153 and FHP Field 47151 in Cross-Domain State 10513, and the Mediated Call SIN may be located by using information saved in FU 10120 state.

6. Neighborhood Calls (Fig. 468, 479, 472)

As previously mentioned, Procedures 602 called via Neighborhood Calls must have the same PED 30303 as calling Procedure 602. The only macrostate values which are not part of PED 30303 are PC and FP; consequently Neighborhood Call need only save PC and FP of the invocation performing the call and calculate these values for the new invocation. In addition, Neighborhood Call saves STO 46704 in order to make it easier to locate the top of the previous invocation's Neighborhood Frame 46947. Neighborhood Return simply restores the saved values. Since the macrostate values copied from or obtained via PED 30303 do not change during the sequence of invocations, and therefore need not be saved on SS Object 10336. Neighborhood Calls do not have SS Frames 47003.

The invention may be embodied in yet other specific forms without departing from the spirit or essential characteristics thereof. Thus, the present embodiments are to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than by the foregoing description.

Claims

1. A digital computer system (CS 101) including processor means (JP 114) for performing operations upon operands, memory means (MEM 112) for storing said operands and procedures, said procedures including instructions for controlling said operations and names referring to certain of said operands to be operated upon, ALU means (2034, 2074) for performing said operations, bus means (MOD 140, JPB 142) for conducting said instructions, names and operands between said memory means and said processor means, and I/O means (IOS 116) for conducting at least said operands between said memory means and devices external to said digital computer system, characterised in that said processor means (JP 114) comprises means for addressing said operands, including name table means (10350) for storing name table entries, each name table entry corresponding to one of said names included in each one of said procedures and each name table entry comprising first data from which may be determined an address of a location in said memory means of the operand referred to by one of said names and second data identifying a format of that operand, and translation means (NAME TRANS UNIT 27015) connected to said

Petitioner Apple Inc. - Exhibit 1004, p. 5340

bus means and responsive to said name table entries for providing outputs to said memory means representing said addresses, and further characterised in that said instructions are intermediate level S-language instructions from a plurality of sets of such instructions, each set corresponding to a particular higher level user programming language, and further characterised by receiving means (INSTB 20262) connected to said bus means for receiving said instructions from said memory means, and microcode control means (10240, 27003, 27013) connected between said receiving means and said ALU means for providing sequences of microinstructions for controlling said ALU means, said sequences being selected from a plurality of sequences of microinstructions corresponding to said S-language instructions respectively.

2. A digital computer system according to claim 1, characterised in that the S-language instructions have a uniform, fixed format.

3. A digital computer system according to claim 1 or 2, characterised in that the names are of uniform length and format.

4. A digital computer system according to any of claims 1 to 3, characterised in that each procedure further includes a name table pointer (NTP 30311) representing a base location in said memory means (MEM 112), and said first data of each name table entry contains information from which may be determined an address offset of a memory location relative to the base location, and in that said translation means (NAME TRANS UNIT 27015) further comprises base register means (NCR, MCR 10366) connected to said bus means for receiving and storing said name table pointer of the procedure currently controlling the operations performed by said ALU means.

5. A digital computer system according to any of claims 1 to 4, characterised by name cache means (10226) connected to outputs of said translation means (NAME TRANS UNIT 27015) and having outputs to said memory means (MEM 112) for storing said addresses, and further connected to said receiving means (INSTB 20262) and responsive to said names to provide name cache outputs to said memory means representing said addresses of certain operands for which said name cache means has stored said addresses.

6. A digital computer system according to any of claims 1 to 5, characterised in that each of said S-Language instructions is a member of an S-Language dialect of a plurality of S-Language dialects, and in that said receiving means (INSTB 20262) further comprises dialect code means (RDIAL 24212) for storing a dialect code specifying the dialect of which the received S-Language instructions are members, and in that said sequences of microinstructions include a set of sequences of microinstructions, corresponding to each said S-Language dialect, each set of sequences of microinstructions including at least one sequence of microinstructions corresponding to each S-Language instruction in a corresponding S-Language dialect, and in that said microcode control means (10240, 27003, 27013) is responsive to the dialect code and to each received S-Language instruction to provide to said ALU means (2034, 2074) a sequence of microinstructions corresponding to that S-Language instruction.

7. A digital computer system according to claim 1 or 2, characterised in that each procedure includes a dialect code denoting an S-Language dialect of which the S-Language instructions of the procedure are members, and in that said microcode control means (10240, 27003, 27013) further comprises control store means (SITT 11012) for storing said sequences of microinstructions for controlling said ALU means (2034, 2074), and dispatch table means (SIDT 11010) for storing addresses corresponding to locations in said control store means of each sequence of microinstructions, and in that said dispatch table means is responsive to said dialect code and to each instruction to provide to said control store means each address corresponding to said at least one microinstruction sequence corresponding to each said instruction, and said control store means is responsive to each address to provide to said ALU means said sequence of microinstructions corresponding to each instruction.

8. A digital computer system according to claim 1, 6 or 7, characterised in that said microcode control means (10240, 27003, 27013) comprises writable control store means (11012) connected to said bus means for storing said sequences of microinstructions, and control store addressing means (SITTNAS 20286) responsive to each S-Language instruction and to operation of said processor means for generating control store read addresses and write addresses (CSADR 20204), and in that said writable control store means is responsive to said read addresses to provide said sequences of microinstructions to said ALU means (2034, 2074) and is responsive to said write addresses to store said sequences of microinstructions.

9. A digital computer system according to claim 7, characterised in that said control store means (SITT 11012) comprises writable control store means connected to said bus means for storing said sequences of microinstructions, and in that said dispatch table means comprises write address means responsive to operation of said processor means for generating write addresses, and in that said writable control store means is responsive to said write addresses for storing said sequences of microinstructions.

Patentansprüche

1. Digitales Datenverarbeitungssystem (CS 101), enthaltend: Prozessormittel (MEM 114) zur Durchführung von Operationen an Operanden, Speichermittel (MEM 112) zum Speichern der Operanden und von Prozeduren, die Befehle zur Steuerung der Operationen und Namen enthalten, die auf gewisse der Operanden Bezug nehmen, an denen Operationen durchgeführt werden sollen, eein Rechenwerk (2034,

Petitioner Apple Inc. - Exhibit 1004, p. 5341

2074) zur Durchführung der Operationen, Bus-Mittel (MOD 140, JPE 118) für den Verkehr der Befehle, Namen und Operanden zwischen den Speichermitteln und den Prozessormitteln, und Eingabe/Ausgabe-Mittel (IOS 116) für den Verkehr wenigstens der Operanden zwischen den Speichermitteln und Geräten außerhalb des digitalen Datenverarbeitungssystems, gekennzeichnet durch Prozessormittel (JP 114), die Mittel zur Adressierung der Operanden einschließlich Namenstabellenmittel (10350) zur Speicherung von Namenstabellen-Einsprungpunkten enthalten, wobei jeder Namenstabellen-Einsprungpunkt einem der Namen entspricht, die in jeder der Prozeduren enthalten sind, und erste Daten, aus denen eine Adresse eines Platzes derjenigen Operanden in den Speichermitteln bestimmt werden kann, auf die durch einen der Namen Bezug genommen wird, und zweite Daten enthalten die ein Format dieses Operanden identifizieren, und durch Übersetzungsmittel (NAME TRANS UNIT 27015), die mit den Bus-Mitteln verbunden sind und auf die Namenstabellen-Einsprungpunkte unter Bereitstellung von diese Adressen repräsentierenden Ausgaben für die Speichermittel ansprechen, ferner dadurch gekennzeichnet, daß die Befehle mittlere S-Sprache-Befehle von einer Vielzahl von Sätzen solcher Befehle sind, von denen jeder Satz einer besonderen höheren Benutzerprogrammiersprache entspricht, und ferner gekennzeichnet durch ein mit den Bus-Mitteln verbundenes Empfangsmittel (INSTB 20262) zum Empfang der Befehle von den Speichermitteln, und durch mit dem Empfangsmittel und dem Rechenwerk verbundene Mikrocode-Steuermittel (10240, 27003, 27013) zur Bereitstellung von Mikrobefehlssequenzen zur Steuerung des Rechenwerks, wobei diese Sequenzen aus einer Vielzahl von Mikrobefehlssequenzen ausgewählt sind, die den jeweiligen S-Sprache-Befehlen entsprechen.

2. Digitales Datenverarbeitungssystem nach Anspruch 1, dadurch gekennzeichnet, daß die S-Sprache-Befehle ein gleichförmiges, festes Format haben.

3. Digitales Datenverarbeitungssystem nach Anspruch 1 oder 2, dadurch gekennzeichnet, daß die Namen eine gleichförmige Länge und ein gleichförmiges Format haben.

4. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, daß jede Prozedur weiter einen Namenstabellenzeiger (NTP 30311) enthält, der einen Basisplatz in den Speichermitteln (MEM 112) repräsentiert, daß die ersten Daten jedes Namenstabellen-Einsprungpunktes Informationen enthalten, aus denen die Adresse eines vom Basisspeicherplatz versetzten Speicherplatzes bestimmt werden können, und daß die Übersetzungsmittel (NAME TRANS UNIT 27015) weiter Basisregistermittel (NCR, MCR 10366) enthalten, die mit den Bus-Mitteln verbunden sind, um den Namenstabellenzeiger derjenigen Prozedur zu empfangen und zu speichern, die gerade die vom Rechenwerk durchgeführten Operationen steuert.

5. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 4, gekennzeichnet durch Namens-Cache-Speichermittel (10226), die mit den Ausgängen der Übersetzungsmittel (NAME TRANS UNIT 27015) verbunden sind und zu den Speichermitteln (MEM 112) führend Ausgänge zum Speichern der Adressen haben, und die weiter mit dem Empfangsmittel (INSTB 20262) verbunden sind und auf die Namen unter Bereitstellung von Namens-Cache-Ausgaben für die Speichermittel ansprechen, die die Adressen von gewissen Operanden repräsentieren, für die die Namens-Cache-Speichermittel die Adressen gespeichert haben.

6. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 5, dadurch gekennzeichnet, daß jeder der S-Sprache-Befehle ein Mitglied eines S-Sprache-Dialekts einer Vielzahl von S-Sprache-Dialekten ist, daß das Empfangsmittel (INSTB 20262) weiter ein Dialekt-Code-Mittel (RDIAL 24212) zur Speicherung eines Dialekt-Codes enthält, der den Dialekt bestimmt, von dem die empfangenen S-Sprache-Befehle Mitglieder sind, daß die Mikrobefehlssequenzen einen Satz von Mikrobefehlssequenzen entsprechend jedem S-Sprache-Dialekt enthalten, wobei jede Mikrobefehlssequenz wenigstens eine jedem S-Sprache-Befehl in einem entsprechenden S-Sprache-Dialekt entsprechenden Mikrobefehlssequenz enthält, und daß die Mikrocode-Steuermittel (10240, 27003, 27013) auf den Dialekt-Code und jeden empfangenen S-Sprache-Befehl unter Bereitstellung einer diesem S-Sprache-Befehl entsprechenden Mikrobefehlssequenz für das Rechenwerk ansprechen.

7. Digitales Datenverarbeitungssystem nach Anspruch 1 oder 2, dadurch gekennzeichnet, daß jede Prozedur einen Dialektcode enthält, der einen S-Sprache-Dialekt bezeichnet, von dem die S-Sprache Befehle der Prozedur Mitglieder sind, daß die Mikrocode-Steuermittel (10240, 27003, 27013) ferner Steuerspeichermittel (SITT 11012) zur Speicherung der Mikrobefehlssequenzen für die Steuerung des Rechenwerks (2034, 2074) und Verteilartabellenmittel (SIDT 11010) zur Speicherung von Adressen enthalten, die Plätzen jeder Mikrobefehlssequenz in den Steuerspeichermitteln entsprechen, und daß die Verteilertabellenmittel auf den Dialektcode und jeden Befehl unter Bereitstellung jeder Adresse, die der wenigstens einen, zu jedem Befehl gehörenden Mikrobefehlssequenz entspricht, für die Steuerspeichermittel ansprechen, während die Steuerspeichermittel auf jede Adresse unter Bereitstellung der jedem Befehl entsprechenden Mikrobefehlssequenz für das Rechenwerk ansprechen.

8. Digitales Datenverarbeitungssystem nach Anspruch 1, 6 oder 7, dadurch gekennzeichnet, daß die Mikrocode-Steuermittel (10240, 27003, 27013) ein mit den Bus-Mitteln verbundenes Schreibsteuerspeichermittel (11012) zur Speicherung der Mikrobefehlssequenzen und Steuerspeicheradressiermittel (SITTNAS 20286) enthalten, die auf jeden S-Sprache-Befehl und auf Operationen des Prozessormittels unter Erzeugung von Steuerspeicherlese- und -schreibadressen (CSADR 20204) ansprechen, und daß die Schreibsteuerspeichermittel auf die Leseadressen unter Bereitstellung der Mikrobefehlssequenzen für das Rechenwerk und auf die Schreibadressen unter Speicherung dieser Mikrobefehlssequenzen ansprechen.

Petitioner Apple Inc. - Exhibit 1004, p. 5342

9. Digitales Datenverarbeitungssystem nach Anspruch 7, dadurch gekennzeichnet, daß die Steuerspeichermittel (SITT 11012) mit den Bus-Mitteln verbundene Schreibsteuerspeichermittel zur Speicherung der Mikrobefehlssequenzen enthalten, daß die Verteilertabellenmittel Schreibadressenmittel enthalten, die auf Operationen des Prozessormittels unter Erzeugung von Schreibadressen ansprechen, und daß die Schreibsteuerspeichermittel auf die Schreibadressen unter Speicherung der Mikrobefehlssequenzen ansprechen.

Revendications

1. Un système d'ordinateur numérique (CS 101), comprenant un processeur (JP 114) pour effectuer des opérations sur des opérandes, une mémoire (MEM 112) pour mémoriser lesdits opérandes et des procédures, lesdites procédures contenant des instructions pour commander lesdites opérations et des désignations se rapportant à certains desdits opérandes pour les traiter, une unité arithmétique et logique ALU (2034, 2074) pour effectuer lesdites opérations, des bus (MOD 140, JPB 142) pour transmettre lesdites instructions, lesdites désignations et lesdits opérandes entre ladite mémoire et ledit processeur, et des moyens d'entrée/sortie I/O (IOS 116) pour transmettre au moins lesdits opérandes entre ladite mémoire et des dispositifs extérieurs audit système d'ordinateur numérique, caractérisé en ce que ledit processeur (JP 114) comprend des moyens pour l'adressage desdits opérandes, comportant une table de désignations (10350) pour mémoriser des entrées de table de désignations, chaque entrée de table de désignations correspondant à une desdites désignations incluses dans chacune desdites procédures et chaque entrée de table de désignations comprenant une première donnée à partir de laquelle peut être déterminée une adresse d'un emplacement de ladite mémoire contenant l'opérande auquel se reflète l'une desdites désignations et une seconde donnée identifiant un format de cet opérande, et des moyens de transcodage (NAME TRANS UNIT 27015) reliés auxdits bus et réagissant auxdites entrées de tables de désignations de façon à transmettre à ladite mémoire des signaux de sortie représentant lesdites adresses, et en outre caractérisé en ce que lesdites instructions sont des instructions en langage-S de niveau intermédiaire provenant d'une pluralité d'ensembles de telles instructions, chaque ensemble correspondant à un langage de programmation par utilisateur de niveau supérieur particulier, et en outre caractérisé en ce que des moyens de réception (INSTB 20262) sont reliés auxdits bus pour recevoir lesdites instructions à partir de ladite mémoire, et des moyens de commande de microcode (10240, 27003, 27013) connectés entre lesdits moyens de réception et ladite ALU pour fournir des séquences de microinstructions servant à commander ladite ALU, les dites séquences étant sélectionnées parmi une pluralité de séquences de micro-instructions correspondant respectivement auxdites instructions en langage-S.

2. Un système d'ordinateur numérique selon la revendication 1, caractérisé en ce que les instructions en langage-S ont un format fixe et uniforme.

3. Un système d'ordinateur numérique selon une des revendications 1 ou 2, caractérisé en ce que les désignations ont une longueur et un format uniformes.

4. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 3, caractérisé en ce que chaque procédure comprend en outre un pointeur de table de désignations (NTP 30311) représentant un emplacement de base dans ladite mémoire (MEM 112) et ladite première donnée de chaque entrée de la table de désignations contient une information à partir de laquelle peut être déterminé un décalage d'adresse d'un emplacement de mémoire par rapport à l'emplacement de base, et en ce que lesdits moyens de transcodage (NAME TRANS UNIT 27015) comprennent en outre un moyen formant registre de base (NCR, MCR 10366), qui est relié auxdits bus de façon à recevoir et mémoriser ledit pointeur de table de désignations dans la procédure qui est en train de commander les opérations effectuées par ladite ALU.

5. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 4, caractérisé par un moyen formant antémémoire de désignations (10226), relié aux sorties desdits moyens de transcodage (NAME TRANS UNIT 27015) et comportant des sorties reliées à ladite mémoire (MEM 112) pour mémoriser lesdites adresses, et en outre relié auxdits moyens de réception (INSTB 20262) et réagissant auxdites désignations pour fournir à ladite mémoire des sorties de l'antémémoire de désignations représentant lesdites adresses de certains opérandes pour lesquels ladite antémémoire de désignations a mémorisé lesdites adresses.

6. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 5, caractérisé en ce que chacune desdites instructions en langage-S est un élément d'un dialecte en langage-S faisant partie d'une pluralité de dialectes en langage-S et en ce que lesdits moyens de réception (INSTB 20262) comprennent en outre un moyen de codage de dialecte (RDIAL 24212) pour mémoriser un code de dialecte spécifiant le dialecte dont les instructions en langage-s reçues sont des éléments, et en ce que lesdites séquences de micro-instructions contiennent un ensemble de séquences de micro-instructions correspondant à chacun desdits dialectes en langage-S, chaque ensemble de séquences de micro-instructions comprenant au moins une séquence de micro-instructions correspondant à chaque instruction en langage-S dans un dialecte en langage-S correspondant, et en ce que lesdits moyens de commande de microcode (10240, 27003, 27013) réagissent audit code de dialecte et à chaque instruction en langage-S reçue pour fournir à ladite ALU (2034, 2074) une séquence de micro-instructions correspondant à cette instruction en langage-S.

Petitioner Apple Inc. - Exhibit 1004, p. 5343

7. Un système d'ordinateur numérique selon une des revendications 1 et 2, caractérisé en ce que chaque procédure comprend un code de dialecte définissant un dialecte en langage-S dont les instructions en langage-S de la procédure sont des éléments et en ce que lesdits moyens de commande de microcode (1020, 27003, 27013 comprennent en outre une mémoire de commande (SITT 11012) pour mémoriser lesdites séquences de micro-instructions pour commander ladite ALU (2034, 2074), et un moyen à table de distribution (SIDT 11010) pour mémoriser des adresses correspondant aux emplacements de chaque séquence de micro-instructions dans ladite mémoire de commande, et en ce que ledit moyen à table de distribution réagit audit code de dialecte et à chaque instruction pour fournir à ladite mémoire de commande chaque adresse correspondant à ladite séquence de micro-instructions au moins prévue correspondant à chacune desdites instructions, et ladite mémoire de commande réagit à chaque adresse pour fournir à ladite ALU ladite séquence de micro-instructions correspondant à chaque instruction.

8. Un système à ordinateur numérique selon une des revendications 1, 6 et 7, caractérisé en ce que lesdits moyens de commande de microcode (10240, 27003, 27013) comprennent une mémoire de commande inscriptible (11012) reliée auxdits bus pour mémoriser lesdites séquences de micro-instructions et un moyen d'adressage de mémoire de commande (SITTNAS 20286) réagissant à chaque instruction en langage-S et au fonctionnement dudit processeur pour produire des adresses de lecture et des adresses d'écriture dans la mémoire de commande (CSADR 20204) et en ce que ladite mémoire de commande inscriptible réagit auxdites adresses de lecture pour fournir lesdites séquences de micro-instructions à ladite ALU (2034, 2074) et réagit auxdites adresses d'écriture pour mémoriser lesdites séquences de micro-instructions.

9. Un système d'ordinateur numérique selon la revendication 7, caractérisé en ce que ladite mémoire de commande (SITT 11012) comprend une mémoire de commande inscriptible qui est reliée auxdits bus de mémoriser lesdites séquences de micro-instructions et en ce que ledit moyen à table de distribution comprend un moyen d'adressage d'écriture réagissant au fonctionnement dudit processeur pour produire des adresses d'écriture, et en ce que la mémoire de commande inscriptible réagit auxdites adresses d'écriture pour mémoriser lesdites séquences de micro-instructions.

Petitioner Apple Inc. - Exhibit 1004, p. 5344

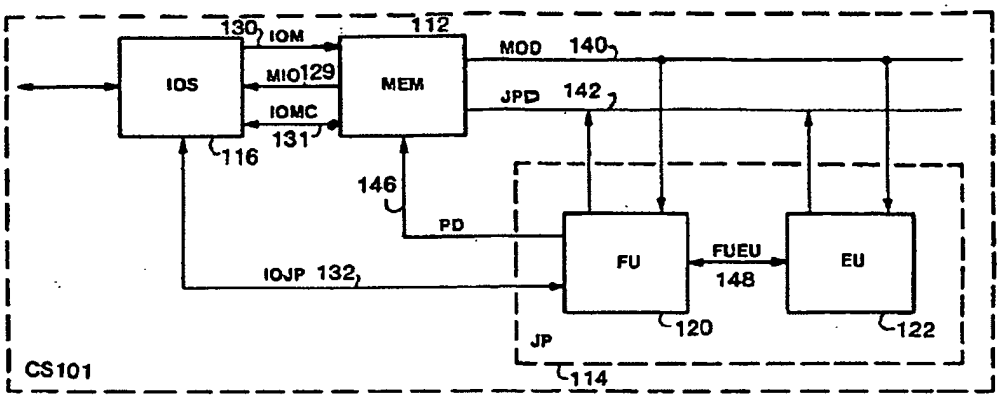


FIG 1

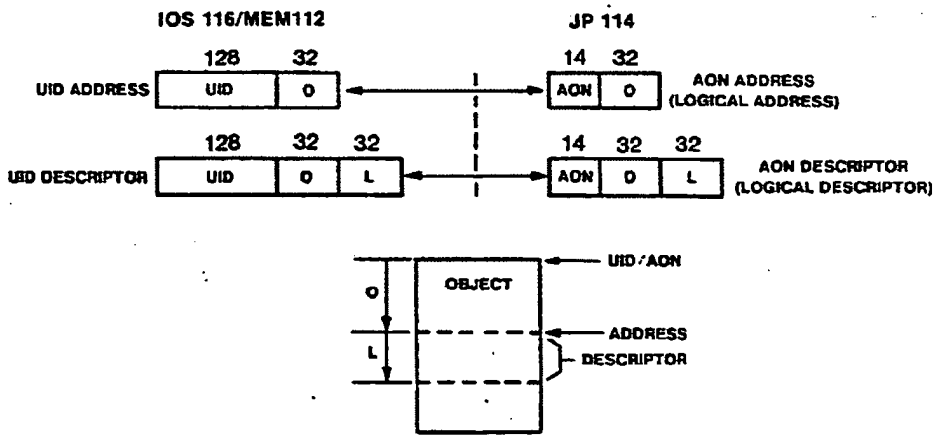


FIG 2

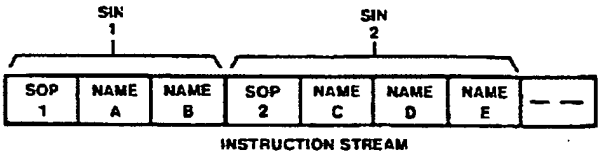


FIG 3

Petitioner Apple Inc. - Exhibit 1004, p. 5345

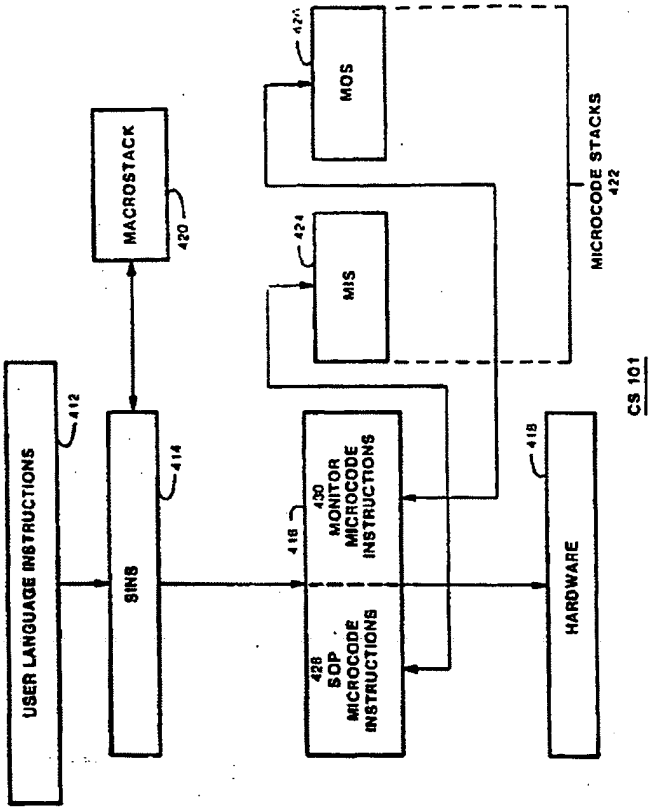


FIG 4A

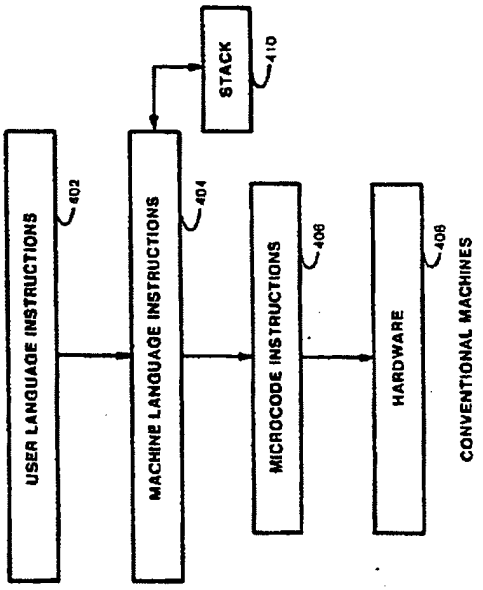


FIG 4

Petitioner Apple Inc. - Exhibit 1004, p. 5346

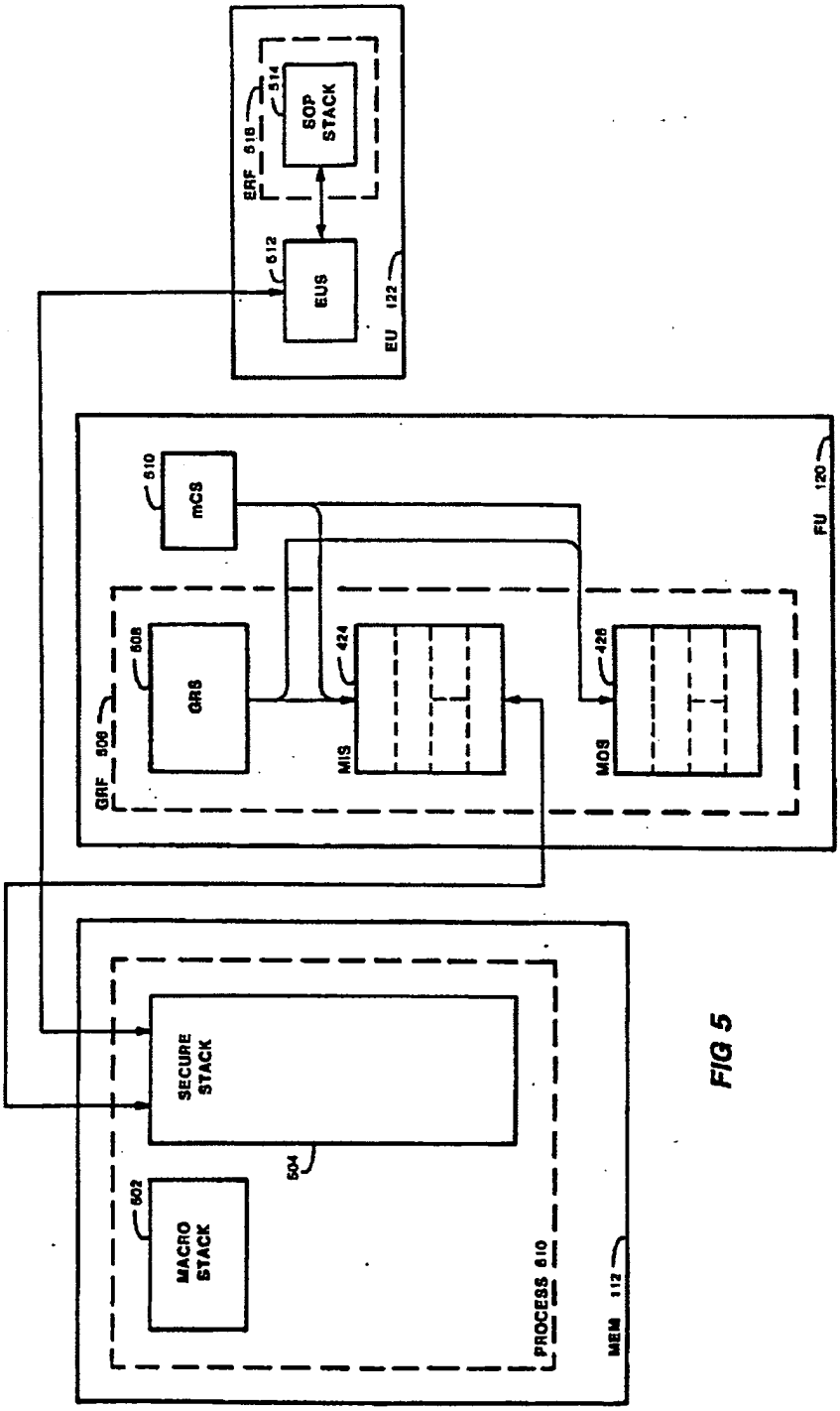


FIG 5

Petitioner Apple Inc. - Exhibit 1004, p. 5347

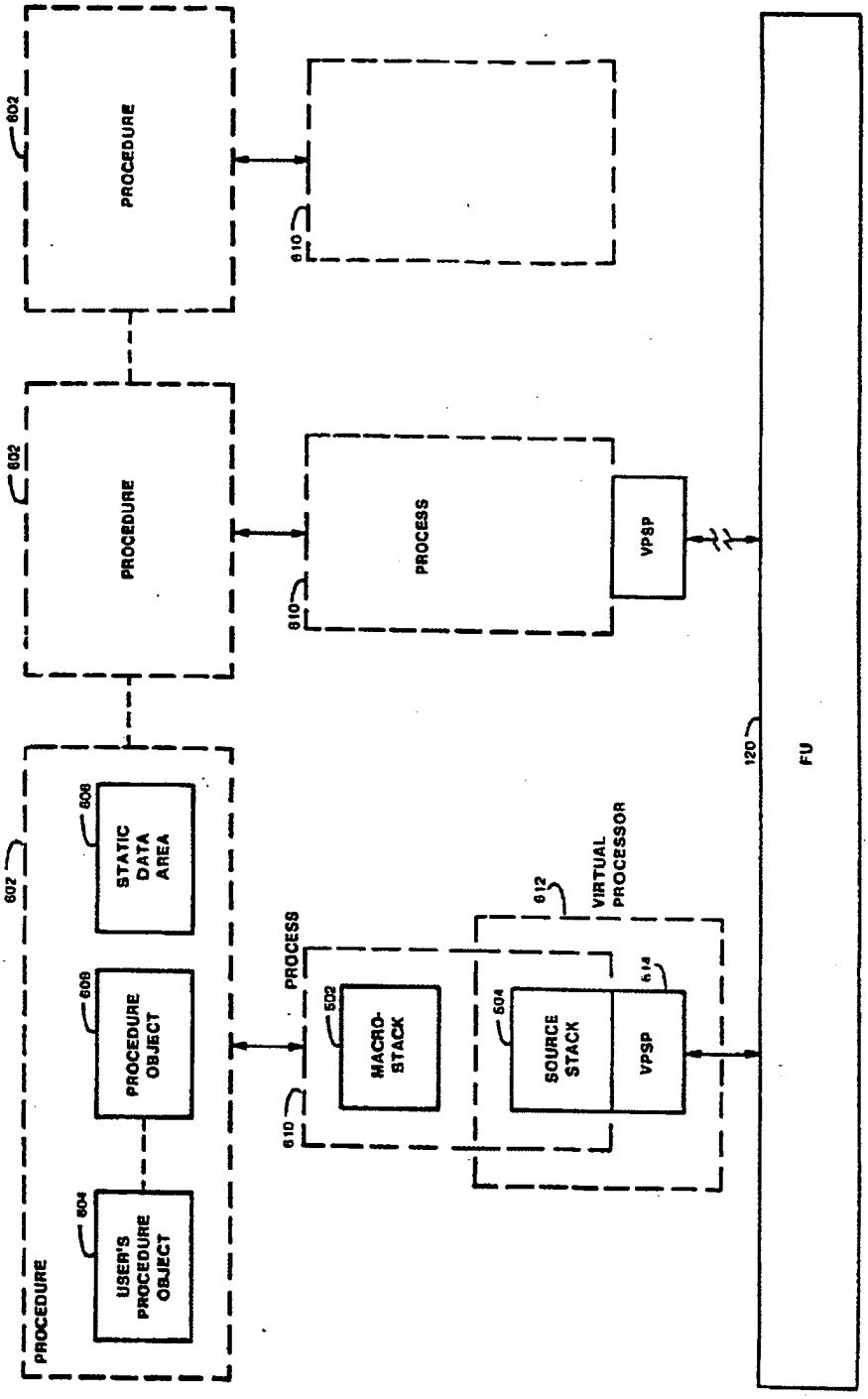


FIG 6

Petitioner Apple Inc. - Exhibit 1004, p. 5348

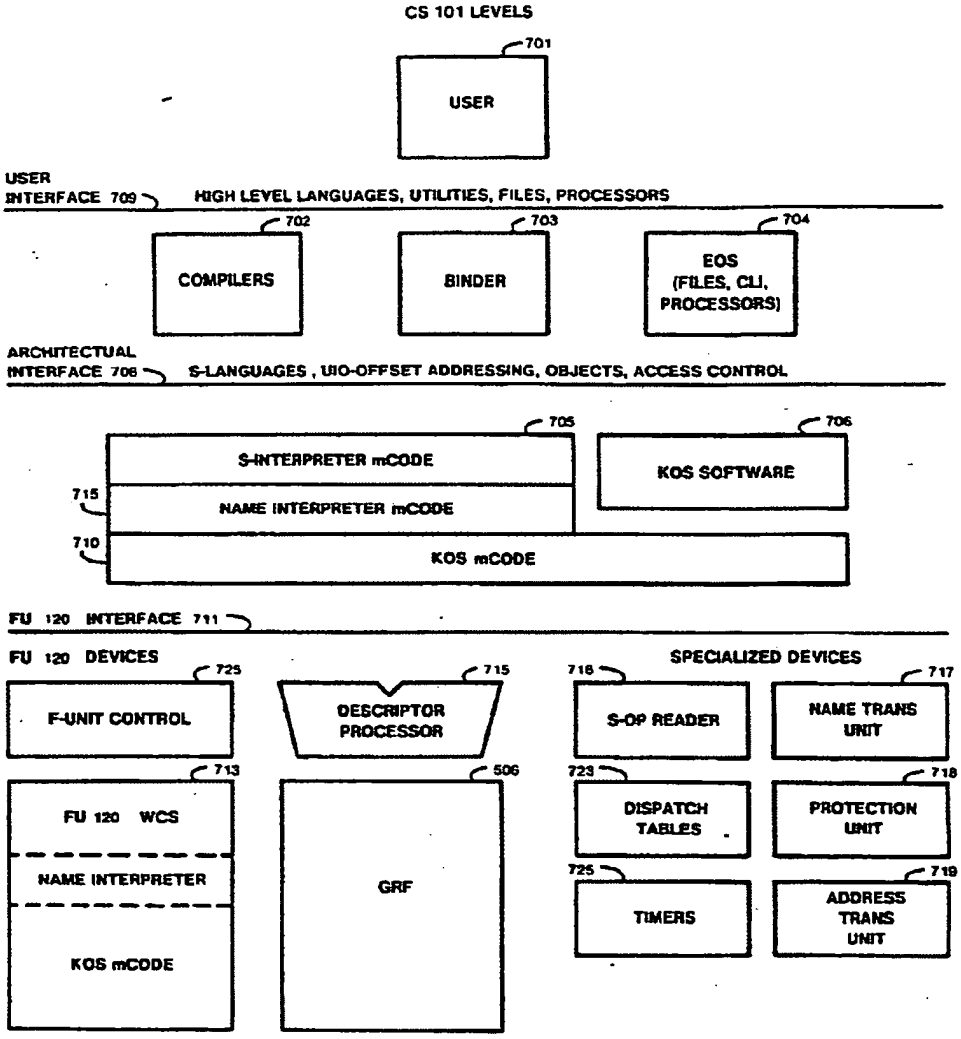


FIG 7

Petitioner Apple Inc. - Exhibit 1004, p. 5349

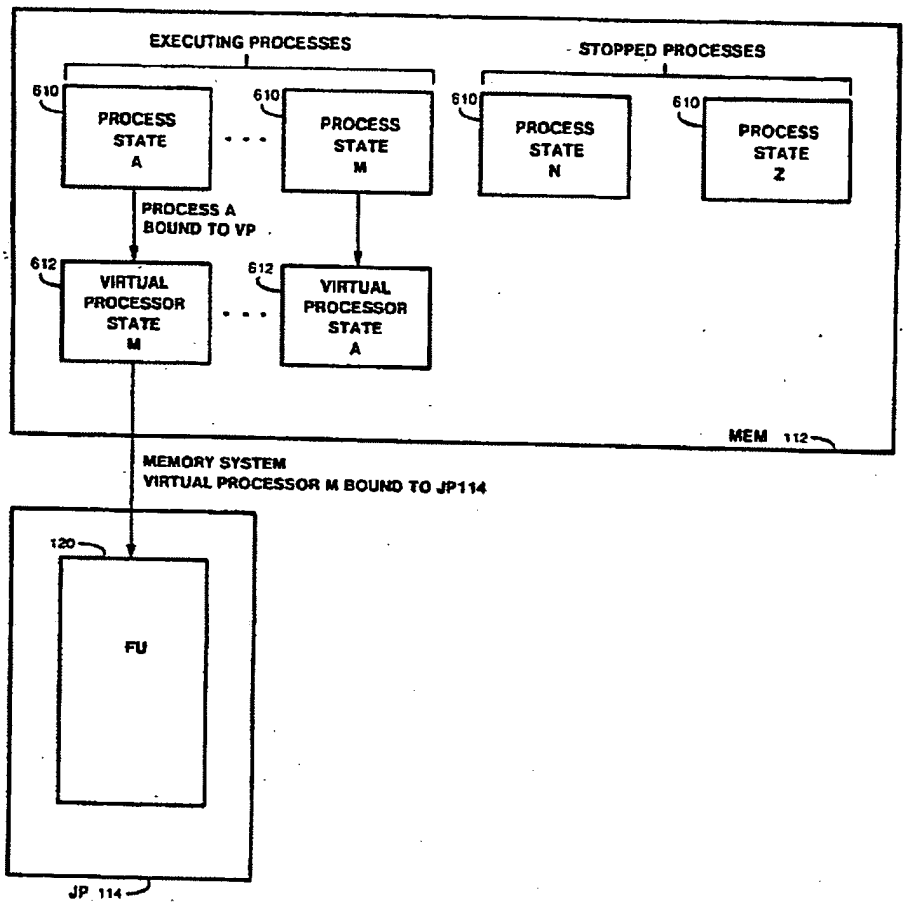


FIG 8

Petitioner Apple Inc. - Exhibit 1004, p. 5350

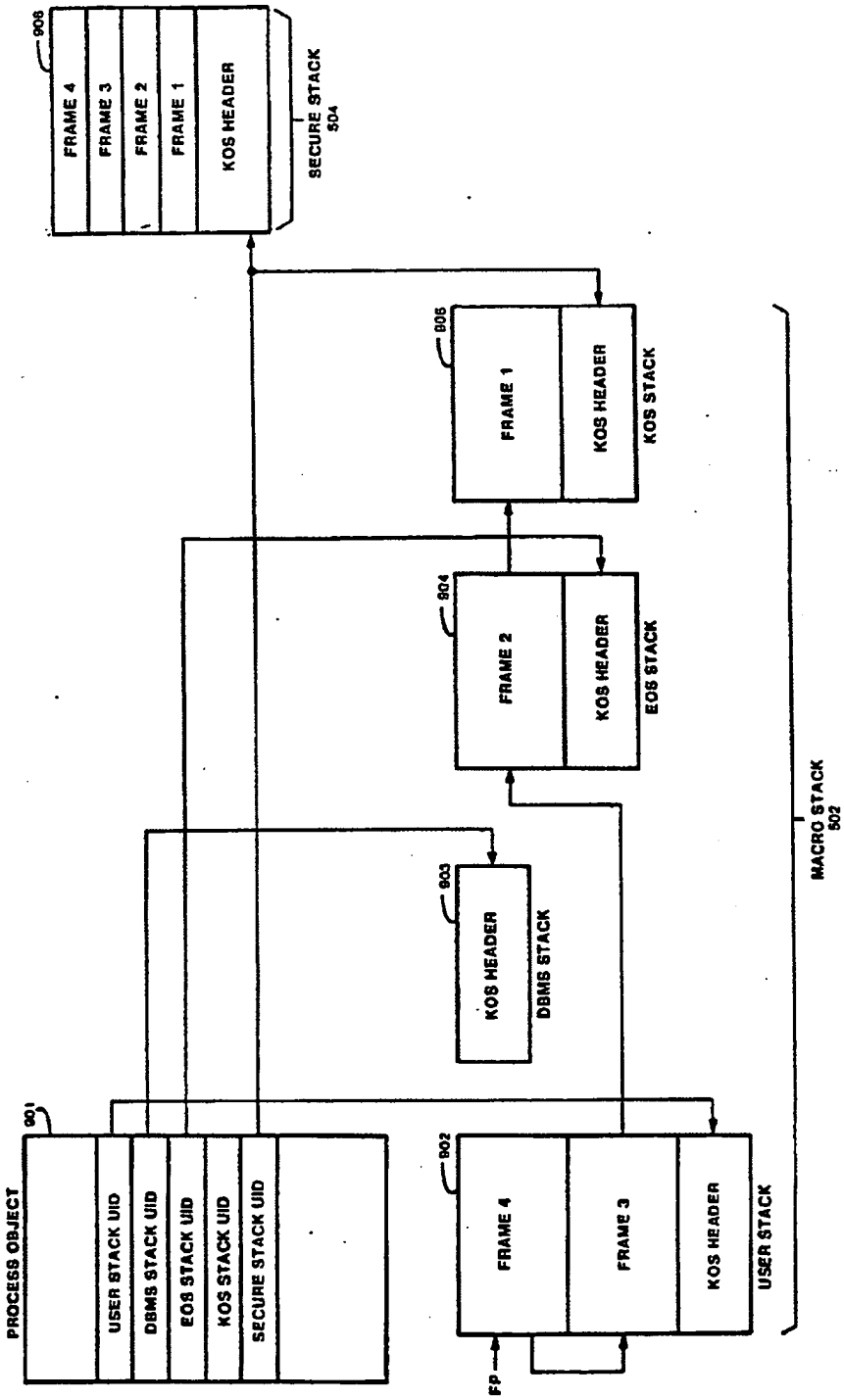


FIG 9

Petitioner Apple Inc. - Exhibit 1004, p. 5351

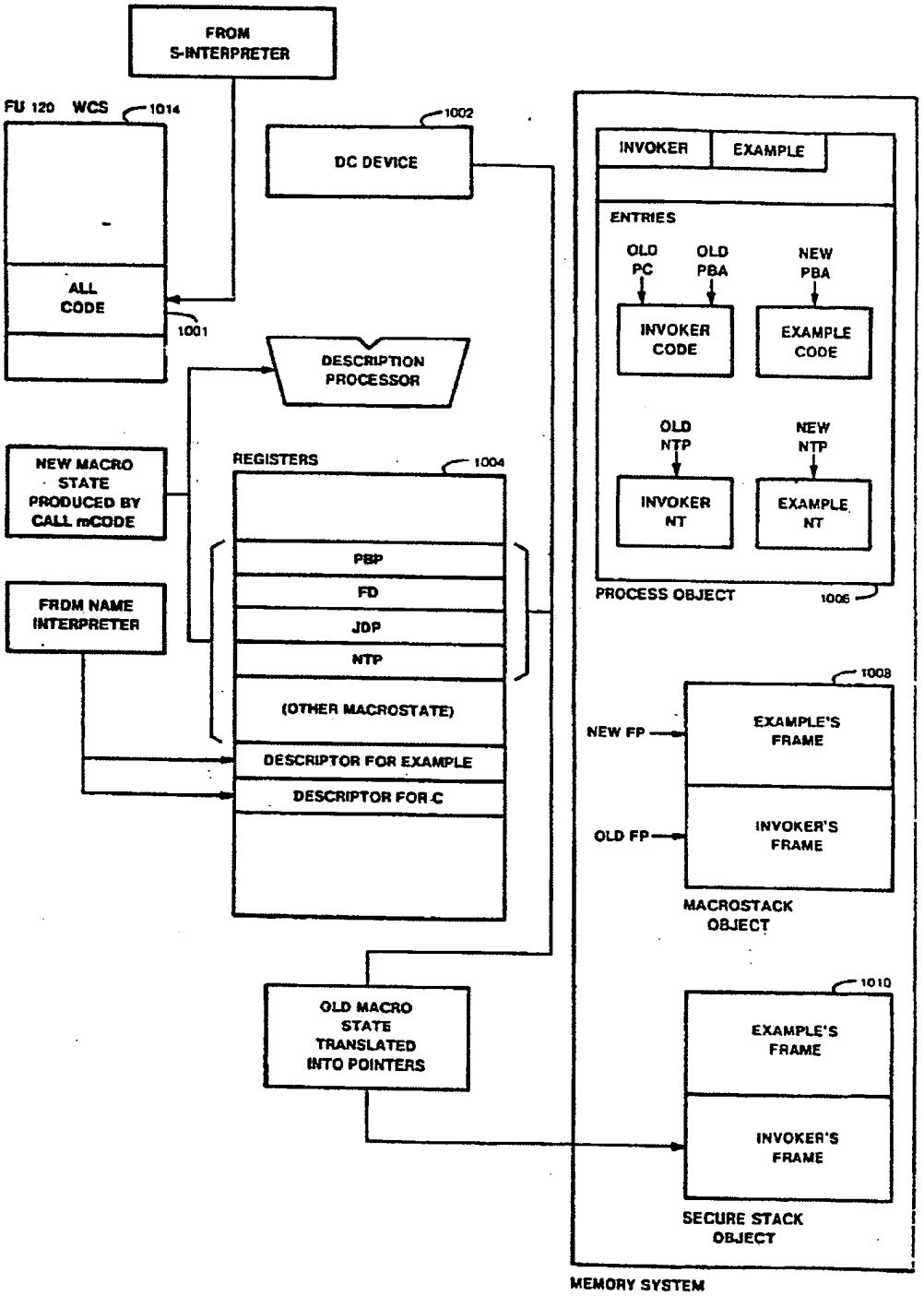


FIG 10

Petitioner Apple Inc. - Exhibit 1004, p. 5352

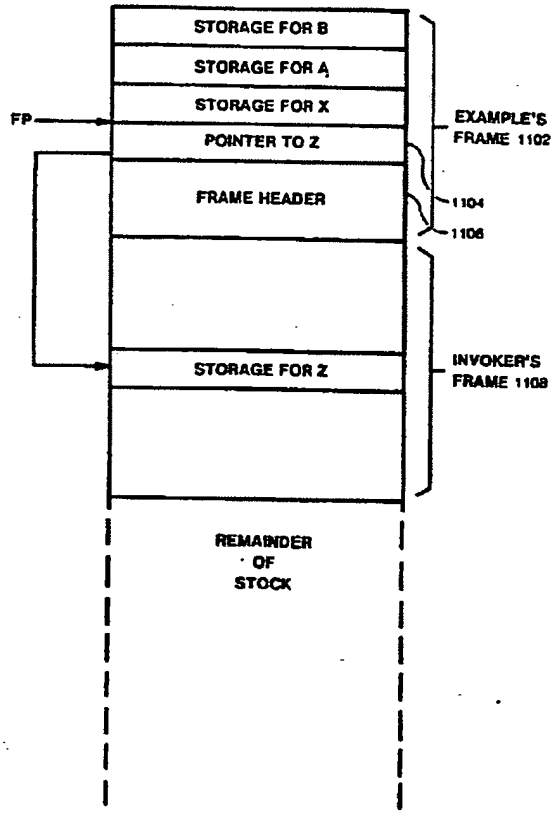


FIG 11

Petitioner Apple Inc. - Exhibit 1004, p. 5353

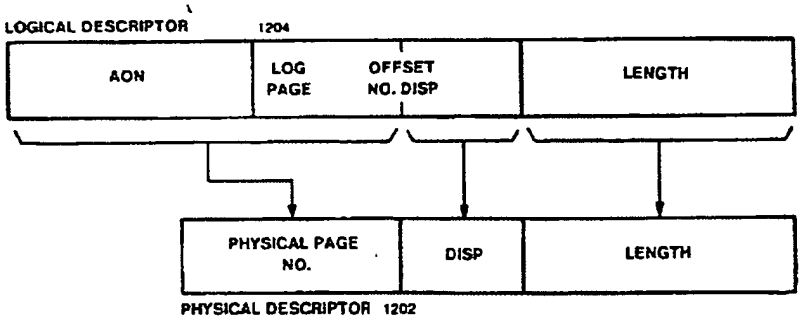


FIG 12

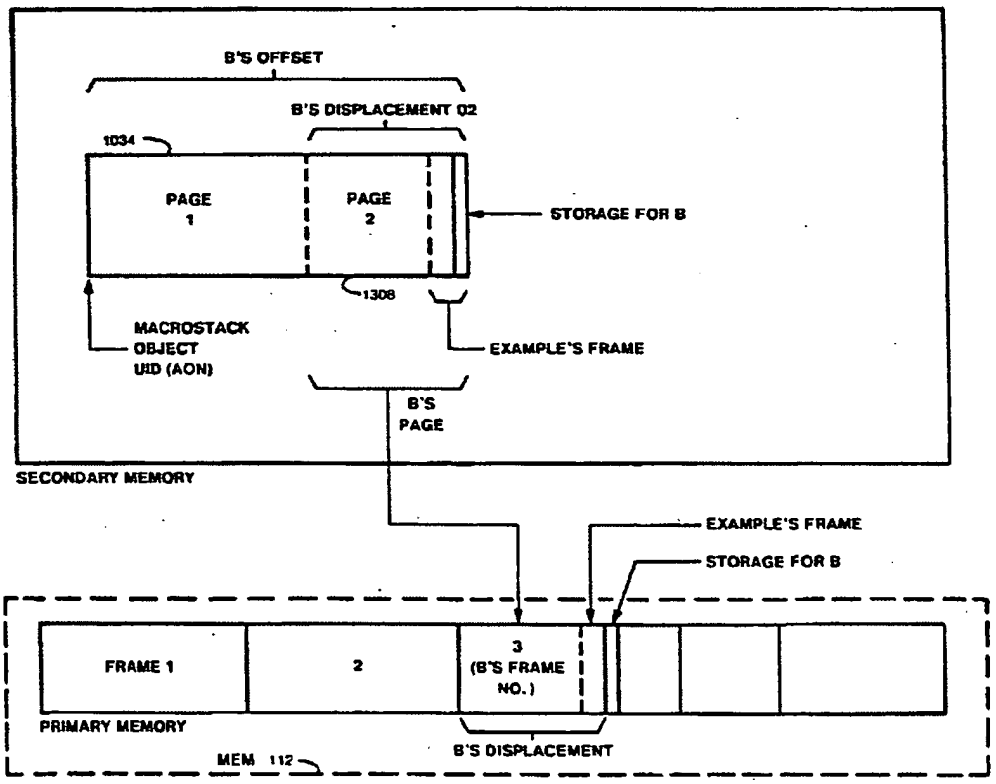


FIG 13

Petitioner Apple Inc. - Exhibit 1004, p. 5354

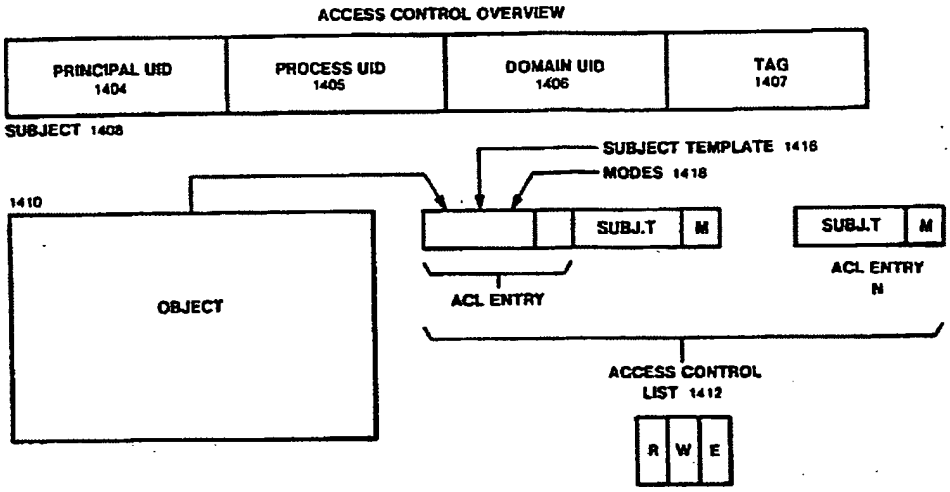


FIG 14

Petitioner Apple Inc. - Exhibit 1004, p. 5355

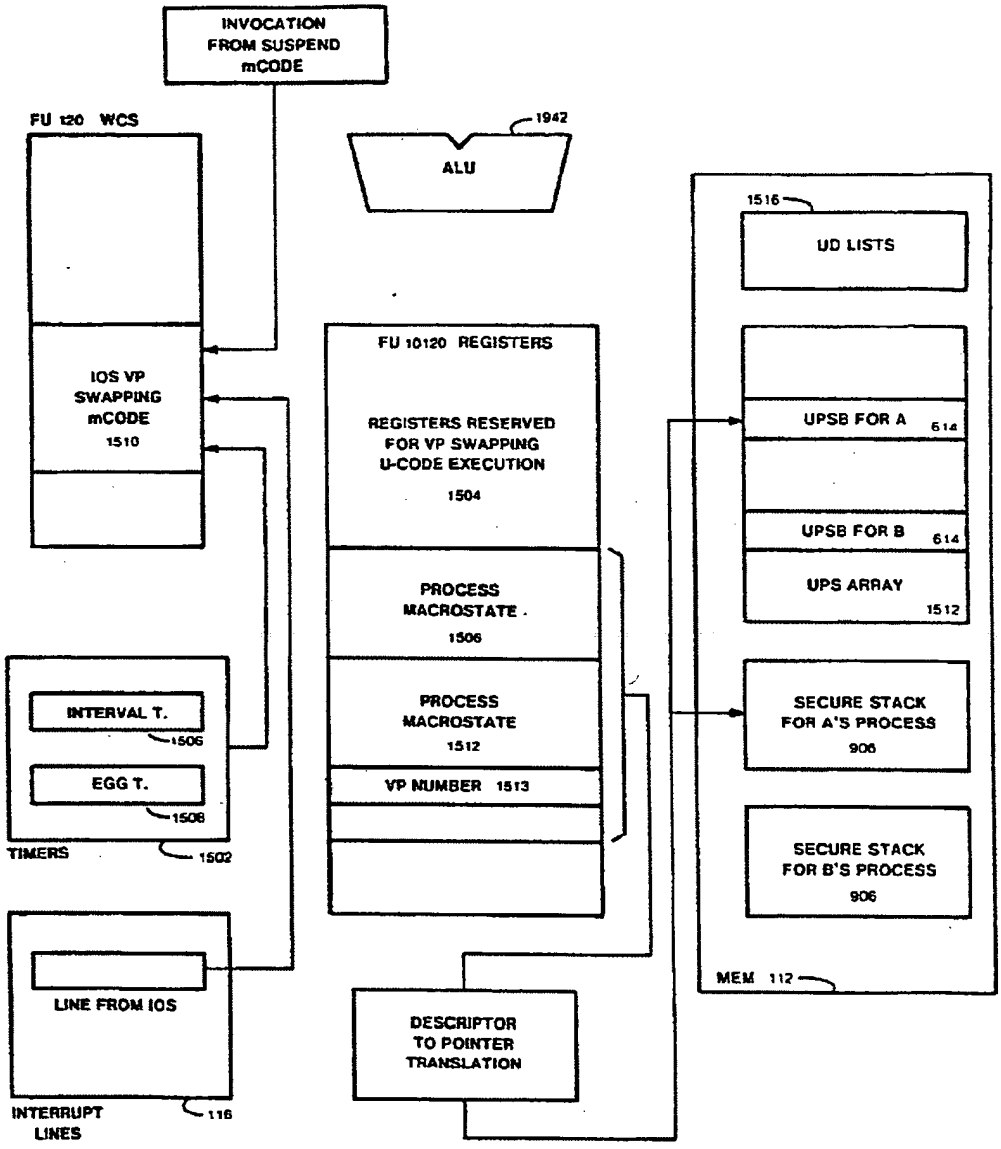


FIG 15

Petitioner Apple Inc. - Exhibit 1004, p. 5356

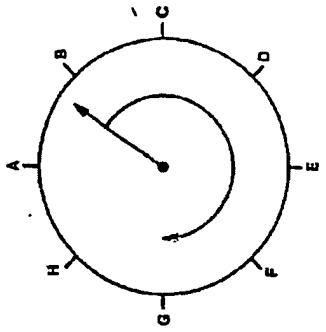


FIG 17

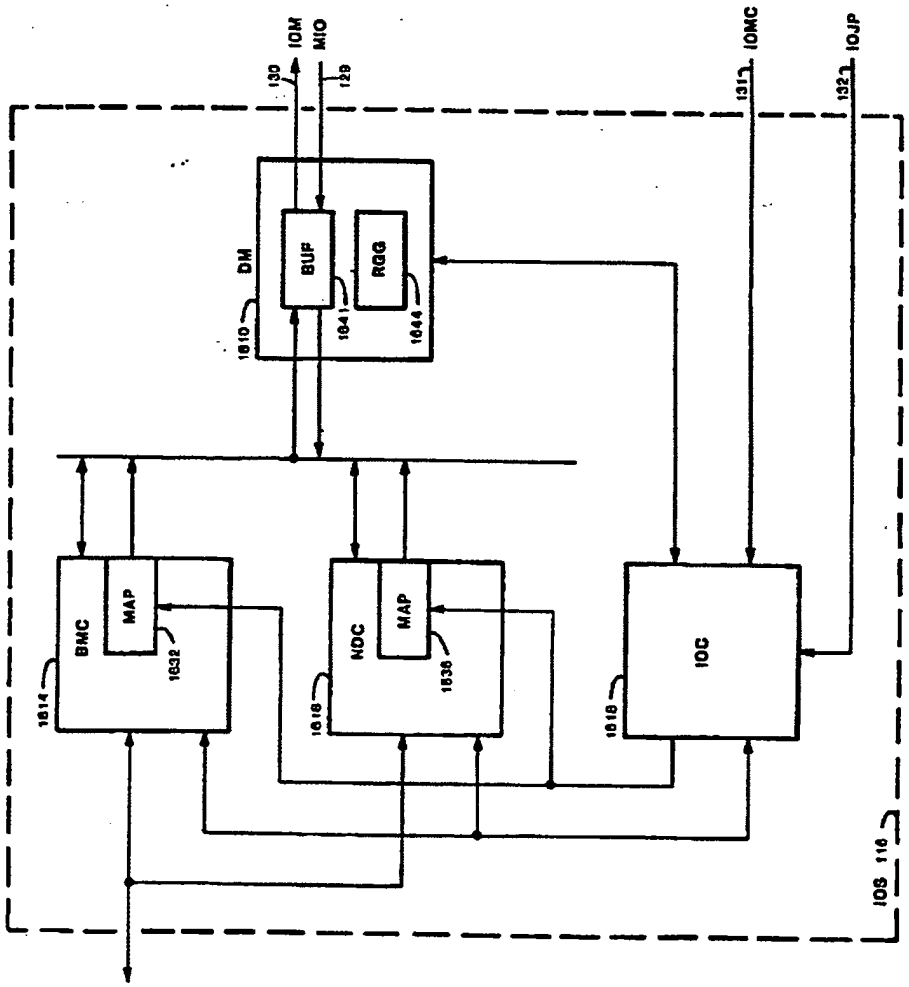


FIG 16

Petitioner Apple Inc. - Exhibit 1004, p. 5357

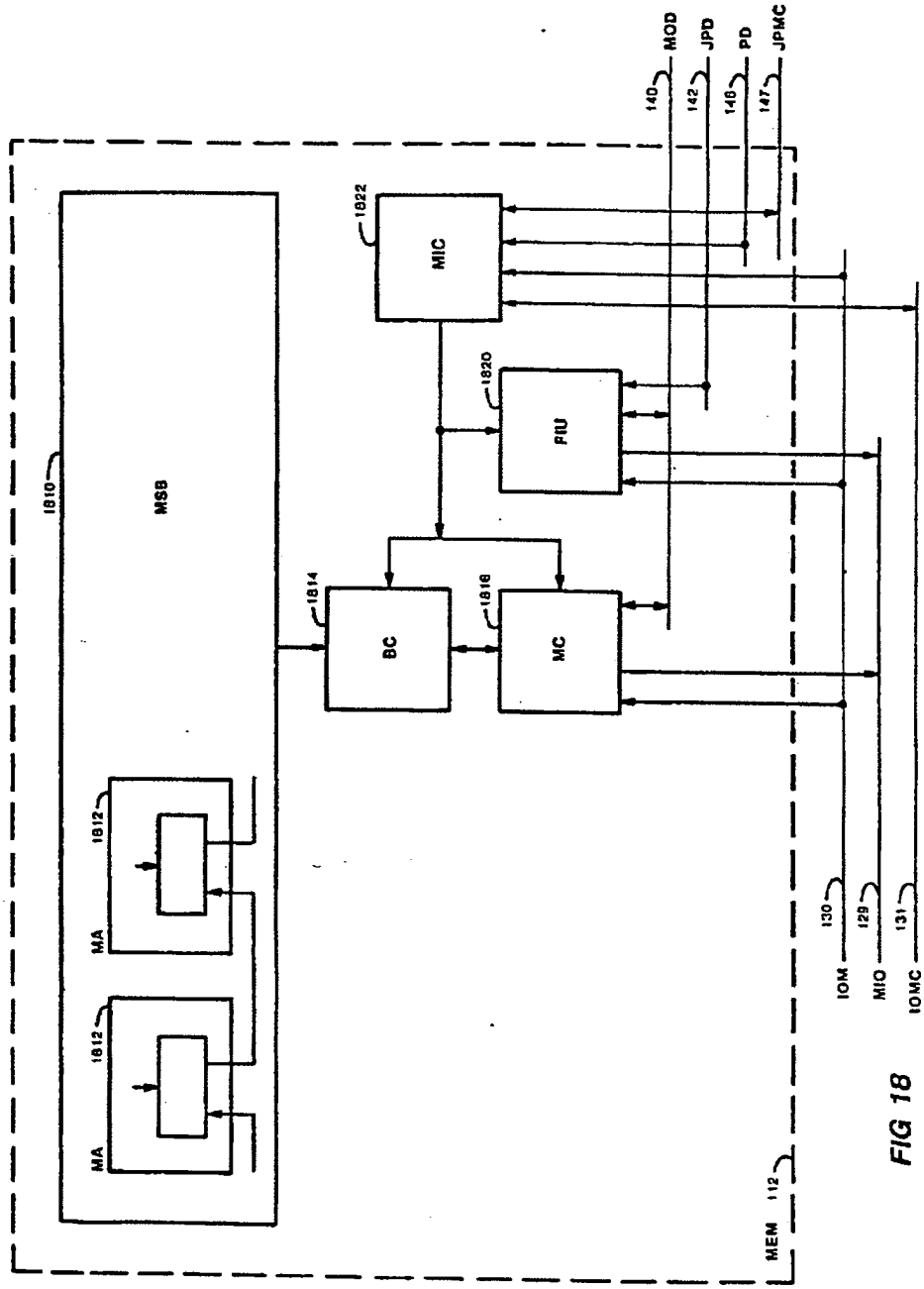


FIG 18

Petitioner Apple Inc. - Exhibit 1004, p. 5358

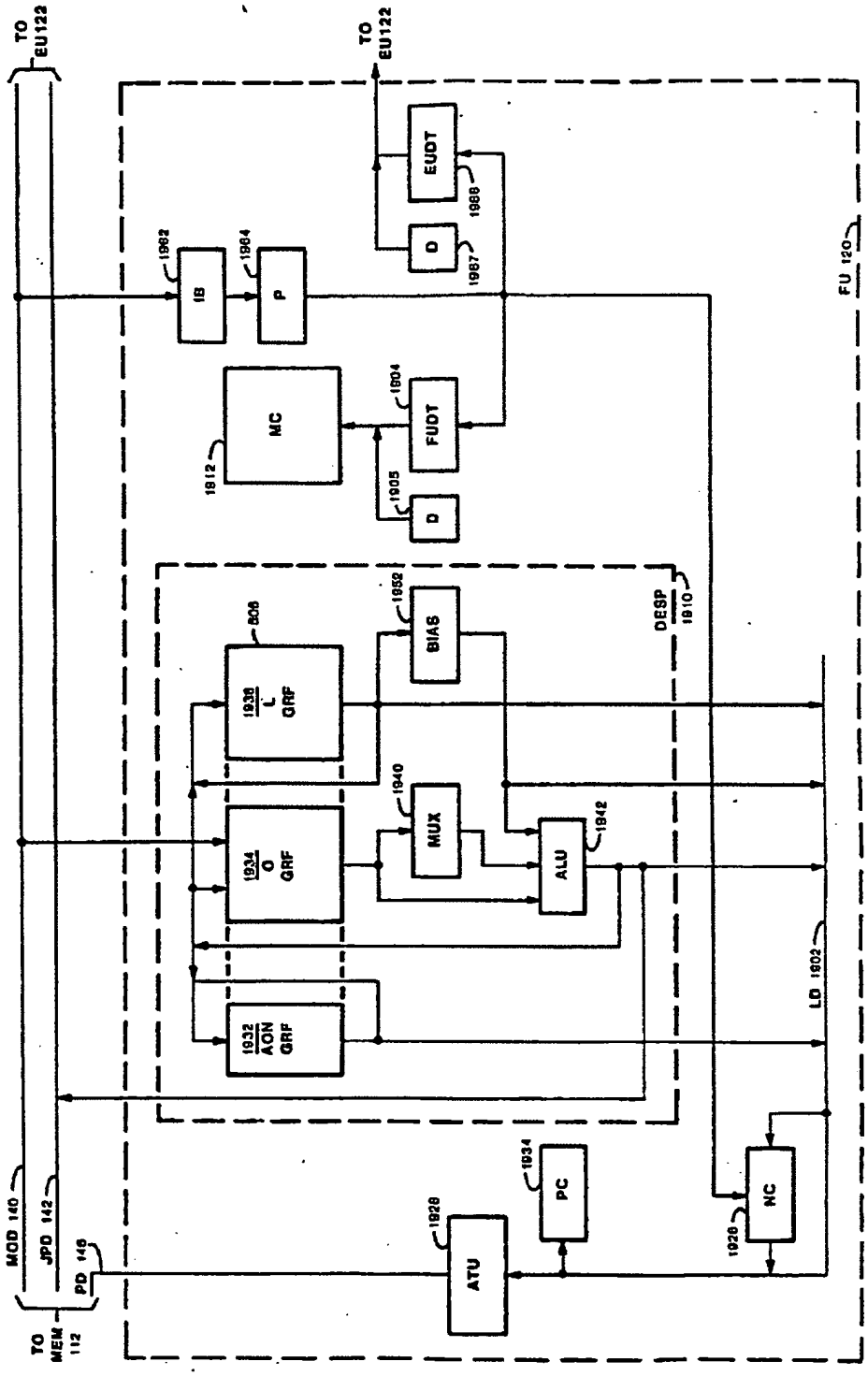


FIG 19

Petitioner Apple Inc. - Exhibit 1004, p. 5359

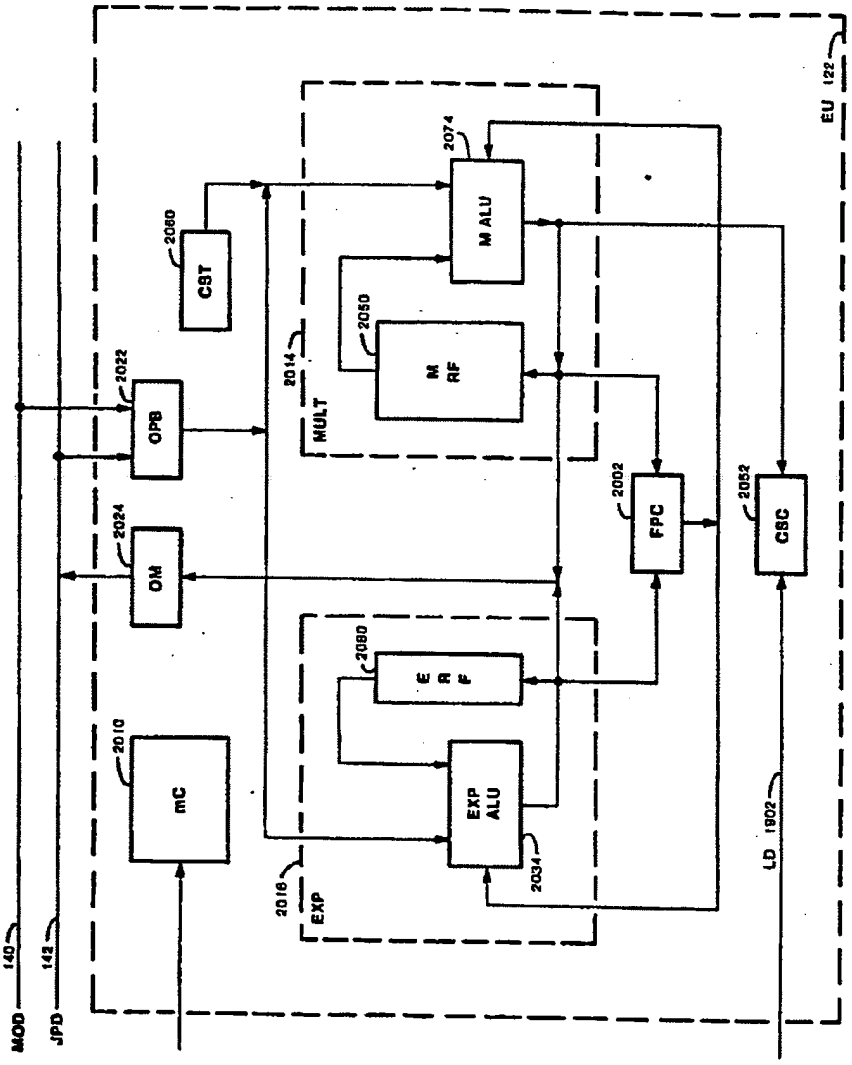


FIG 20

Petitioner Apple Inc. - Exhibit 1004, p. 5360

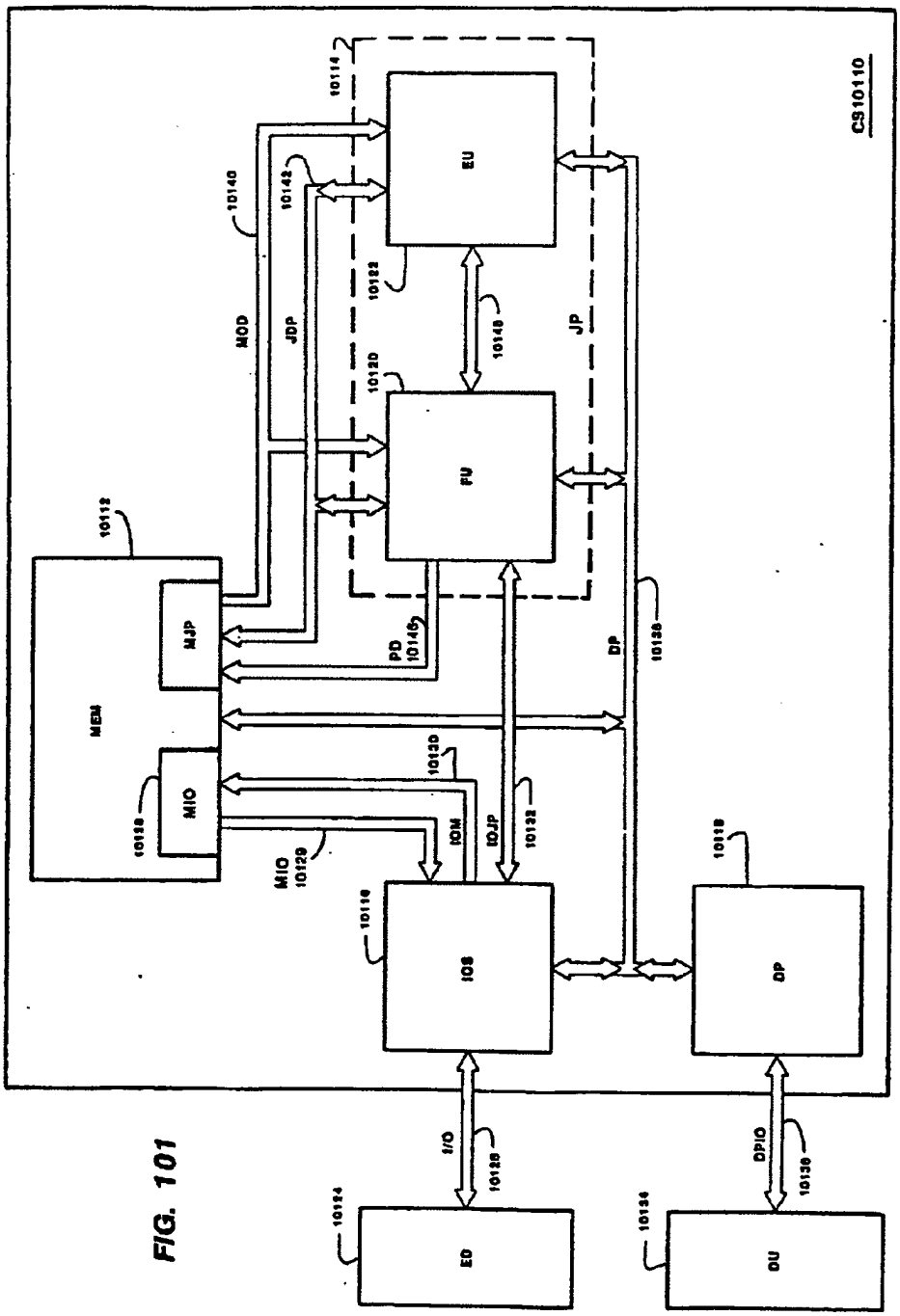


FIG. 101

Petitioner Apple Inc. - Exhibit 1004, p. 5361

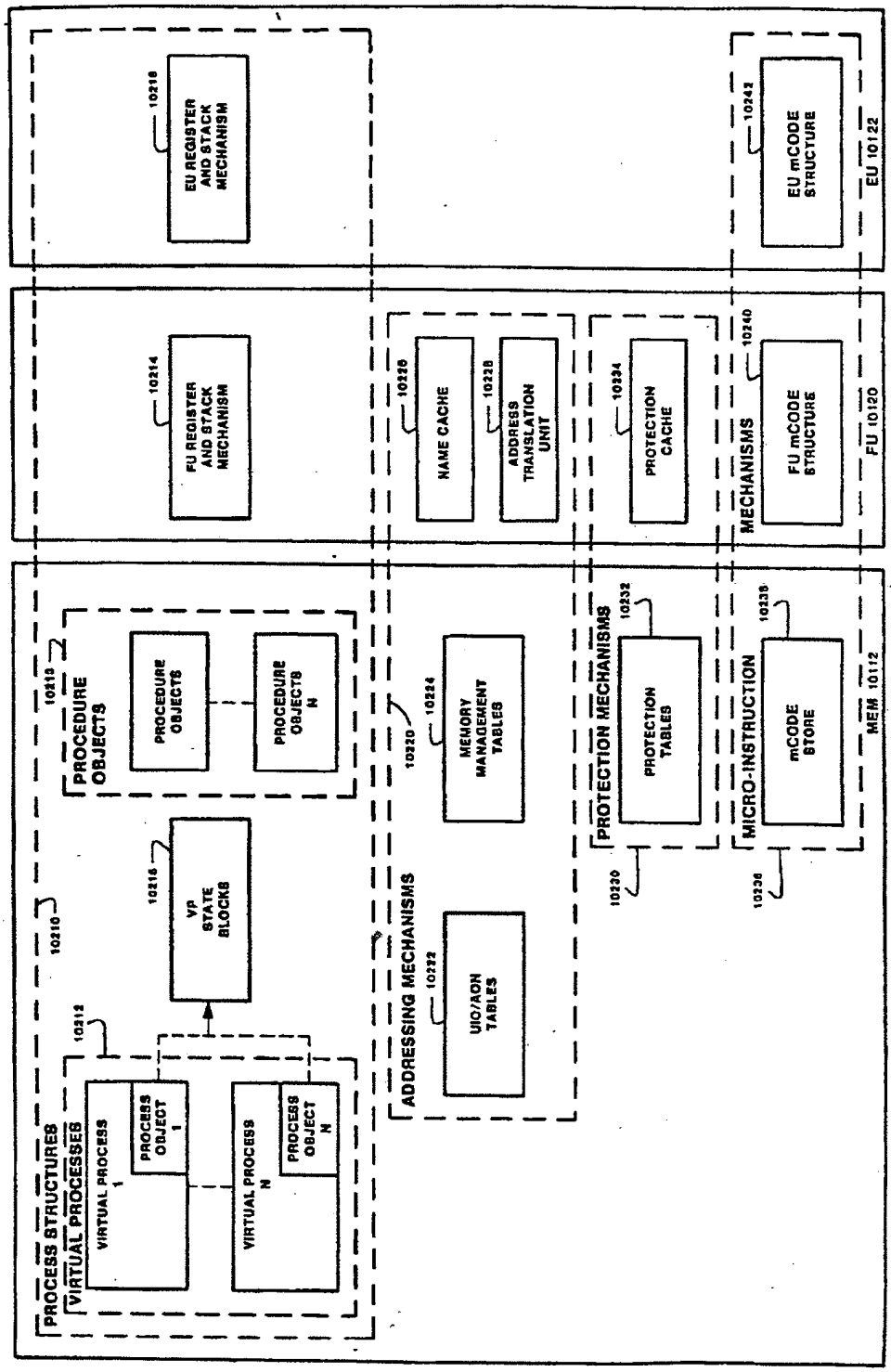


FIG. 102

Petitioner Apple Inc. - Exhibit 1004, p. 5362

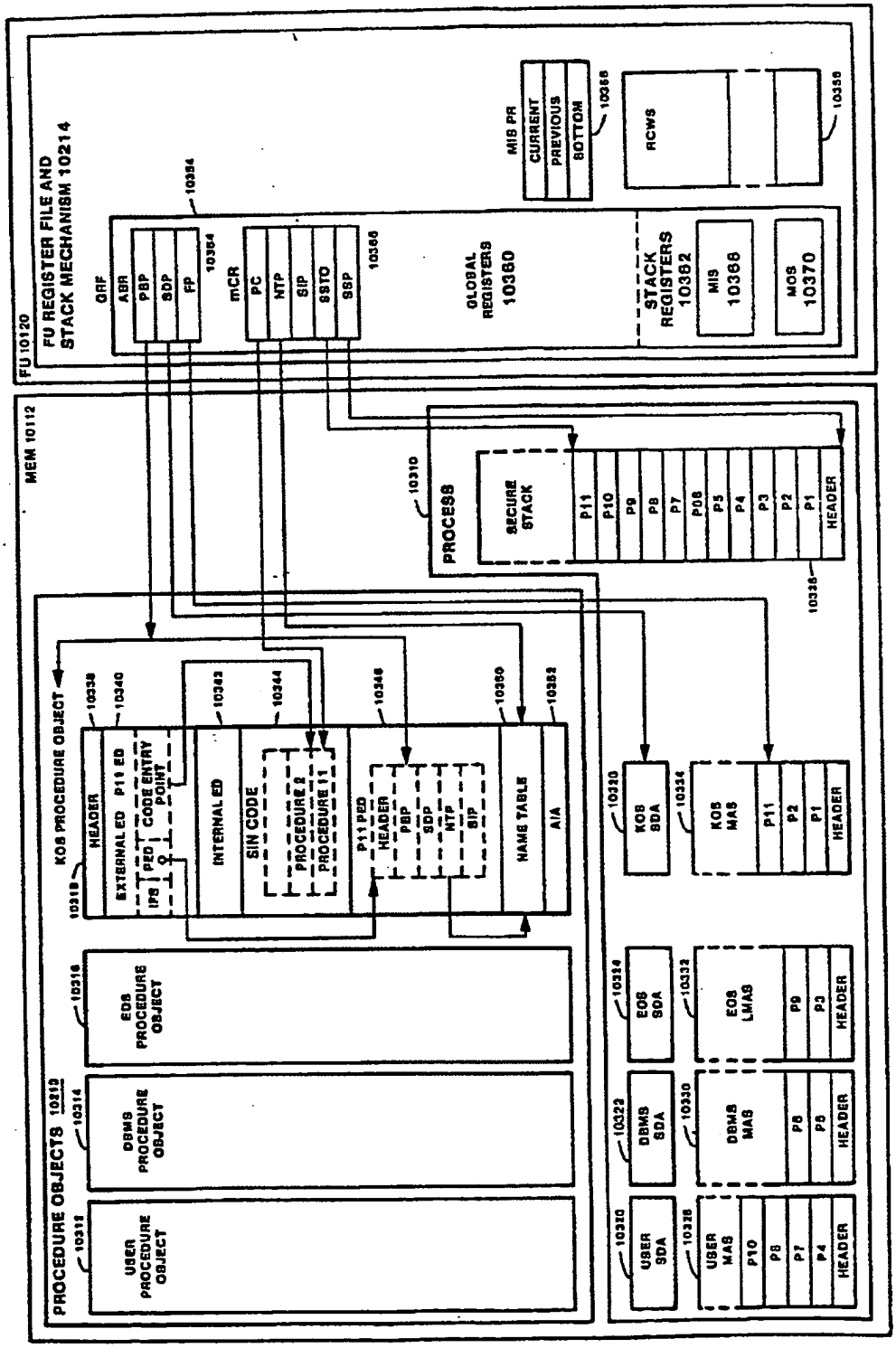


FIG. 103

Petitioner Apple Inc. - Exhibit 1004, p. 5363

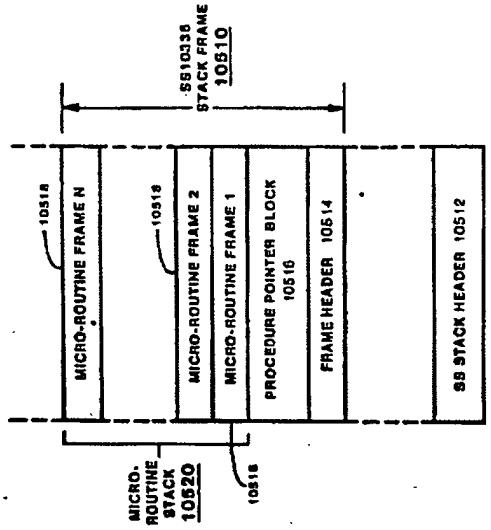


FIG. 105

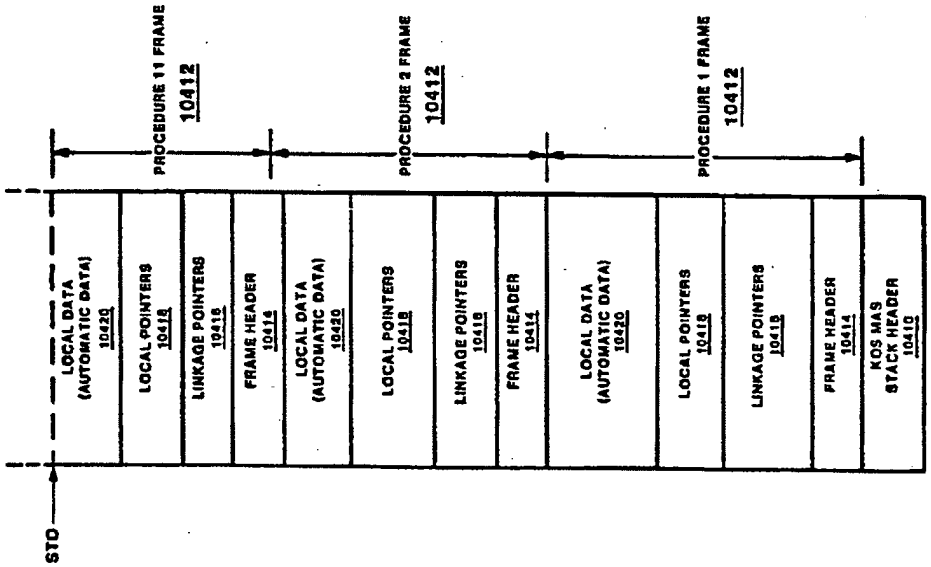


FIG. 104

Petitioner Apple Inc. - Exhibit 1004, p. 5364

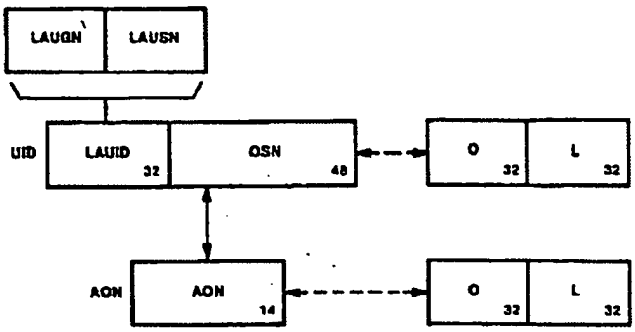


FIG. 106A

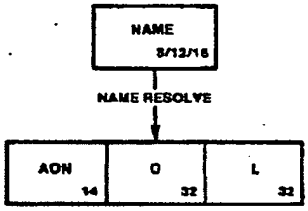


FIG. 106B

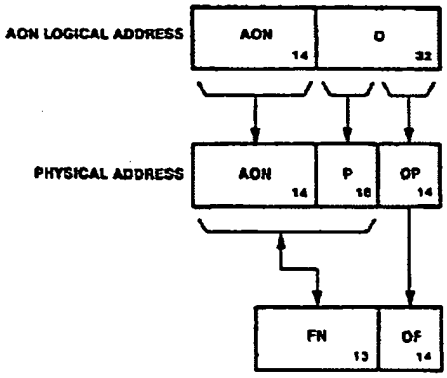


FIG. 106C

Petitioner Apple Inc. - Exhibit 1004, p. 5365

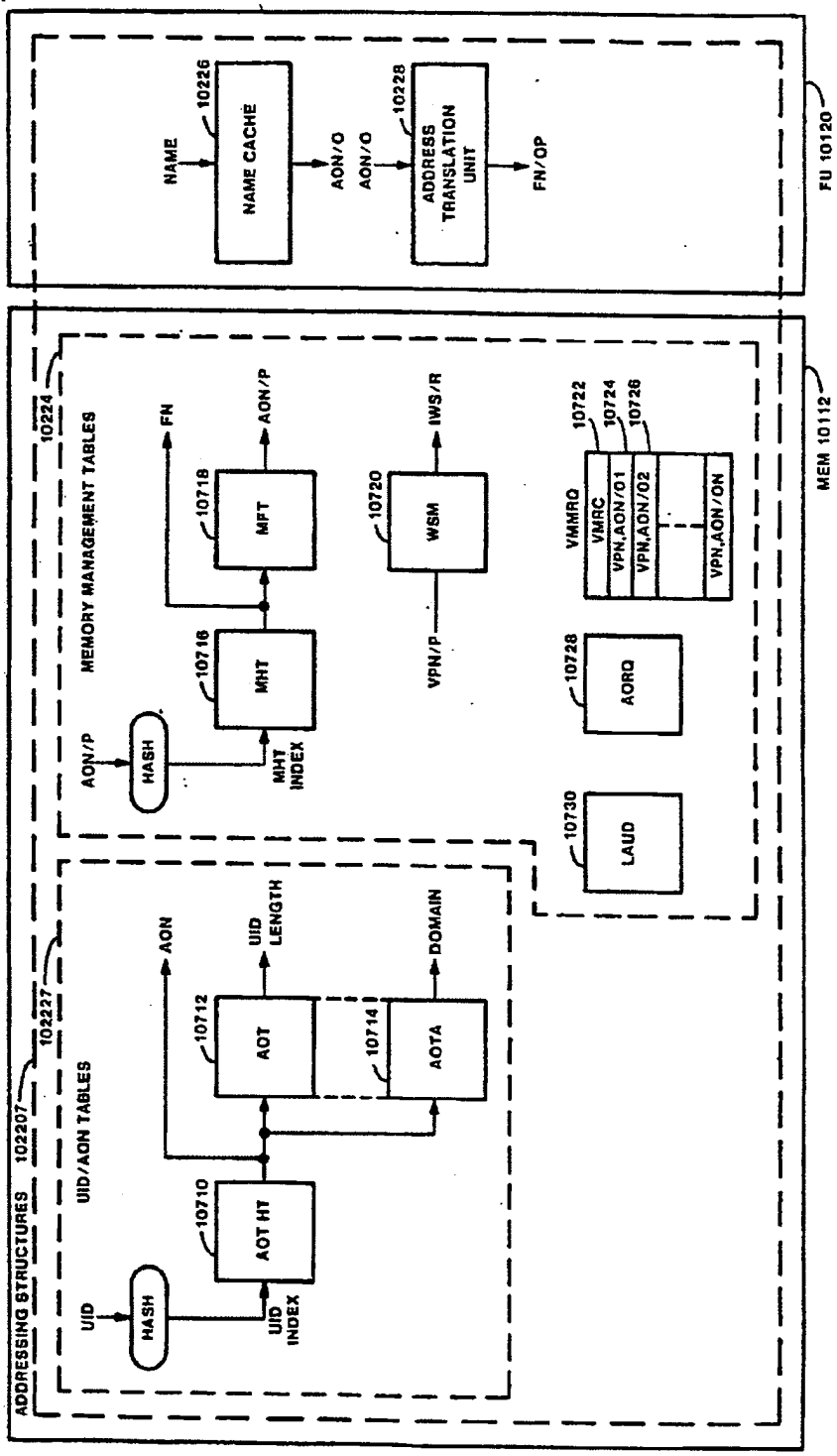


FIG 107

Petitioner Apple Inc. - Exhibit 1004, p. 5366

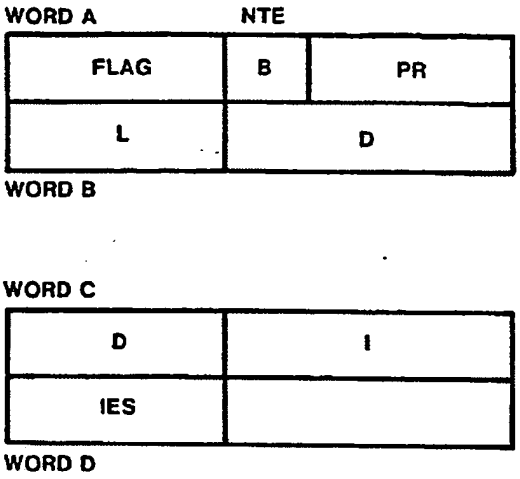


FIG. 108

Petitioner Apple Inc. - Exhibit 1004, p. 5367

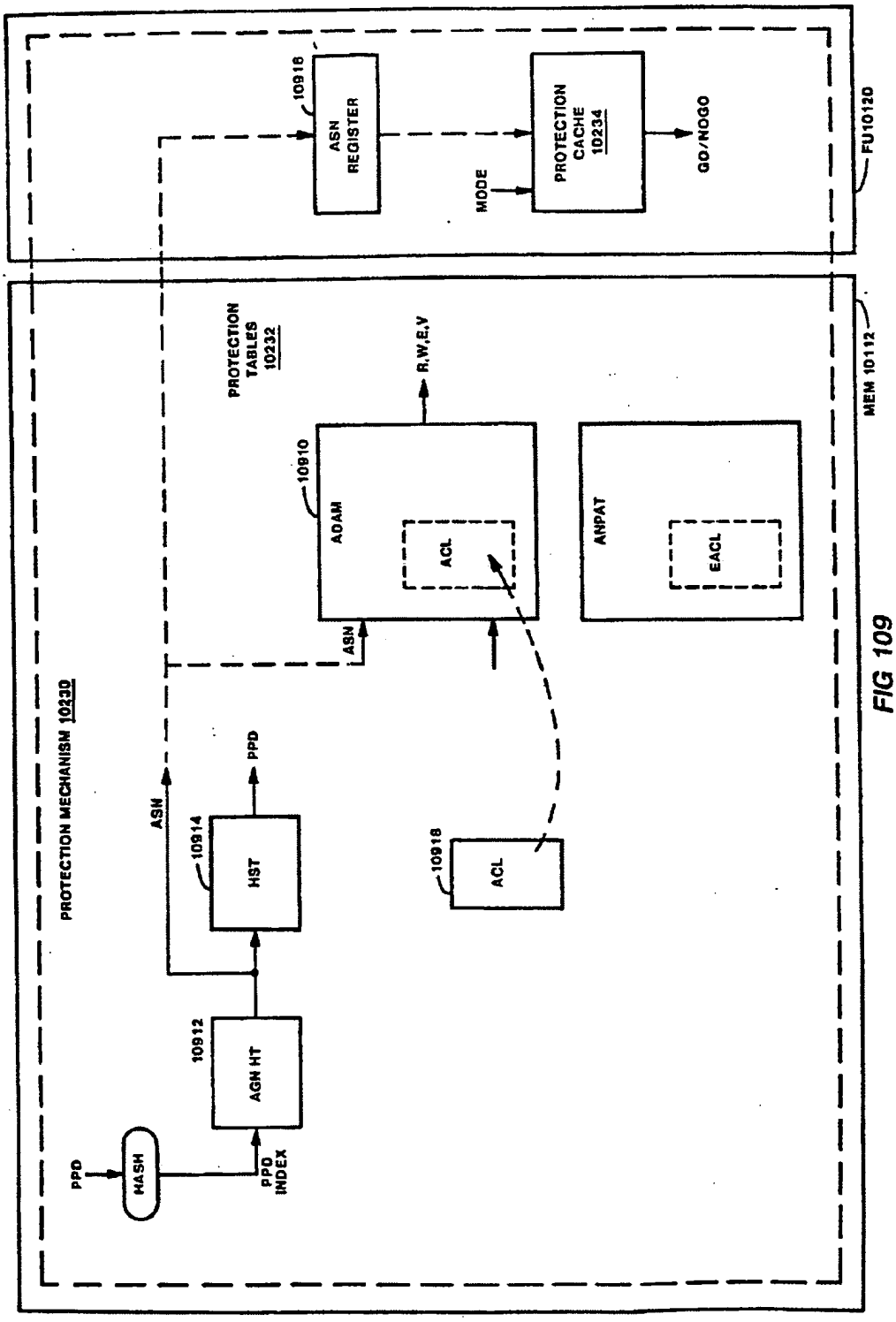


FIG 109

Petitioner Apple Inc. - Exhibit 1004, p. 5368

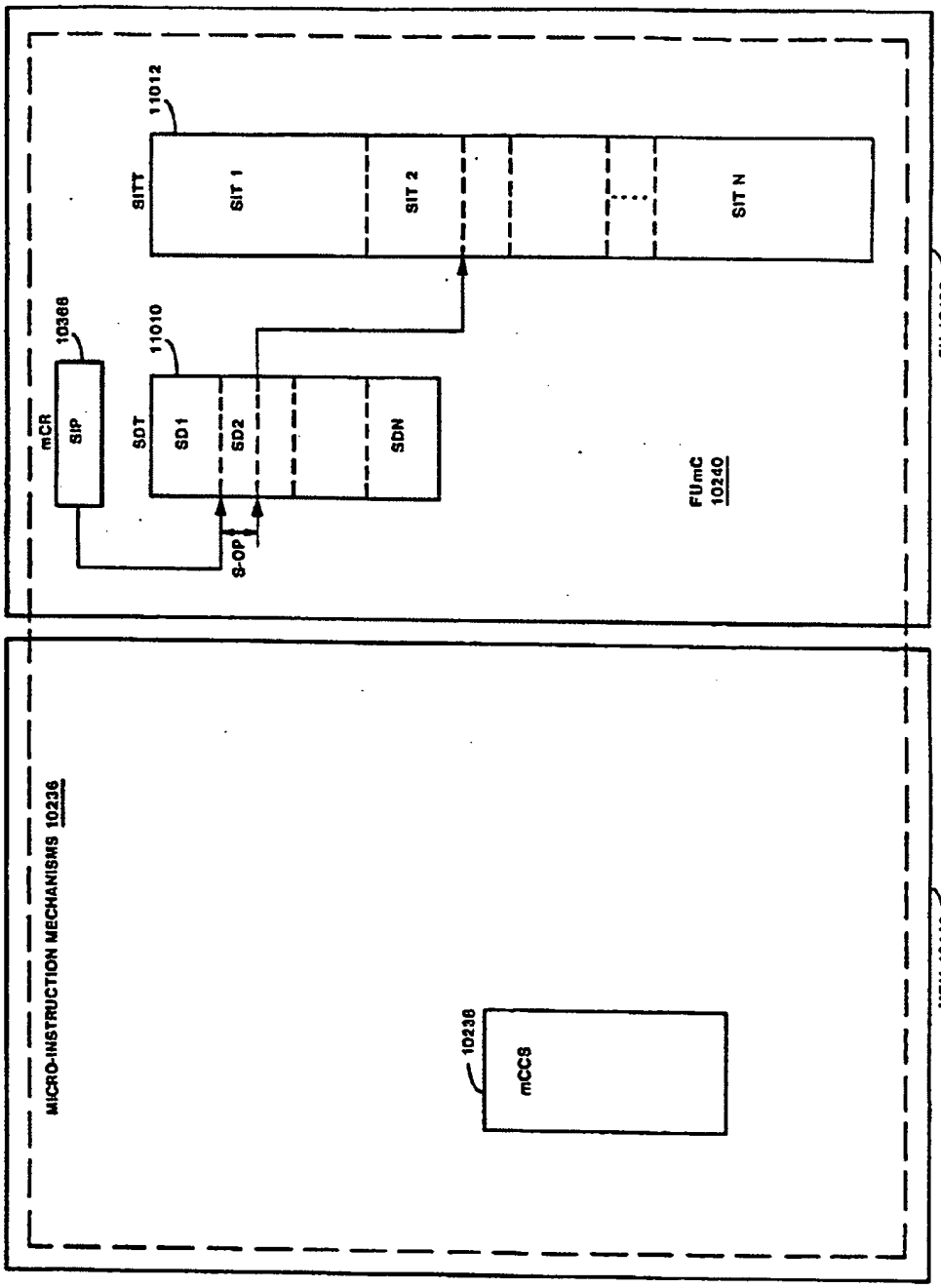


FIG 110

Petitioner Apple Inc. - Exhibit 1004, p. 5369

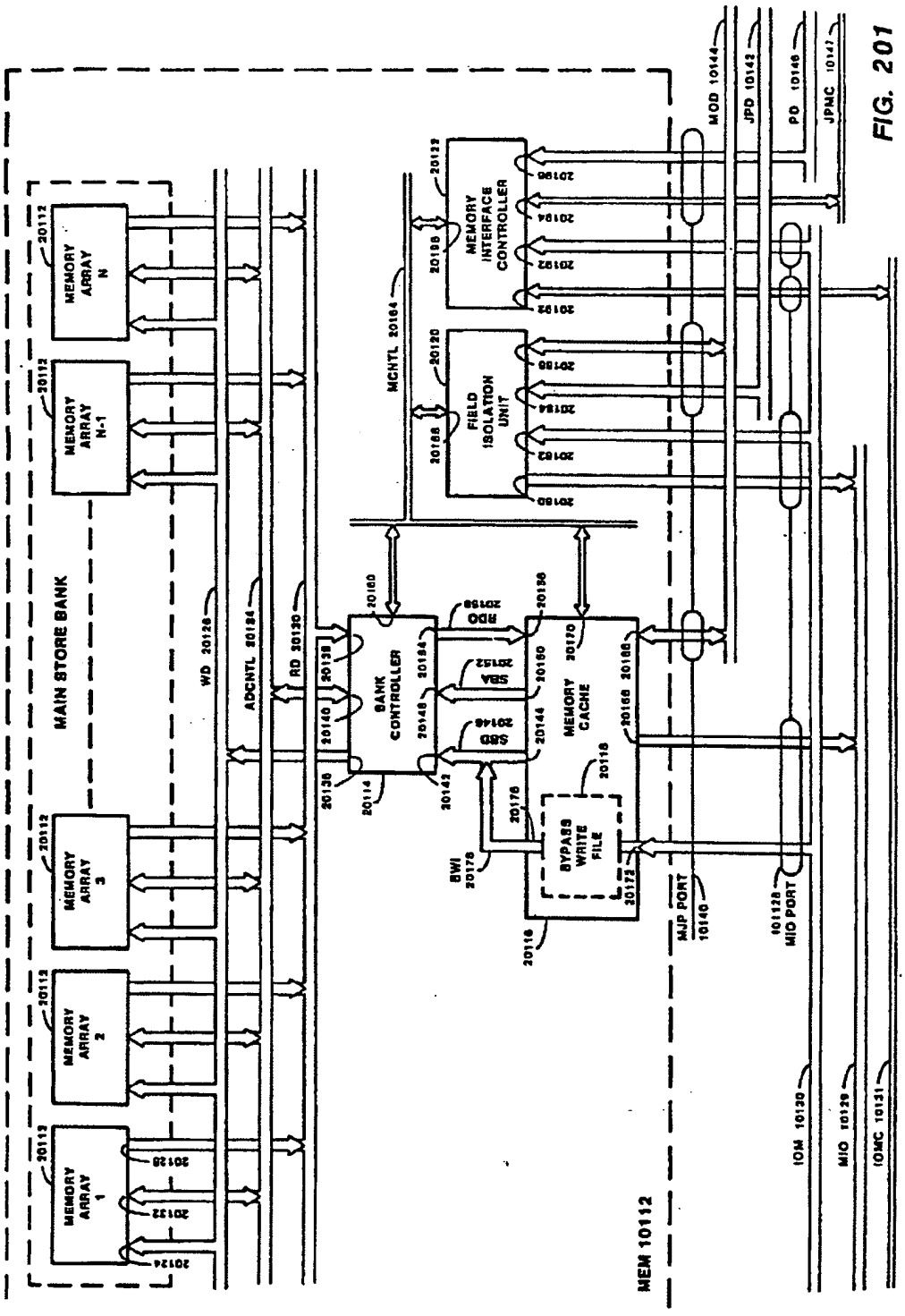


FIG. 201

Petitioner Apple Inc. - Exhibit 1004, p. 5370

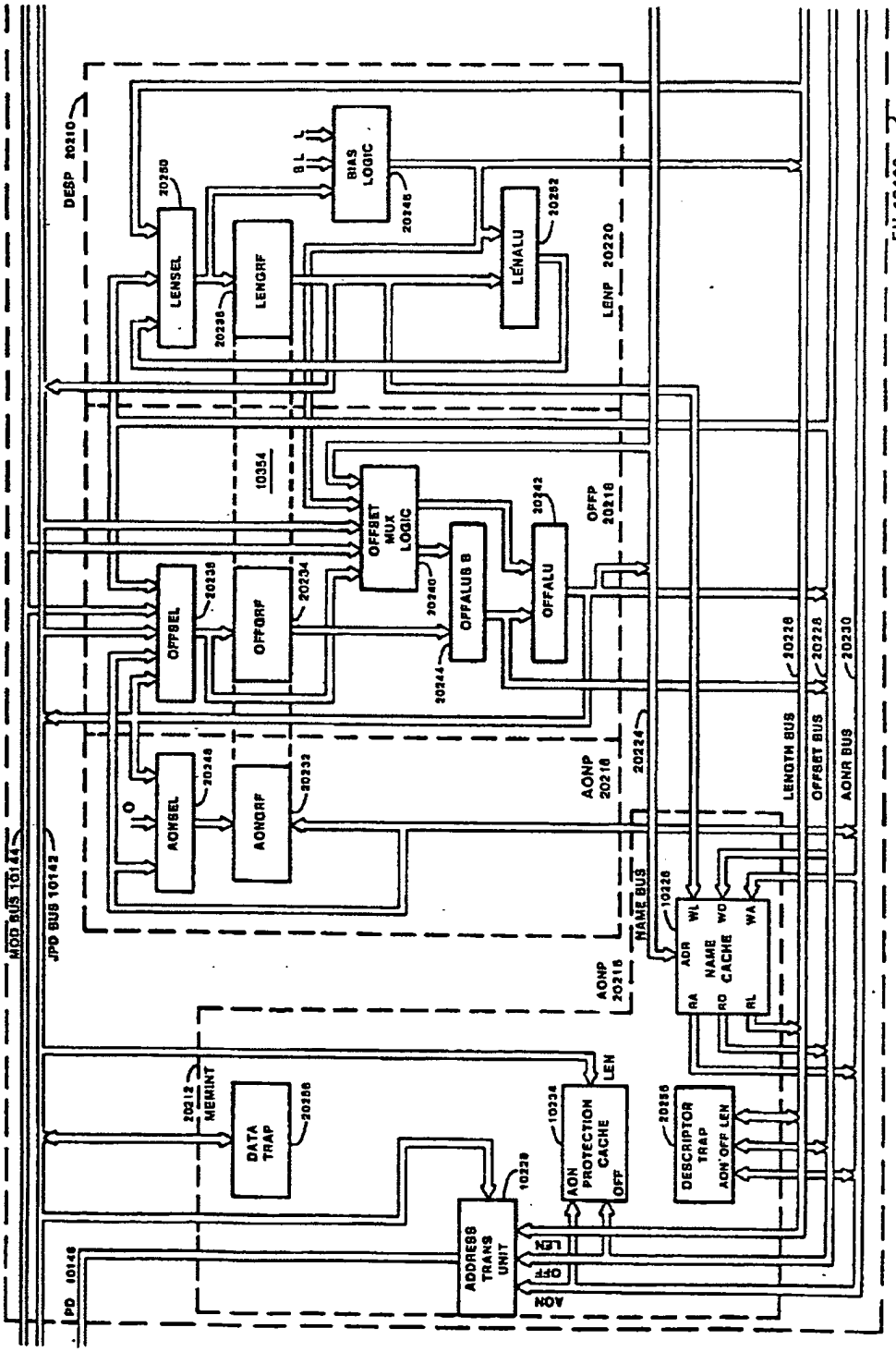


FIG. 202

Petitioner Apple Inc. - Exhibit 1004, p. 5371

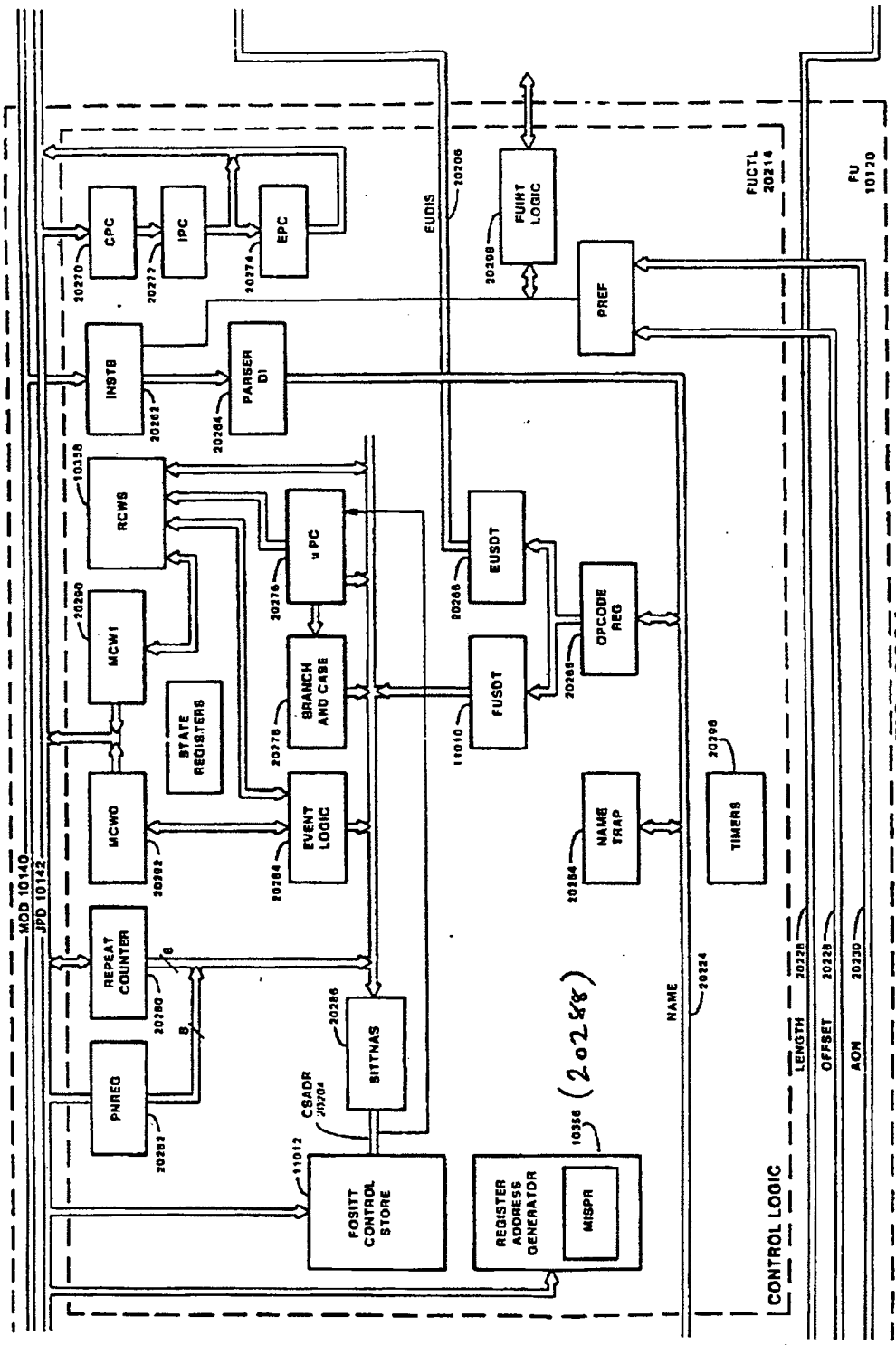


FIG. 202A

Petitioner Apple Inc. - Exhibit 1004, p. 5372

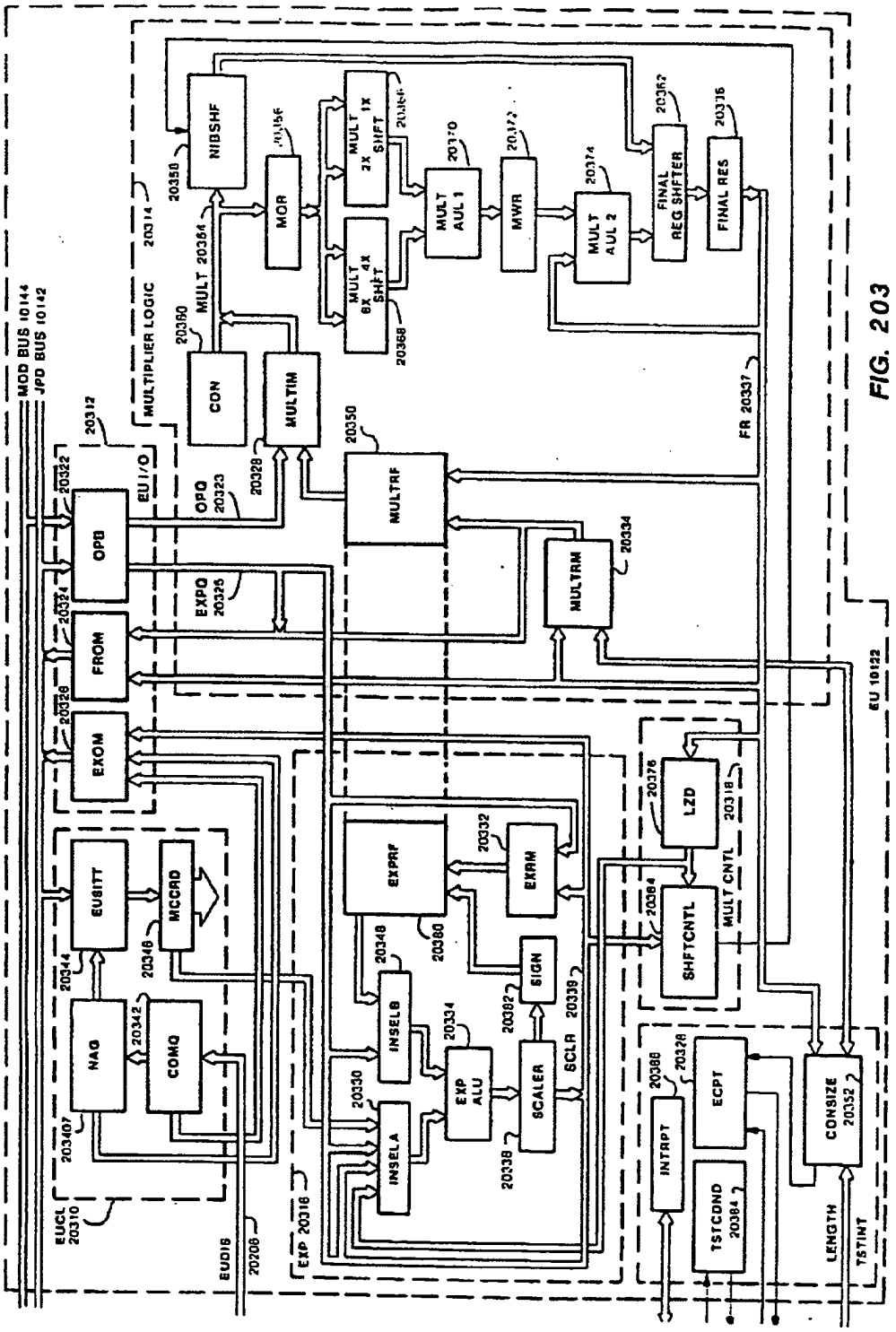


FIG. 203

Petitioner Apple Inc. - Exhibit 1004, p. 5373

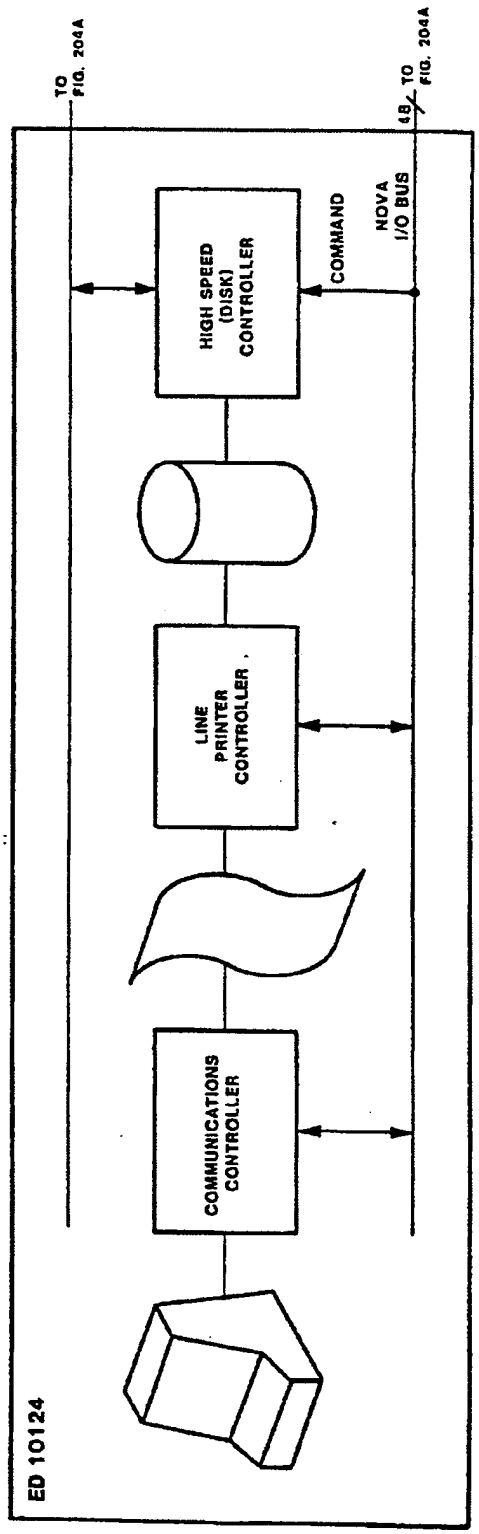


FIG. 204

Petitioner Apple Inc. - Exhibit 1004, p. 5374

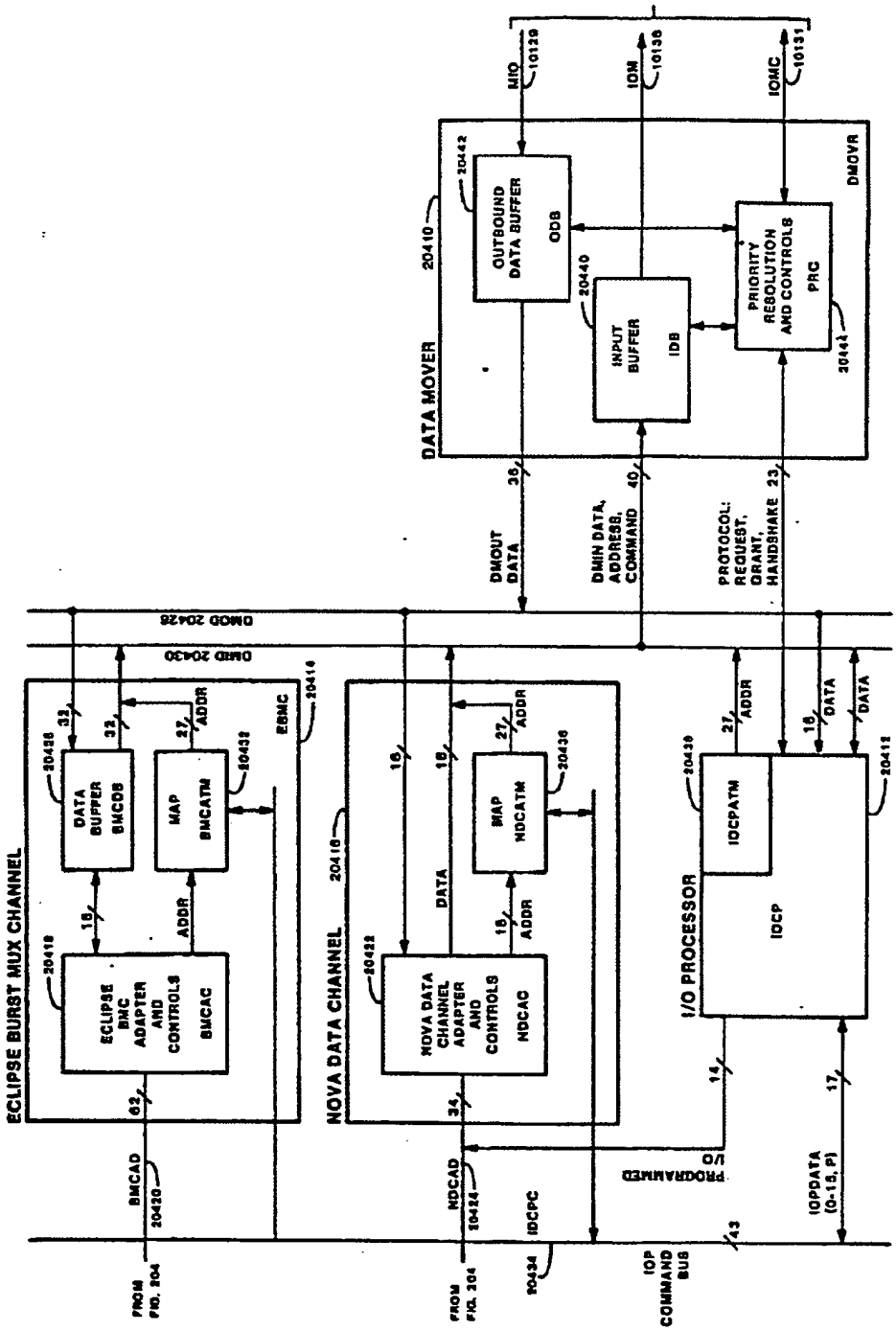


FIG. 204A

Petitioner Apple Inc. - Exhibit 1004, p. 5375

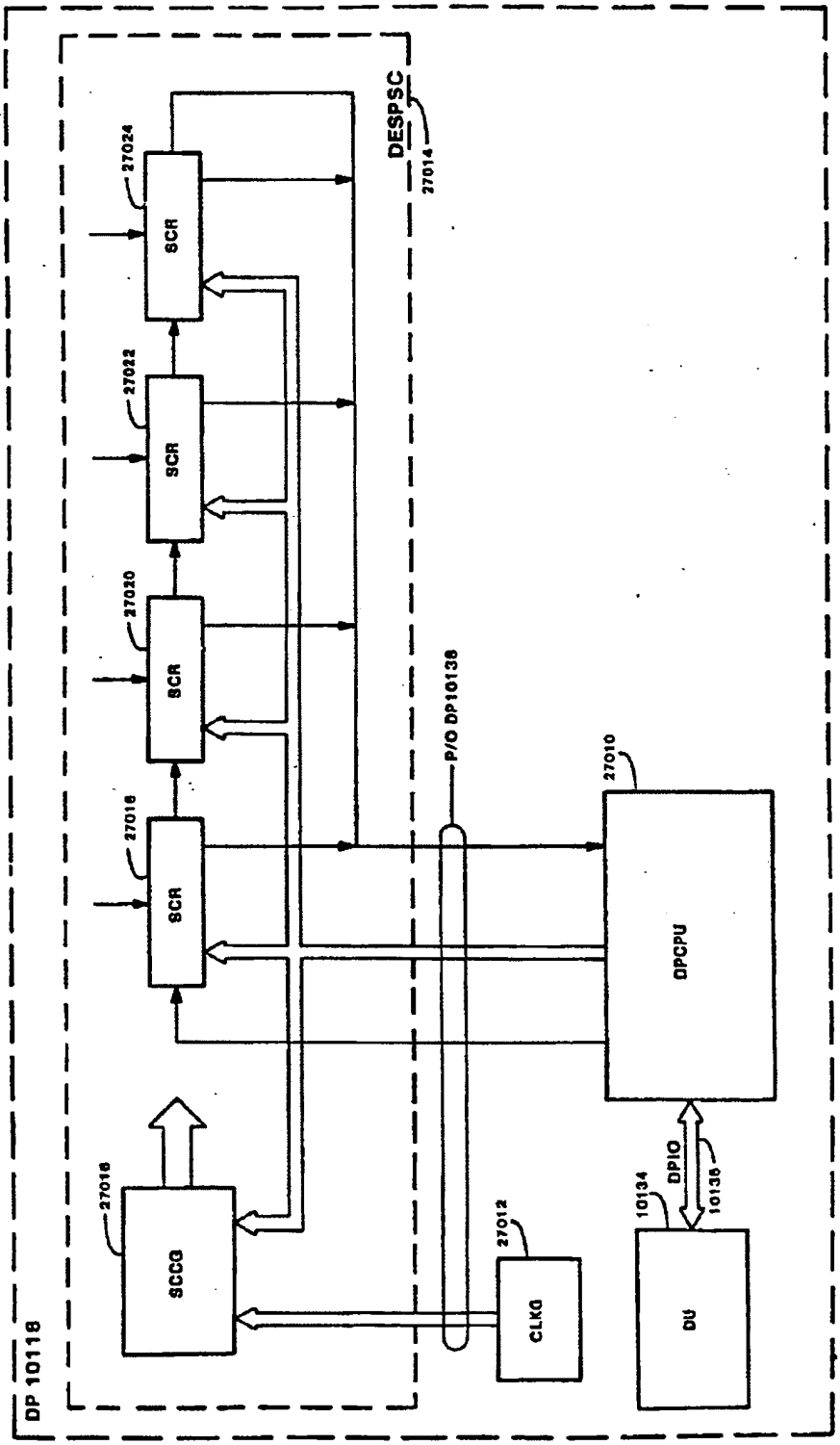


FIG. 205

Petitioner Apple Inc. - Exhibit 1004, p. 5376

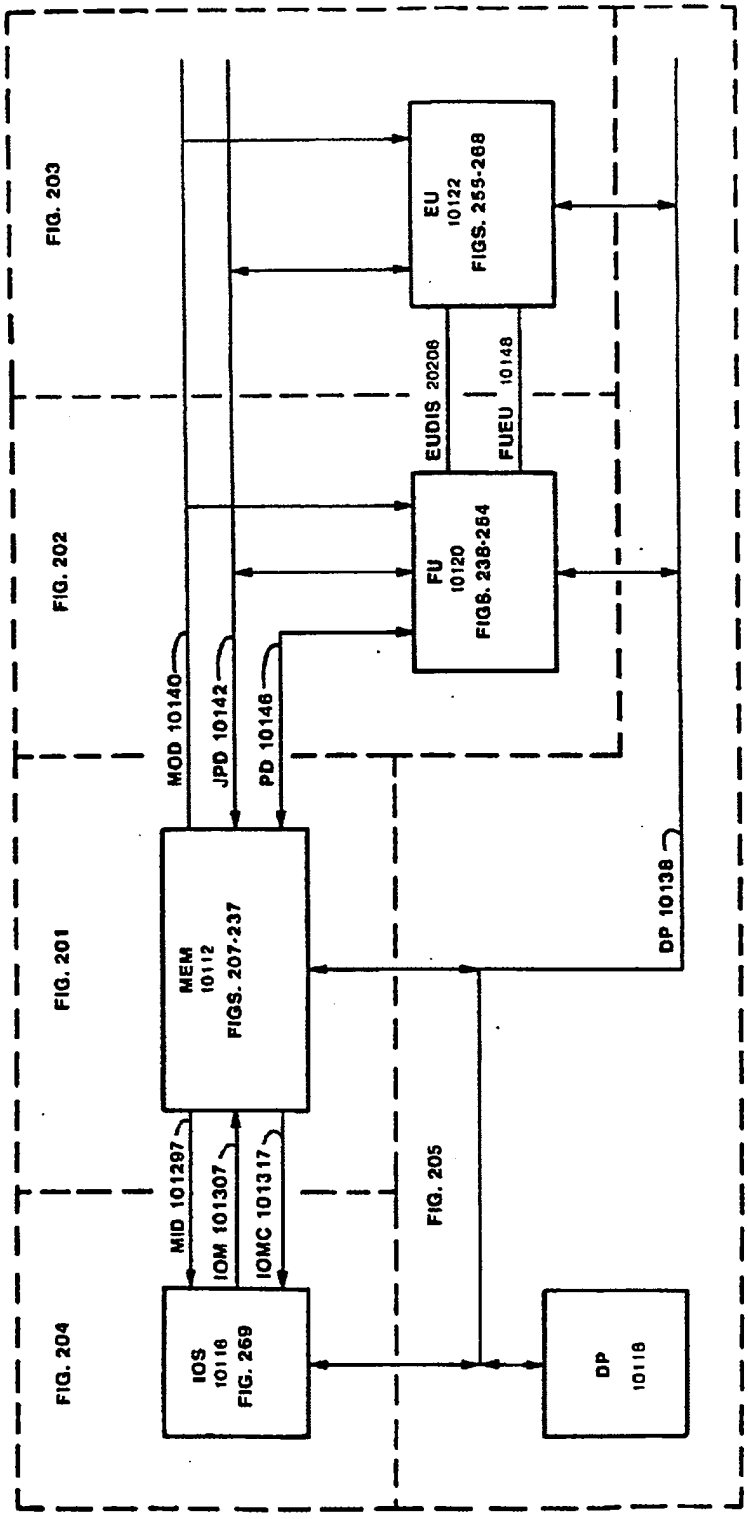


FIG. 206

Petitioner Apple Inc. - Exhibit 1004, p. 5377

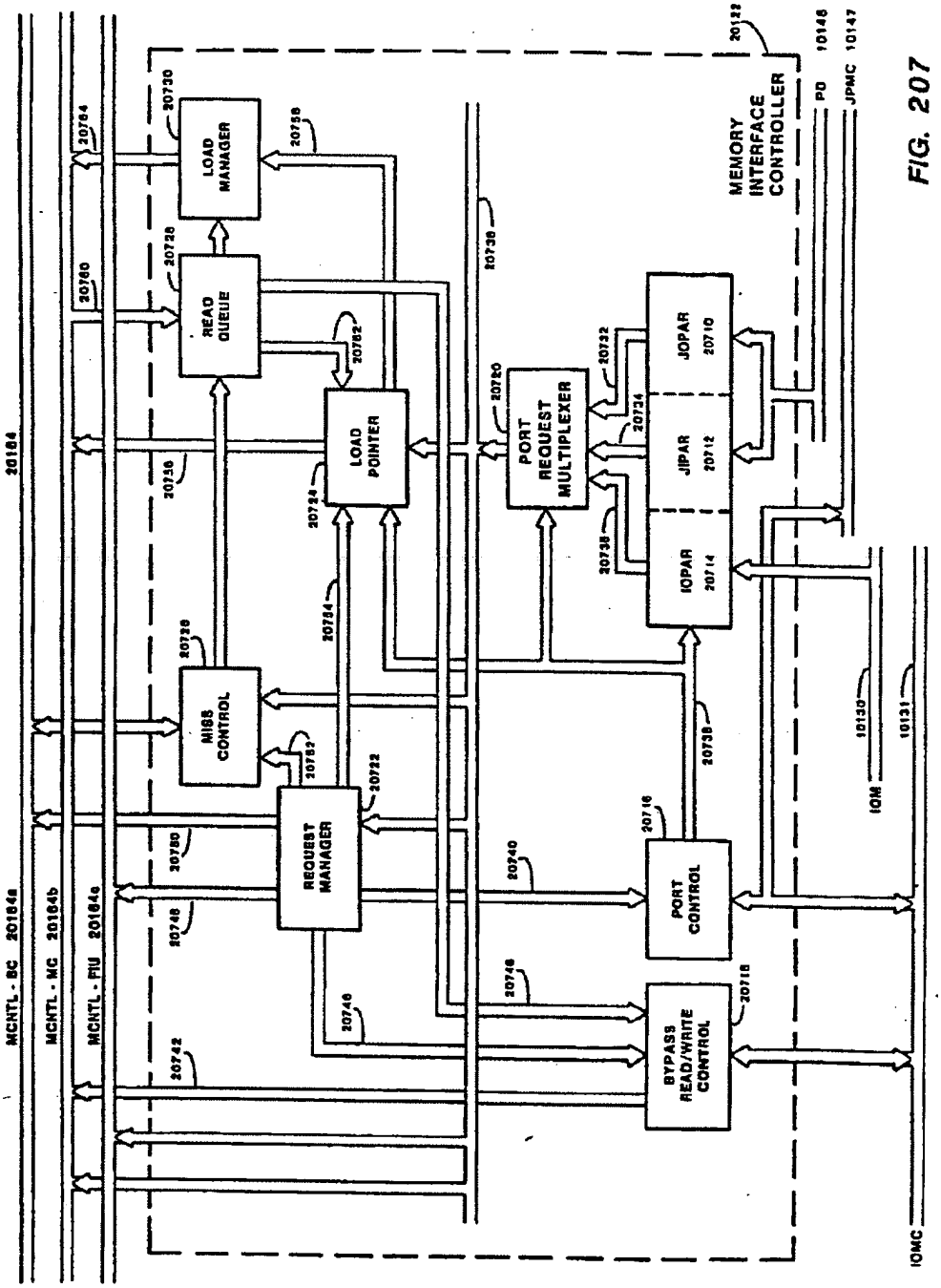


FIG. 207

Petitioner Apple Inc. - Exhibit 1004, p. 5378

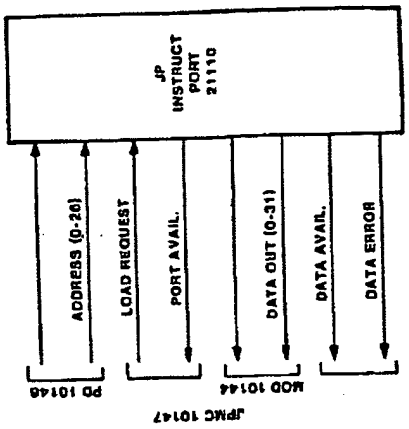


FIG. 211

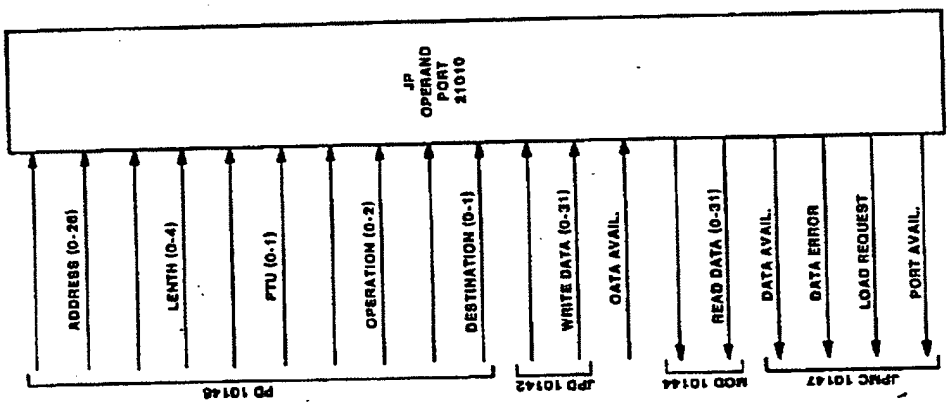


FIG. 210

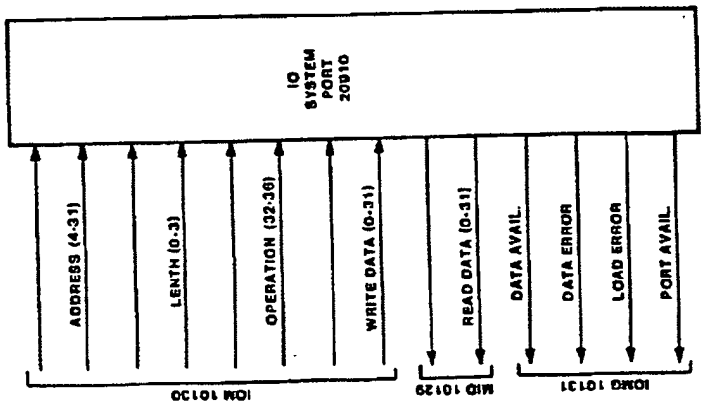


FIG. 209

Petitioner Apple Inc. - Exhibit 1004, p. 5379

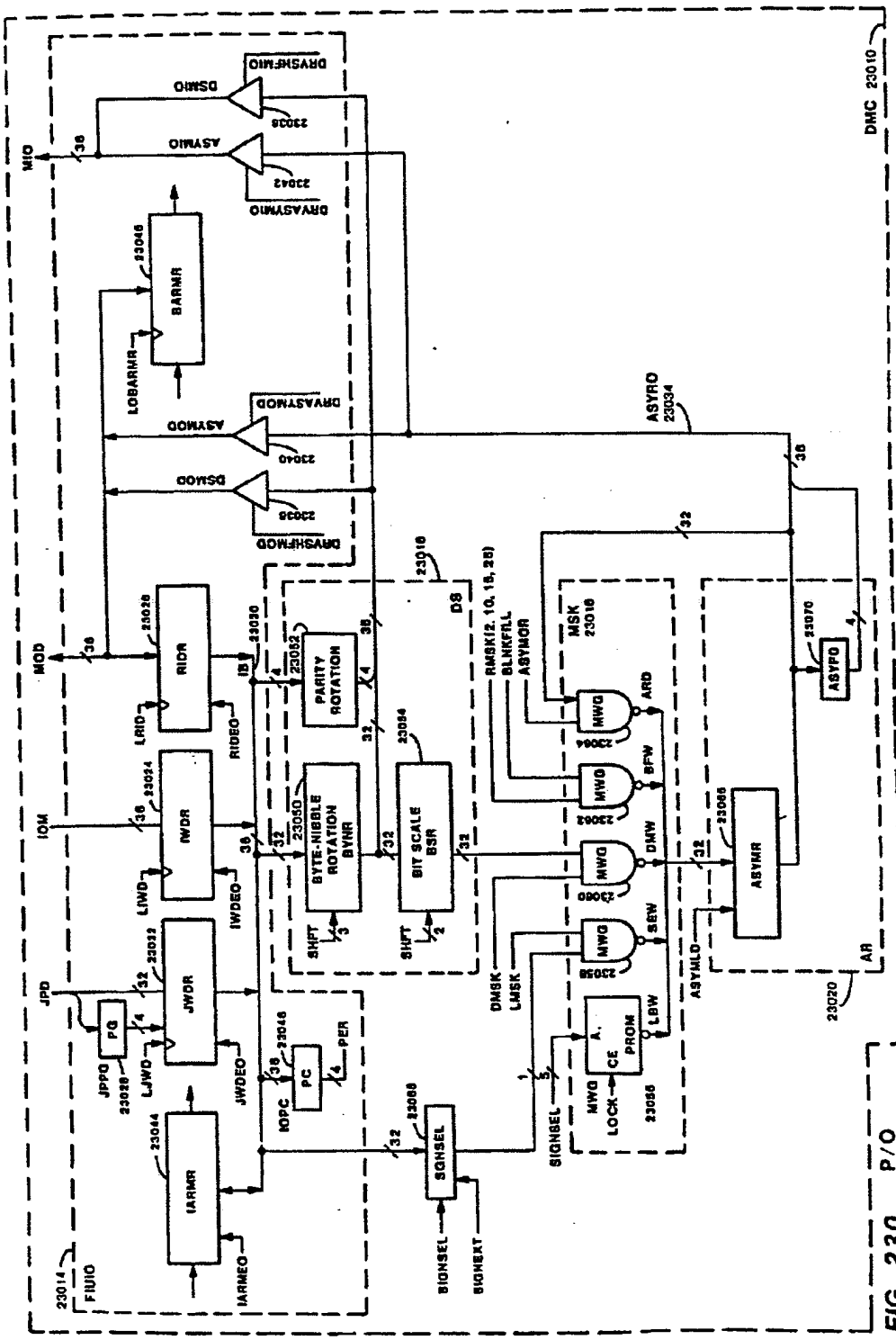


FIG. 230

Petitioner Apple Inc. - Exhibit 1004, p. 5380

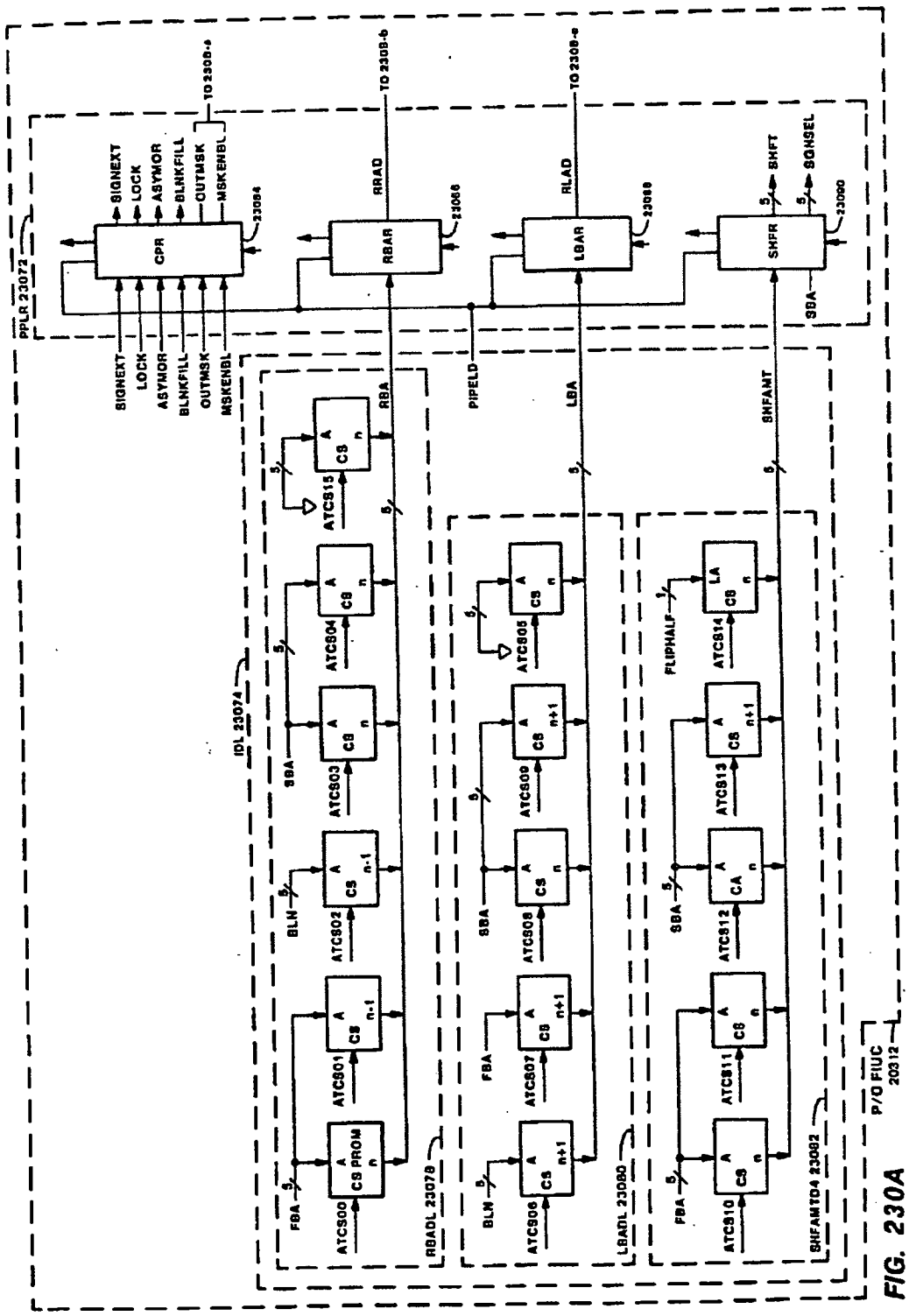


FIG. 230A

Petitioner Apple Inc. - Exhibit 1004, p. 5381

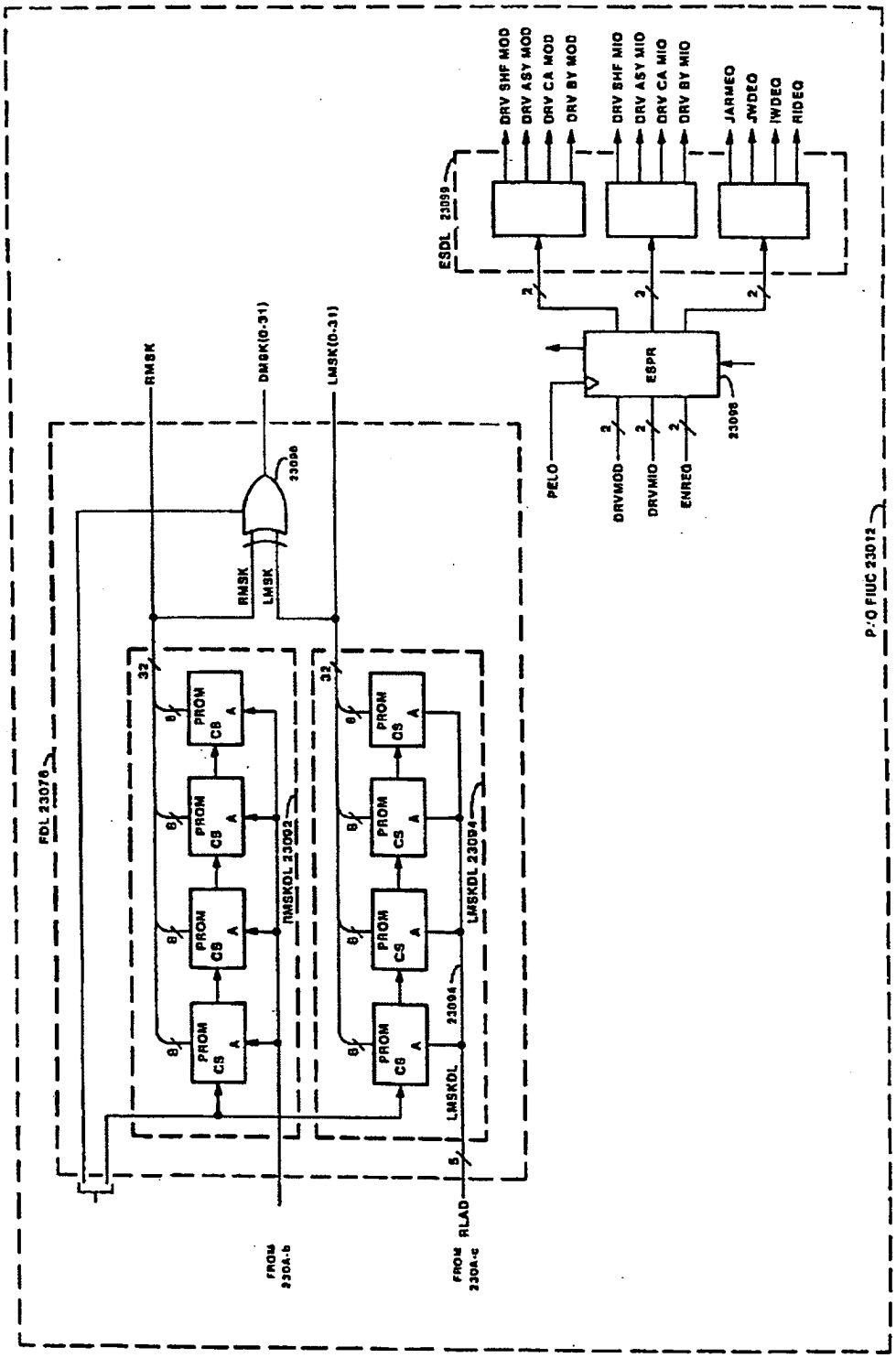


FIG 230B

Petitioner Apple Inc. - Exhibit 1004, p. 5382

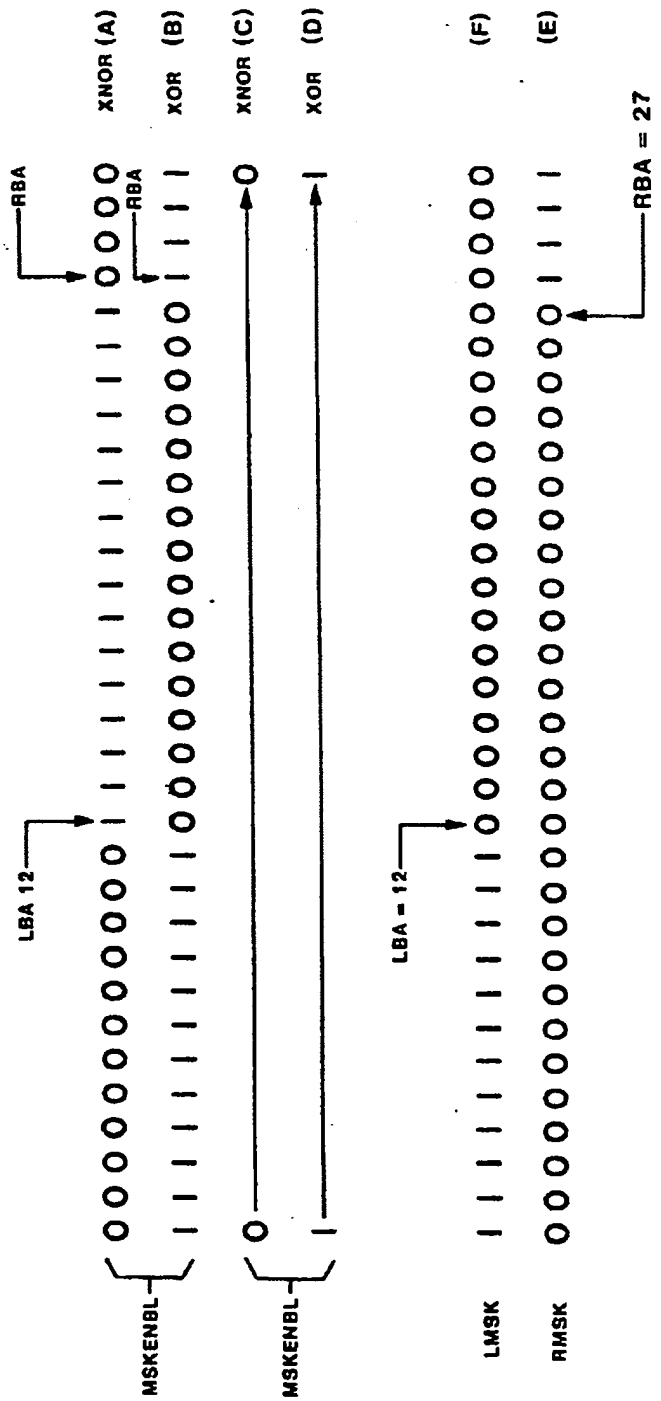


FIG. 231

Petitioner Apple Inc. - Exhibit 1004, p. 5383

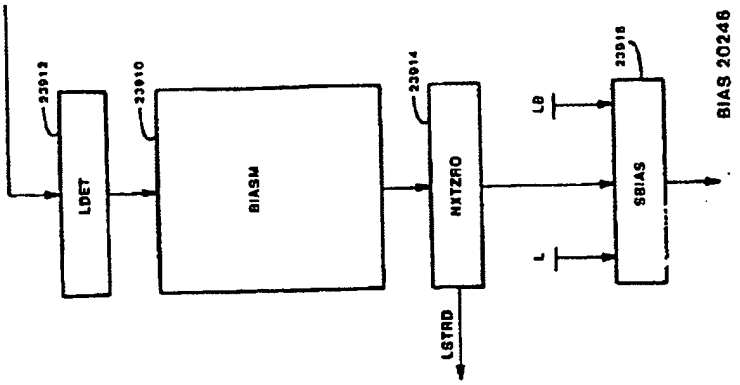


FIG. 239

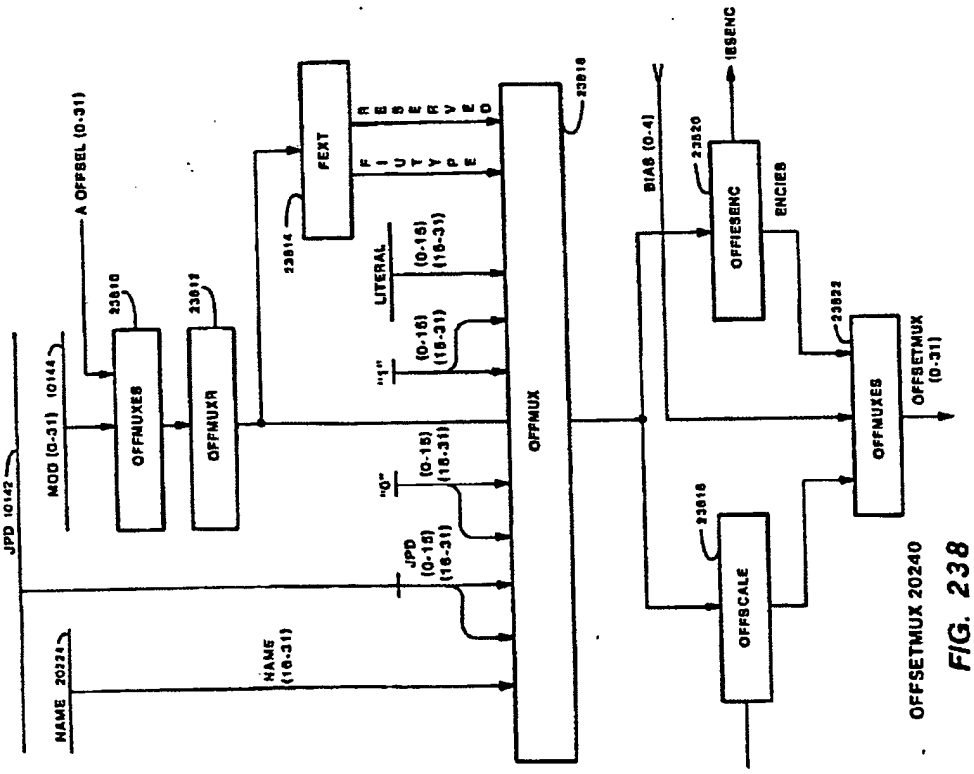
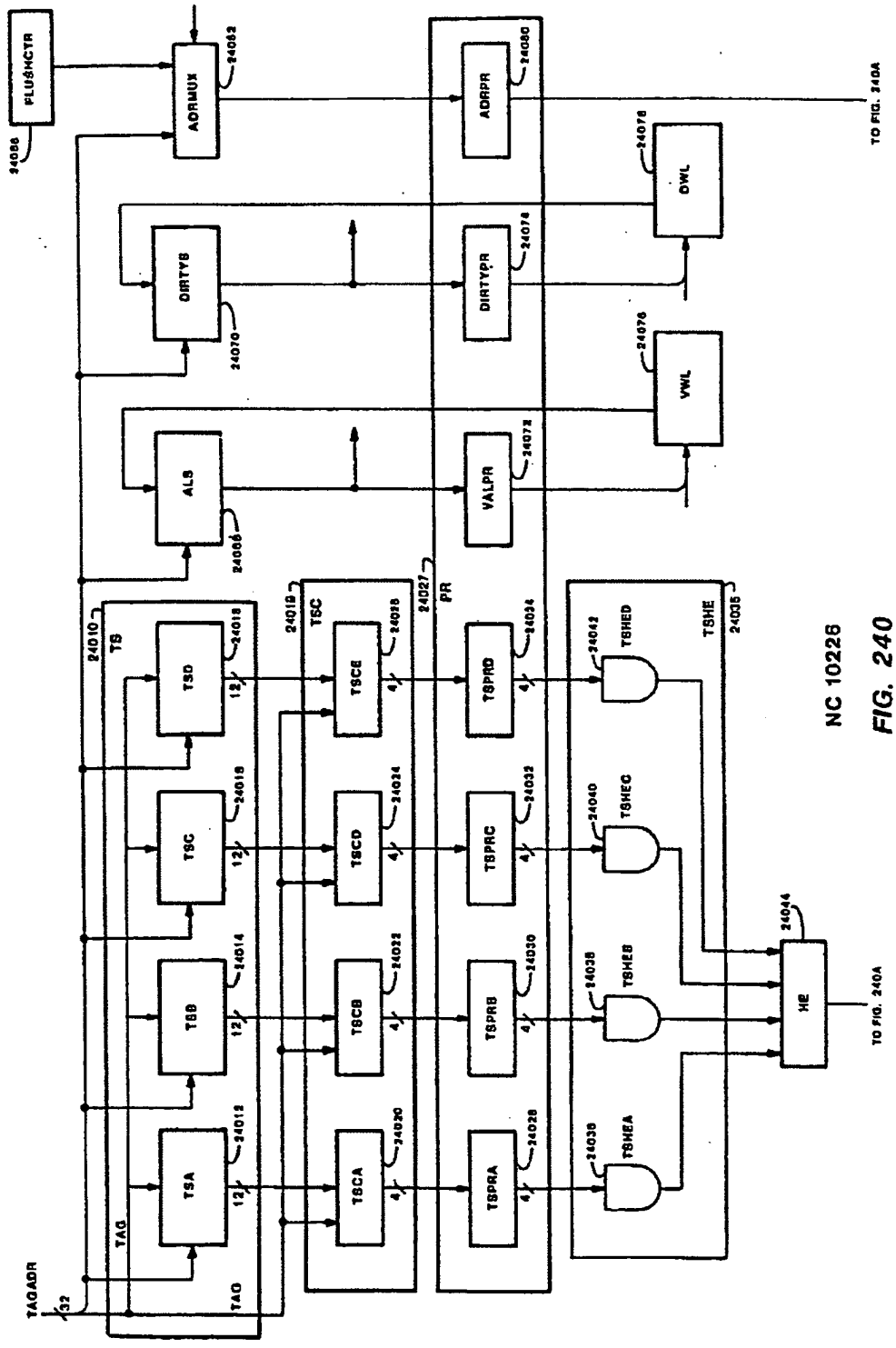


FIG. 238

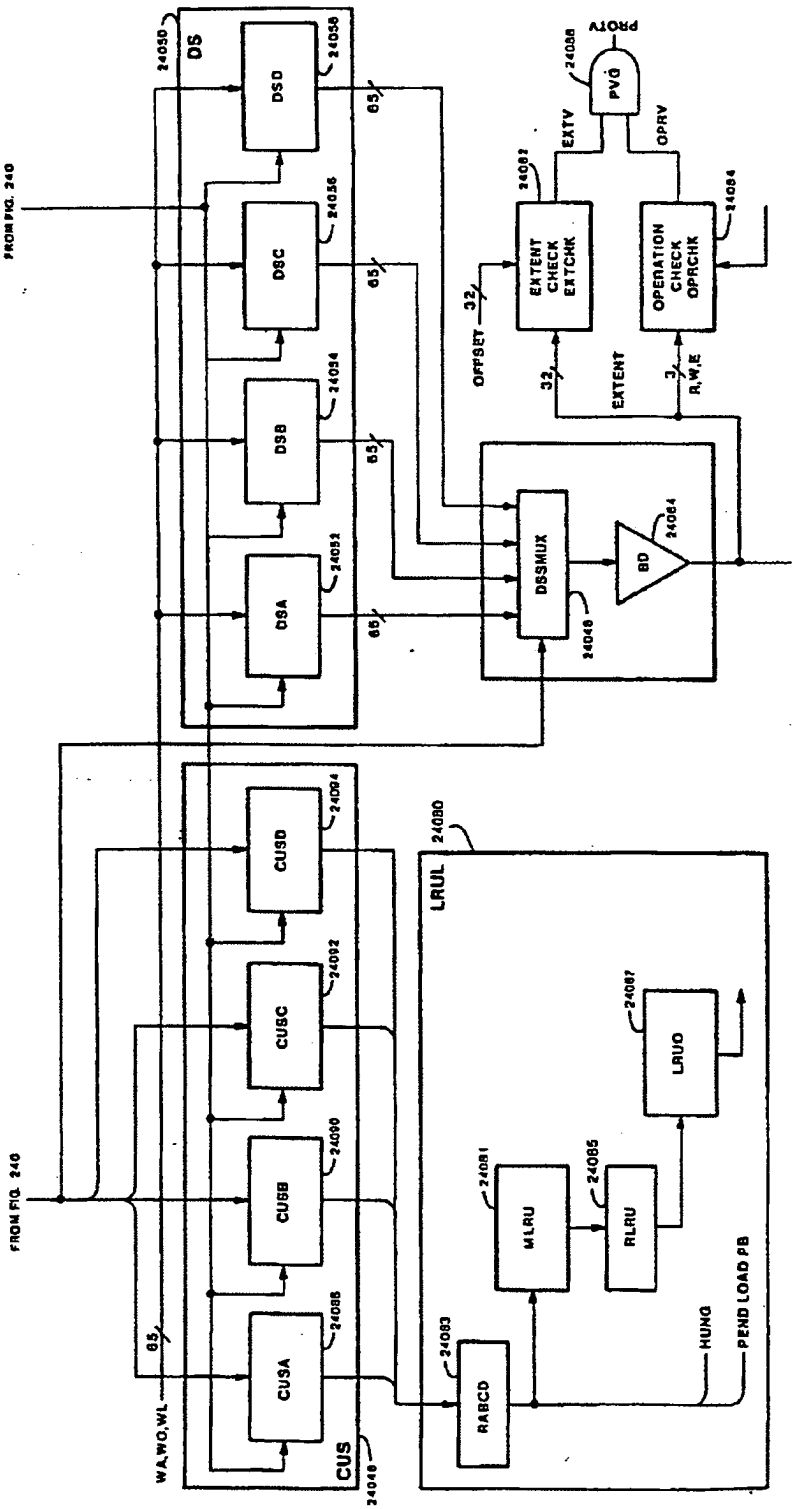
Petitioner Apple Inc. - Exhibit 1004, p. 5384



NC 10226

FIG. 240

Petitioner Apple Inc. - Exhibit 1004, p. 5385



NC 10226

FIG. 240A

Petitioner Apple Inc. - Exhibit 1004, p. 5386

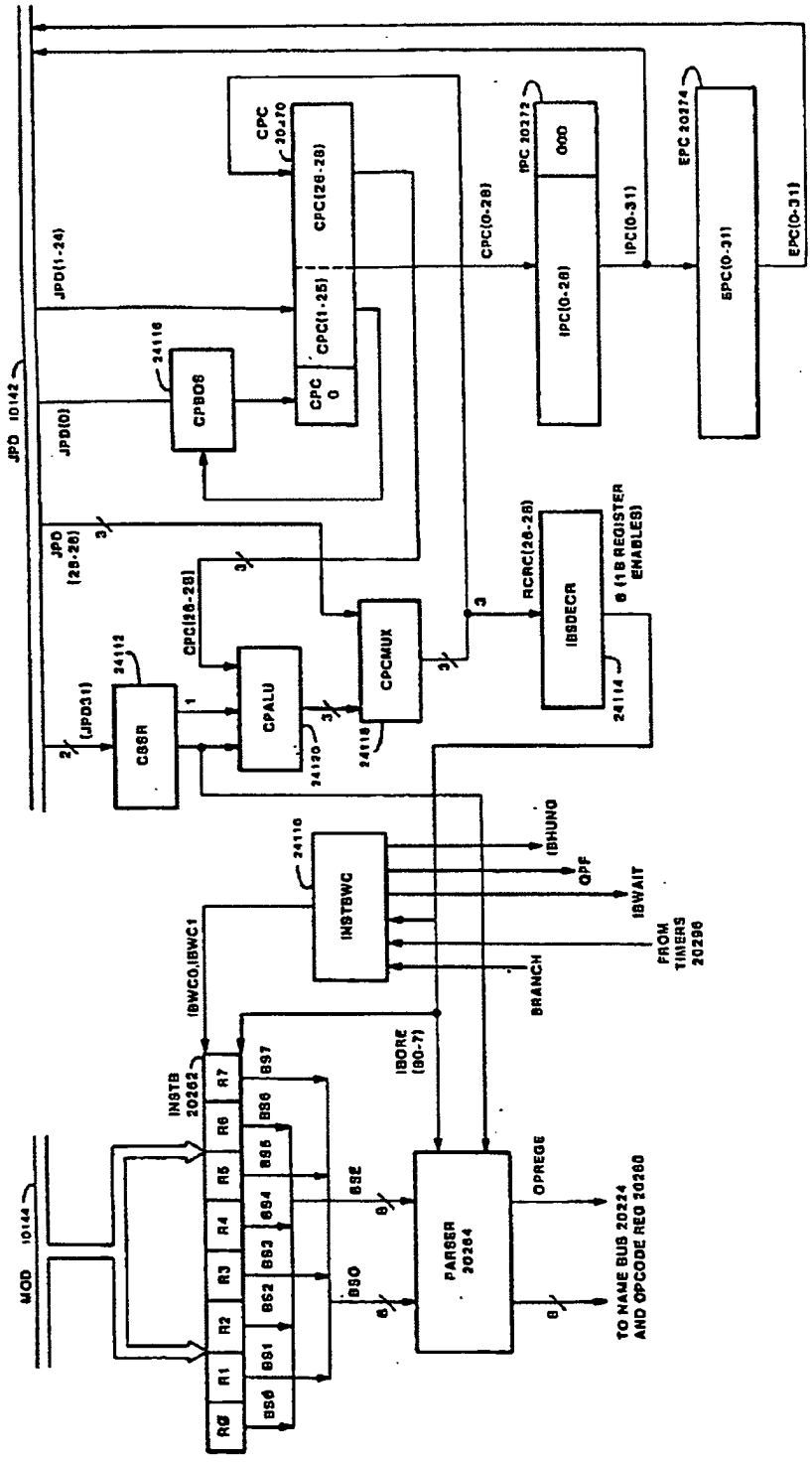


FIG. 241

Petitioner Apple Inc. - Exhibit 1004, p. 5387

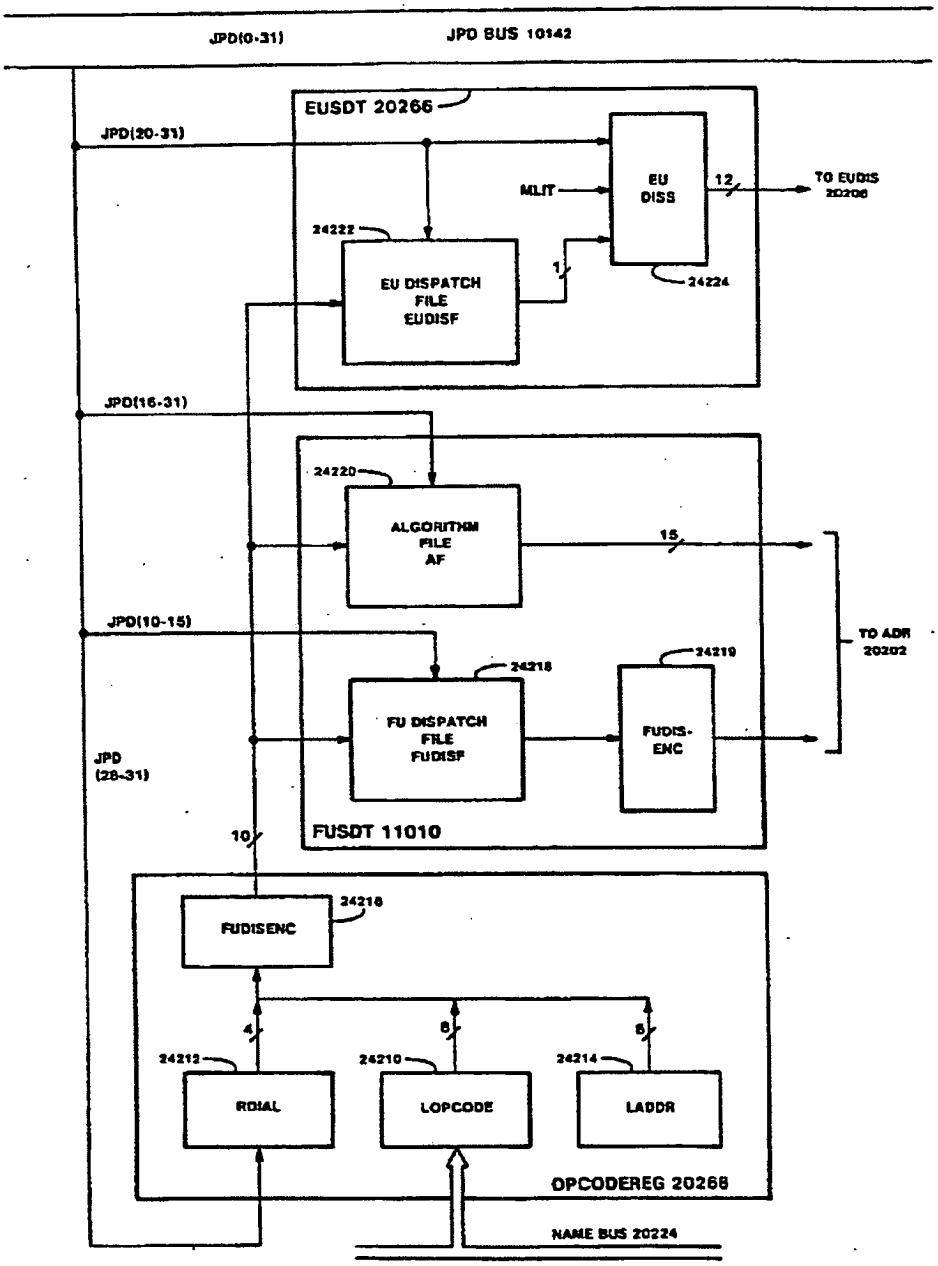


FIG. 242

Petitioner Apple Inc. - Exhibit 1004, p. 5388

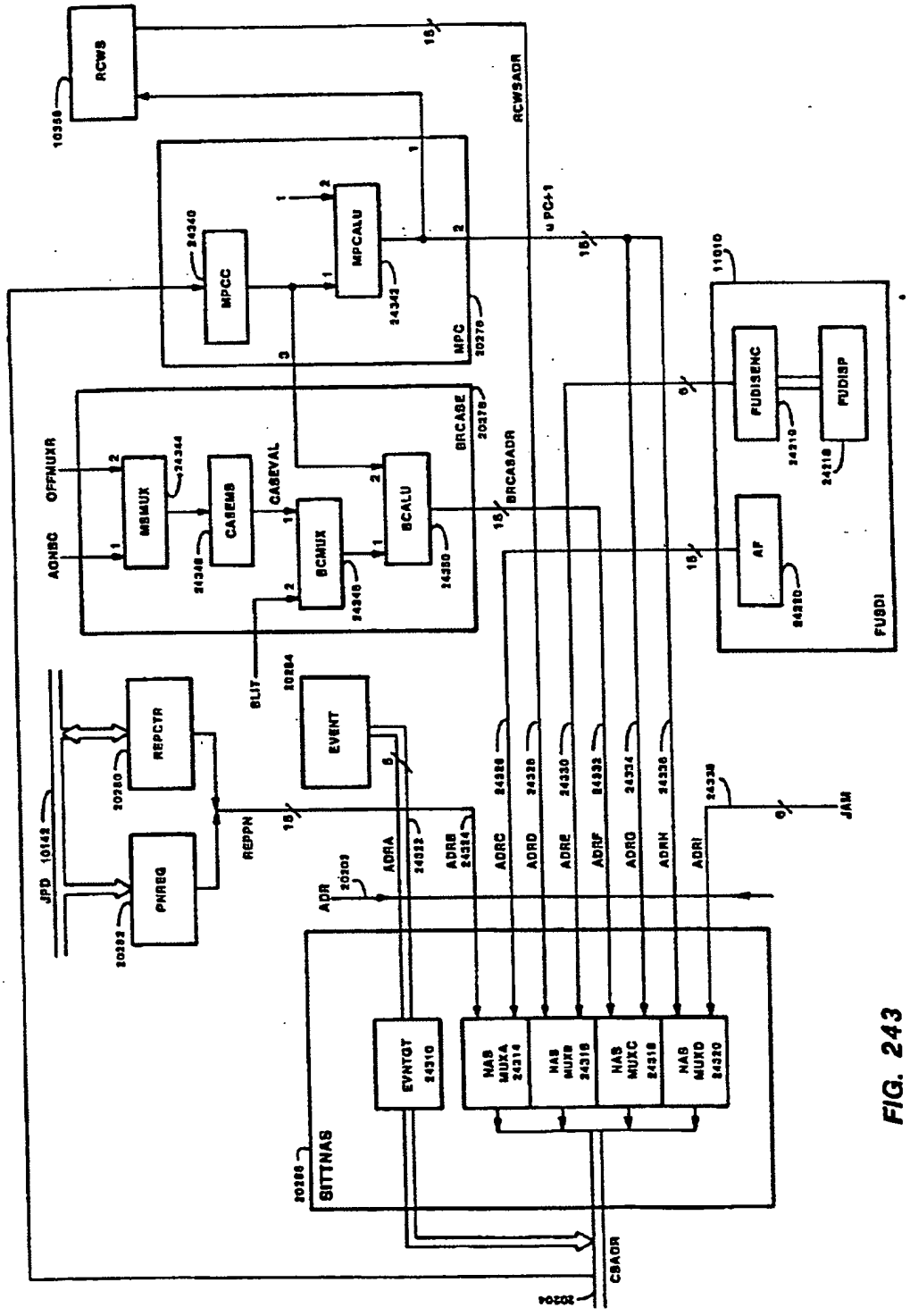


FIG. 243

Petitioner Apple Inc. - Exhibit 1004, p. 5389

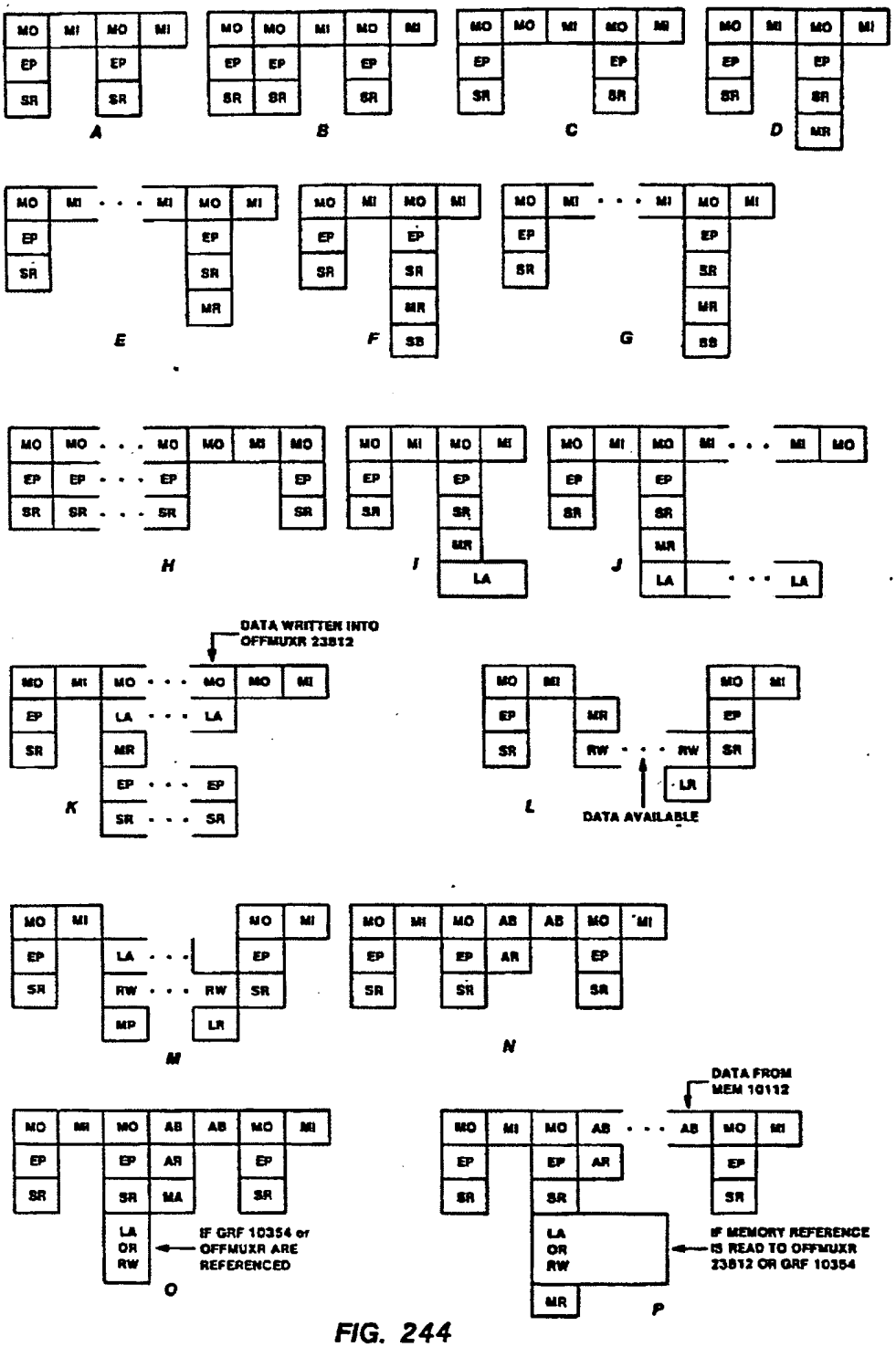


FIG. 244

Petitioner Apple Inc. - Exhibit 1004, p. 5390

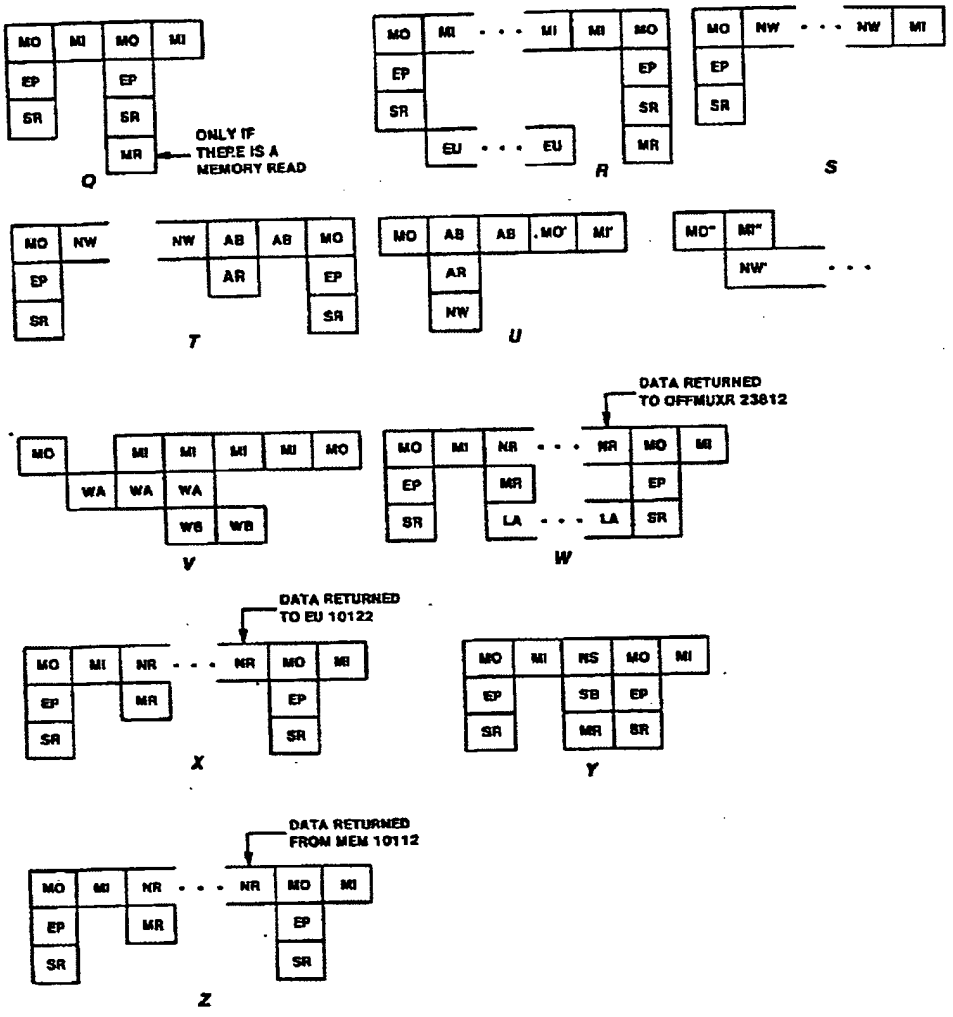


FIG. 244A

Petitioner Apple Inc. - Exhibit 1004, p. 5391

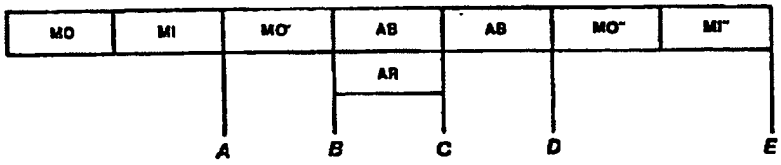


FIG. 245

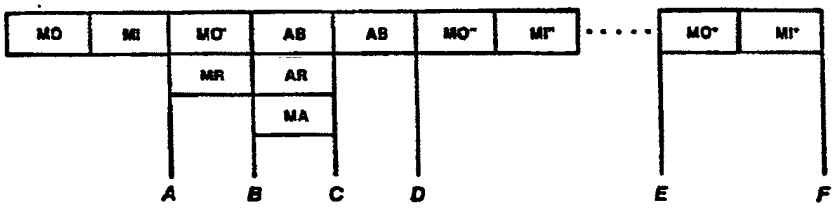


FIG. 246

Petitioner Apple Inc. - Exhibit 1004, p. 5392

PRIORITY LEVEL	EVENT	MASKED BY
0	E-UNIT STACK OVERFLOW	NONE
1	FATAL MEMORY ERROR	NONE
2	POWER FAIL	I
3	F-BOX STACK OVERFLOW	M,T,I
4	ILLEGAL E-UNIT DISPATCH (GATE FAULT)	NONE
5	STOREBACK EXCEPTION	MCWD
6	NAME TRACE TRAP	T,I
7	LOGICAL READ TRACE TRAP	T,I AND DES
8	LOGICAL WRITE TRACE TRAP	T,I AND DES
9	UID READ DEREFERENCE TRAP	DES
10	UID WRITE DEREFERENCE TRAP	DES
11	PROTECTION CACHE MISS	NONE
12	PROTECTION VIOLATION	MCWD
13	PAGE CROSSING INTERRUPT	NONE
14	LAT	NONE
15	WRITE LAT	NONE
16	MEMORY REFERENCE REPEAT	NONE
17	EGG TIMER OVERFLOW	A,M,T,I
18	E-BOX STACK UNDERFLOW	A,M,T,I
19	NON-FATAL MEMORY ERROR	NONE
20	INTERVAL TIMER OVEREFLOW	NONE
21	IPM INTERRUPT	NONE
22	S-OP TRACE TRAP	NONE
23	ILLEGAL S-OP	NONE
24	MICROINSTRUCTION TRACE TRAP	NONE
25	NON-PRESENT MICROINSTRUCTION	NONE
26	INSTRUCTION PREFETCH IS HUNG	NONE
27	F-BOX STACK UNDERFLOW	NONE
28	MICROINSTRUCTION BREAK POINT TRACE TRAP	T,I AND MCWD
29	MISS ON NAME CACHE LOAD OR READ REGISTER	NONE

FIG. 247

Petitioner Apple Inc. - Exhibit 1004, p. 5393

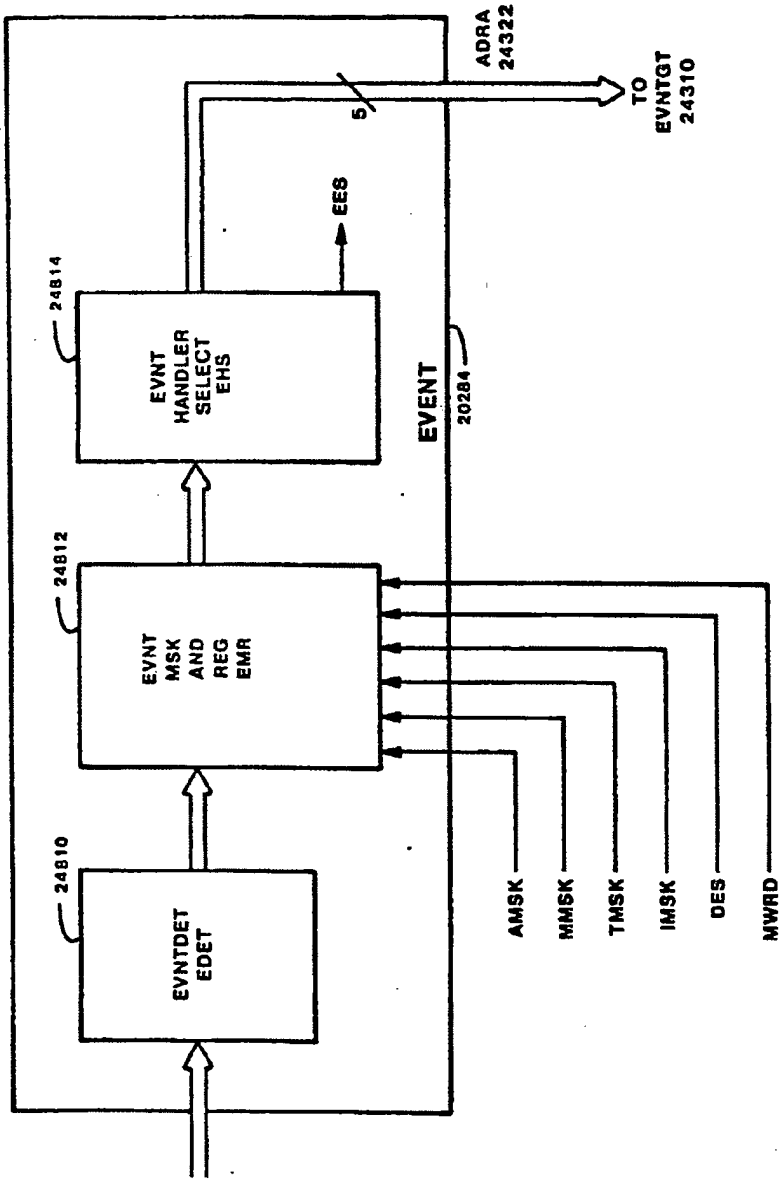


FIG. 248

Petitioner Apple Inc. - Exhibit 1004, p. 5394

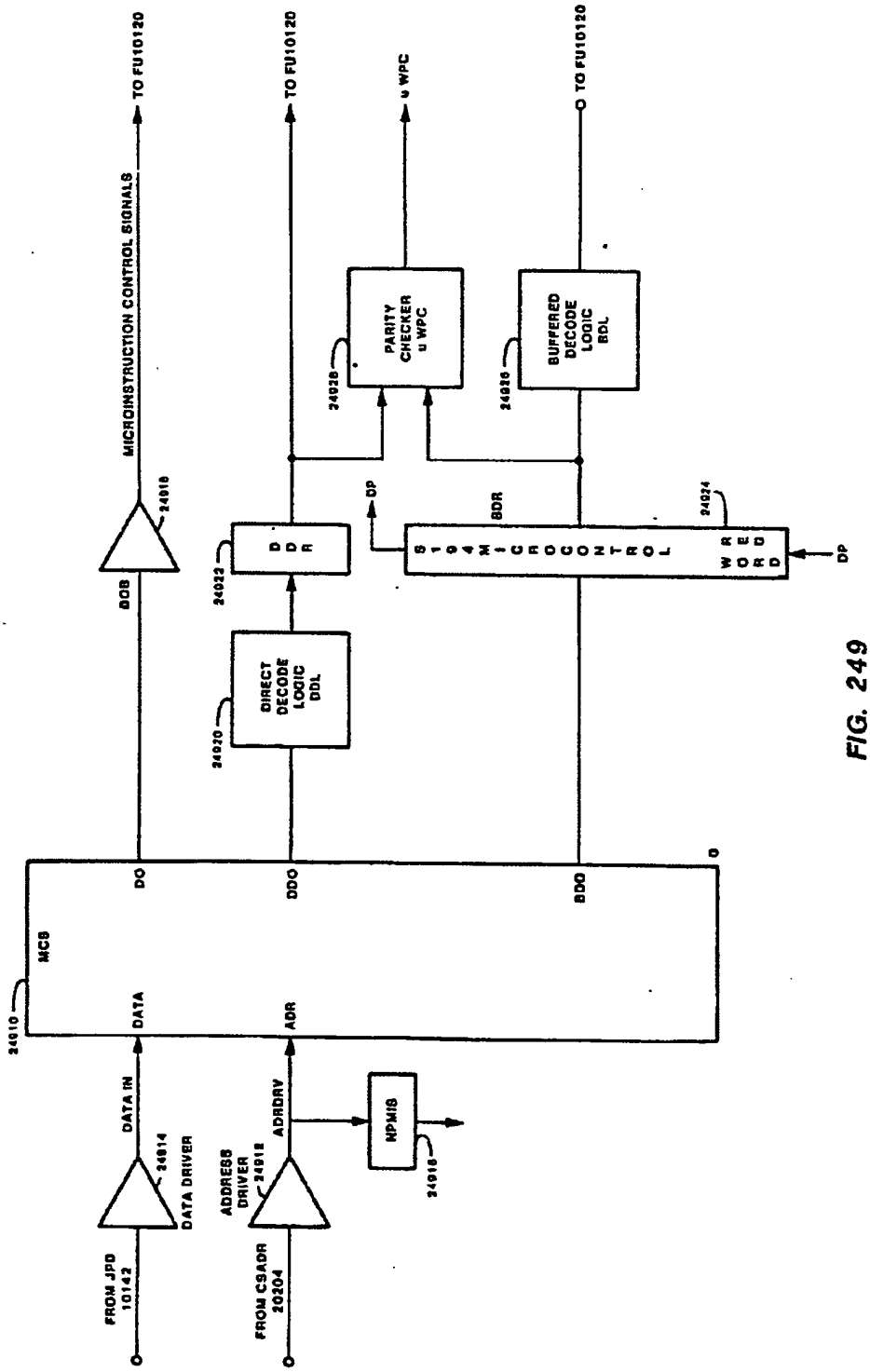


FIG. 249

Petitioner Apple Inc. - Exhibit 1004, p. 5395



FIG. 251

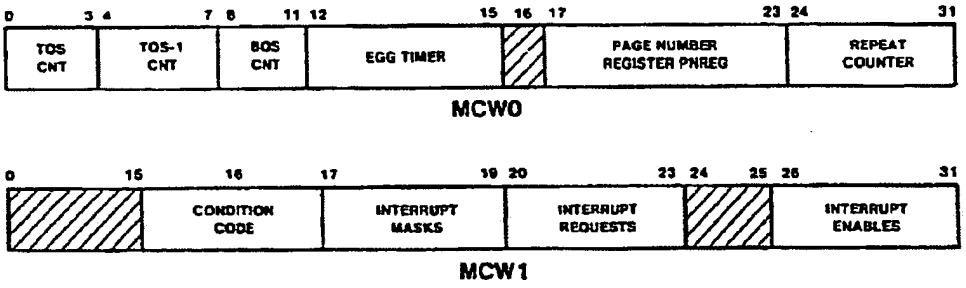


FIG. 252

Petitioner Apple Inc. - Exhibit 1004, p. 5396

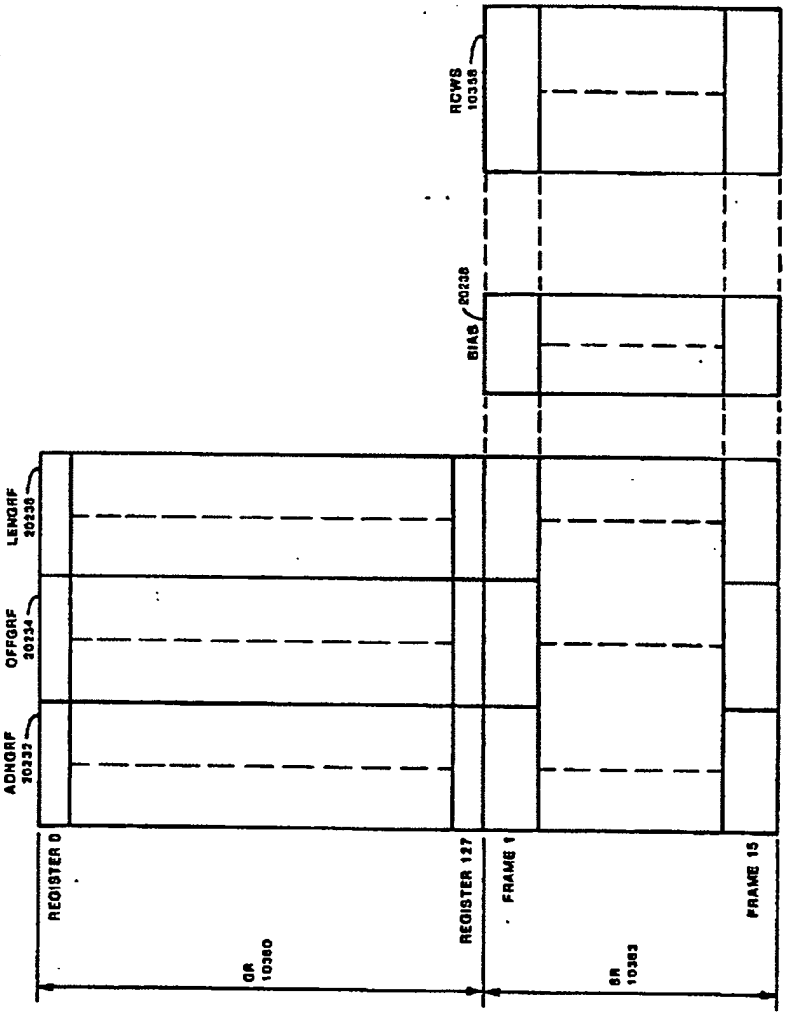


FIG. 253

Petitioner Apple Inc. - Exhibit 1004, p. 5397

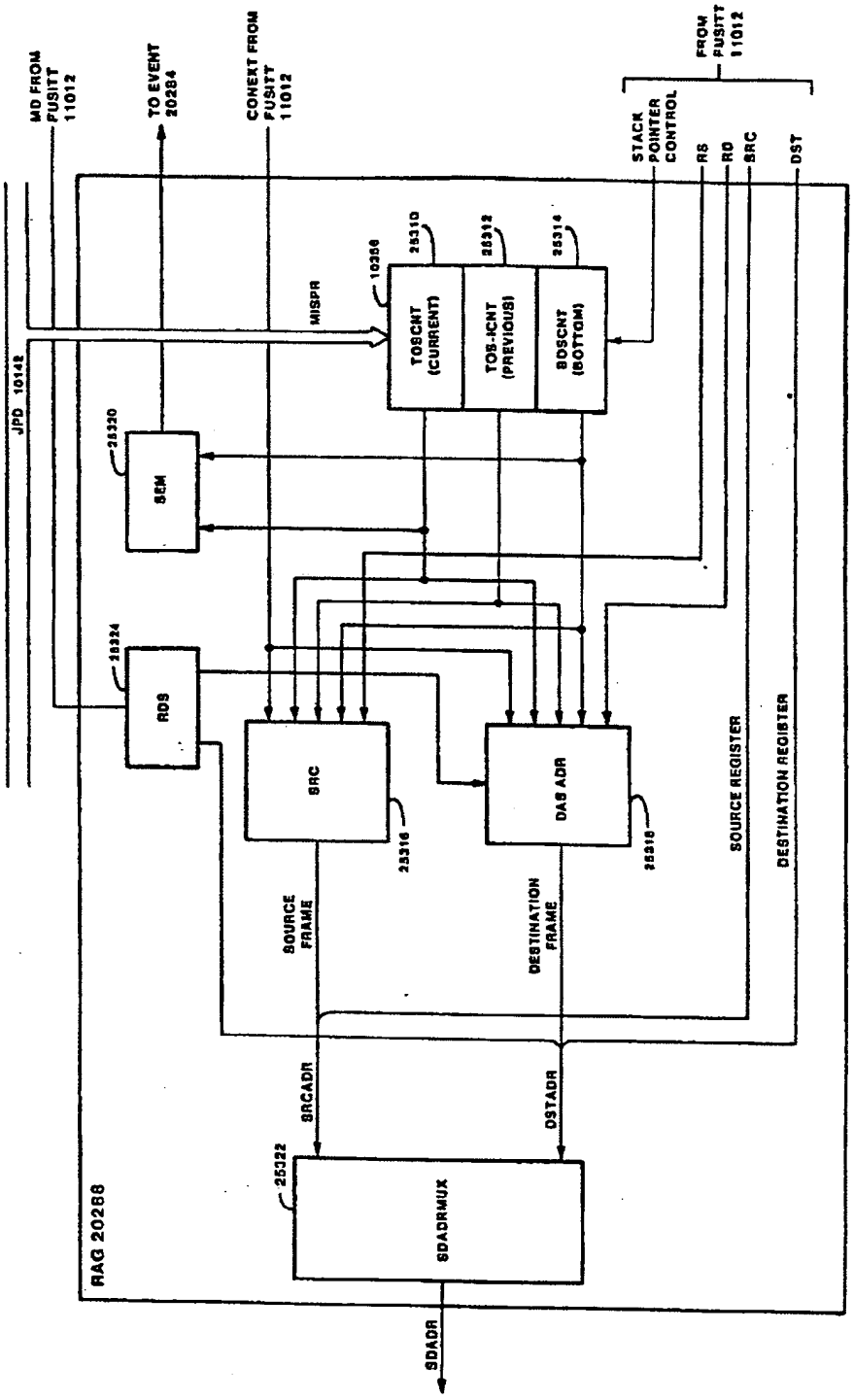


FIG. 253A

Petitioner Apple Inc. - Exhibit 1004, p. 5398

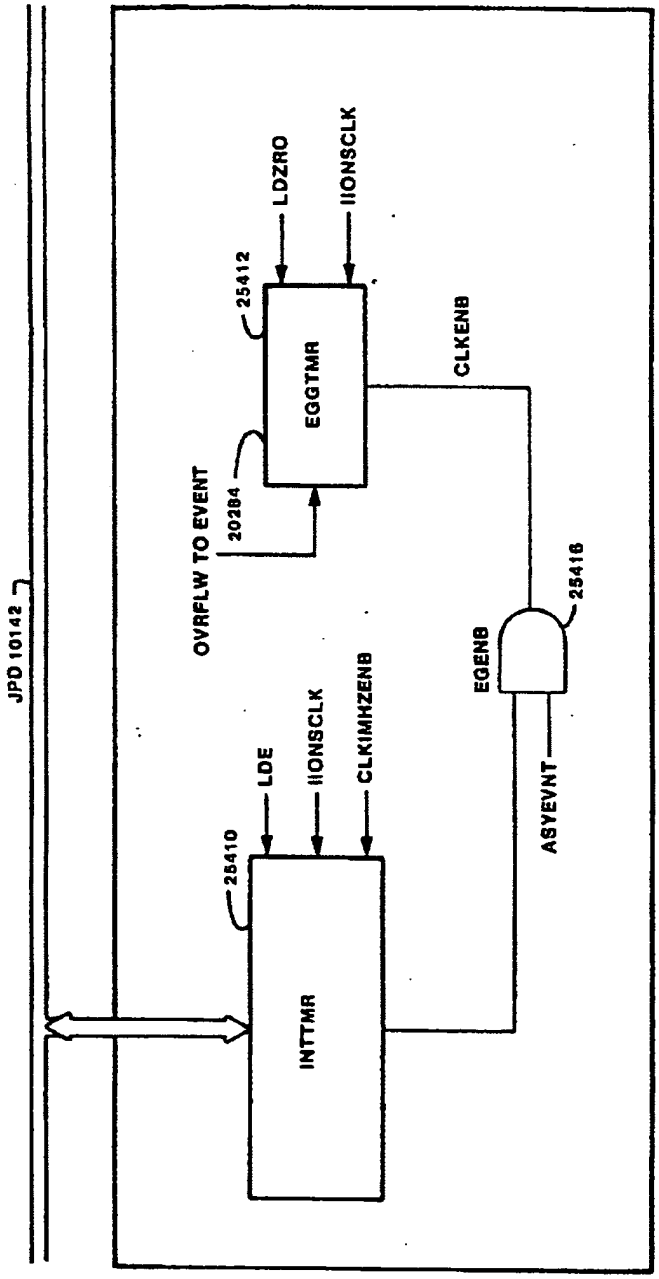


FIG. 254

Petitioner Apple Inc. - Exhibit 1004, p. 5399

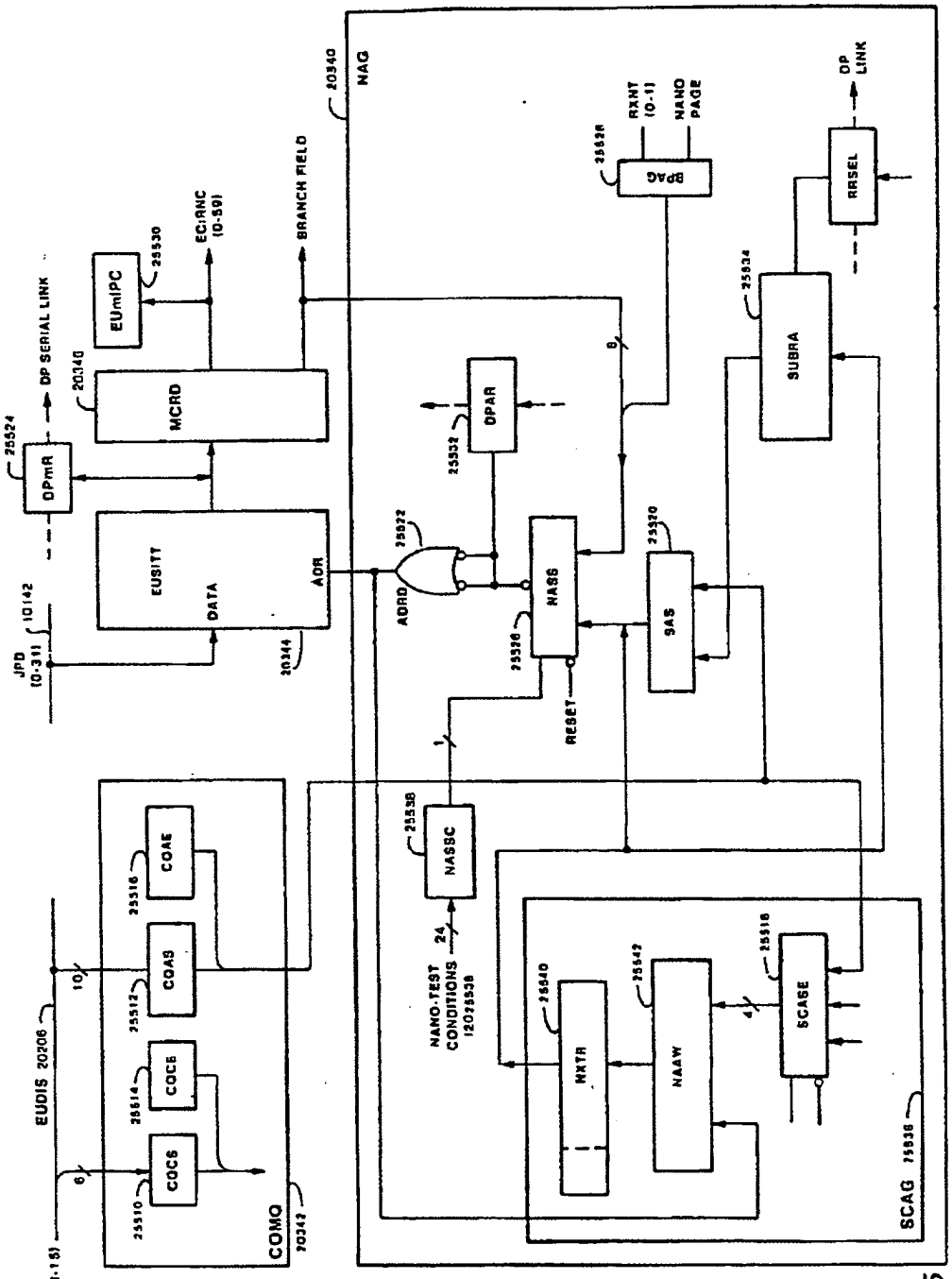


FIG. 255

Petitioner Apple Inc. - Exhibit 1004, p. 5400

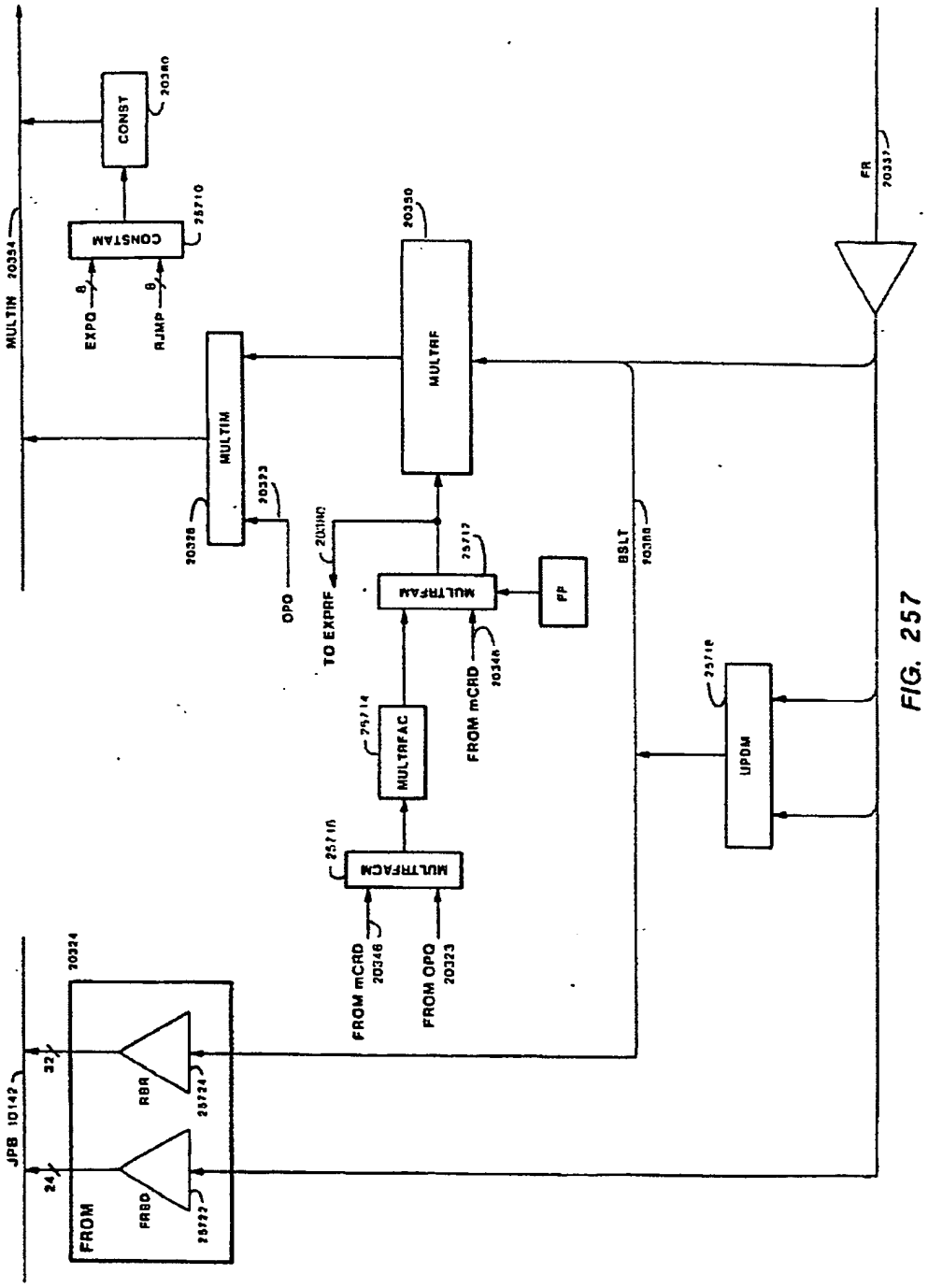


FIG. 257

Petitioner Apple Inc. - Exhibit 1004, p. 5401

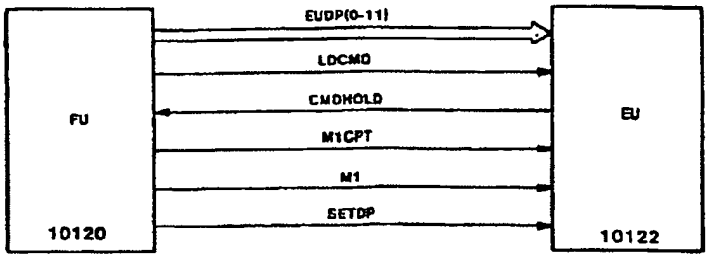


FIG. 260

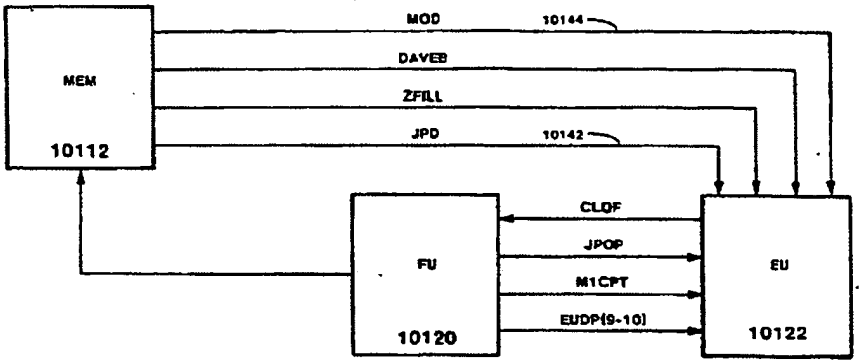


FIG. 261

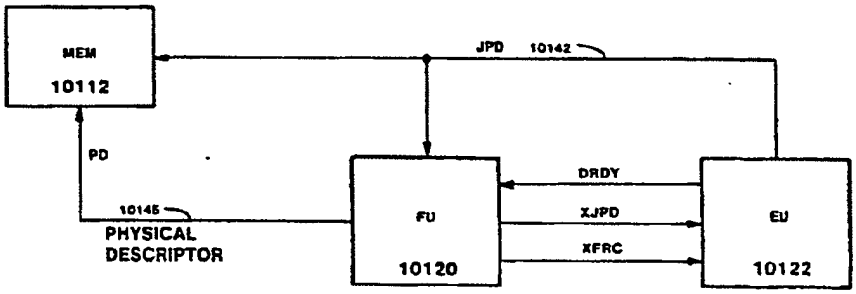


FIG. 262

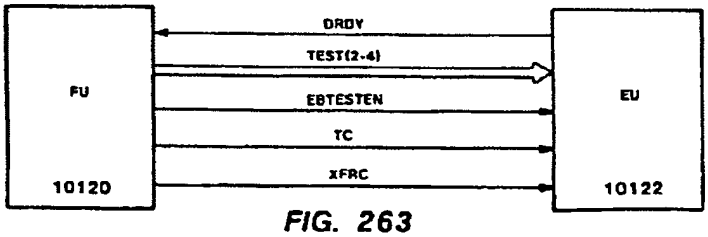


FIG. 263

Petitioner Apple Inc. - Exhibit 1004, p. 5402

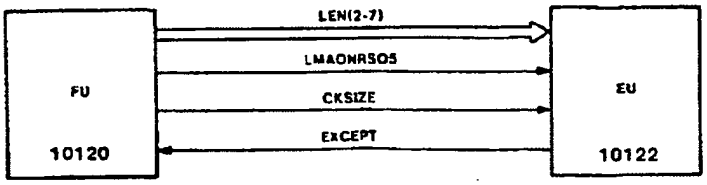


FIG. 264

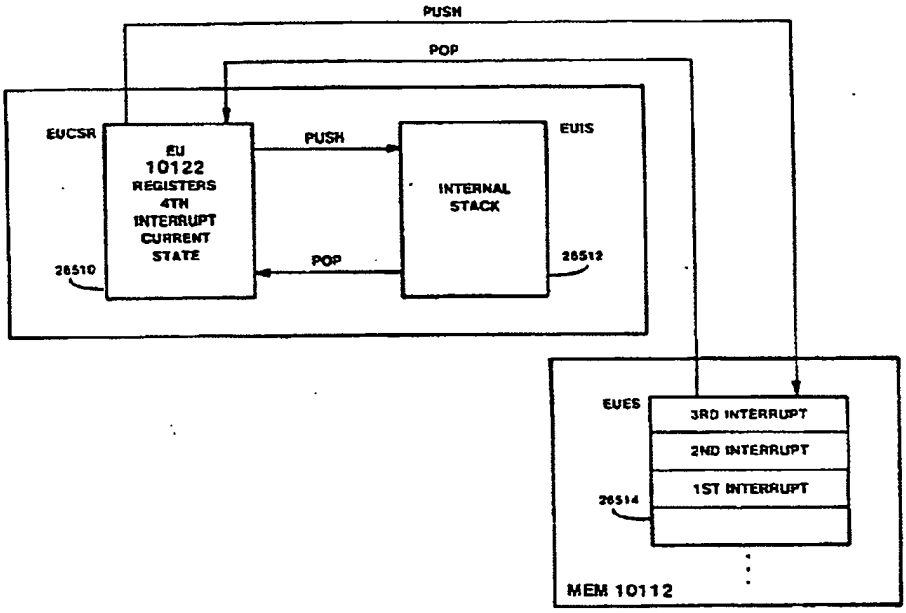


FIG. 265

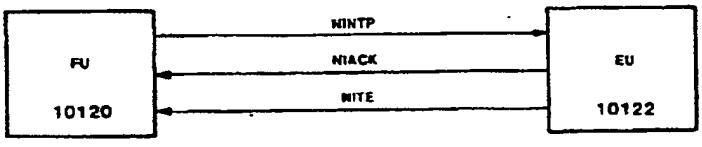


FIG. 266

Petitioner Apple Inc. - Exhibit 1004, p. 5403

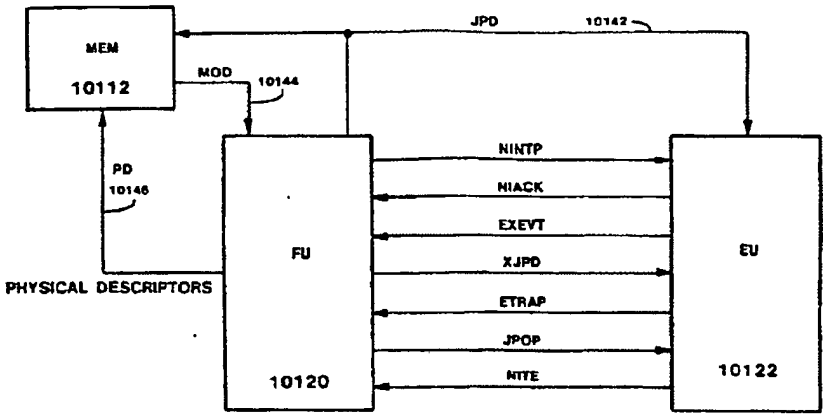


FIG. 267

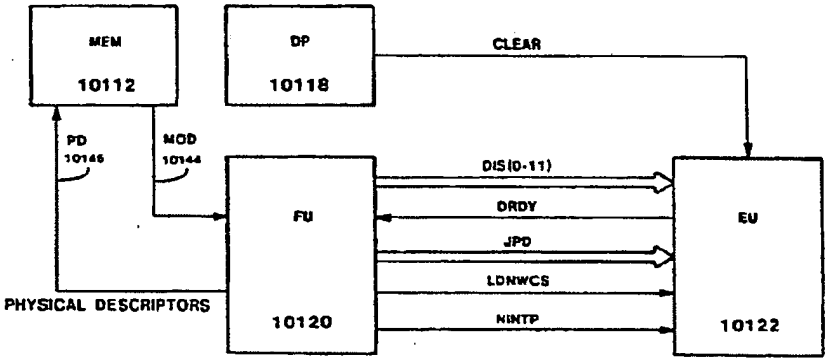


FIG. 268

Petitioner Apple Inc. - Exhibit 1004, p. 5404

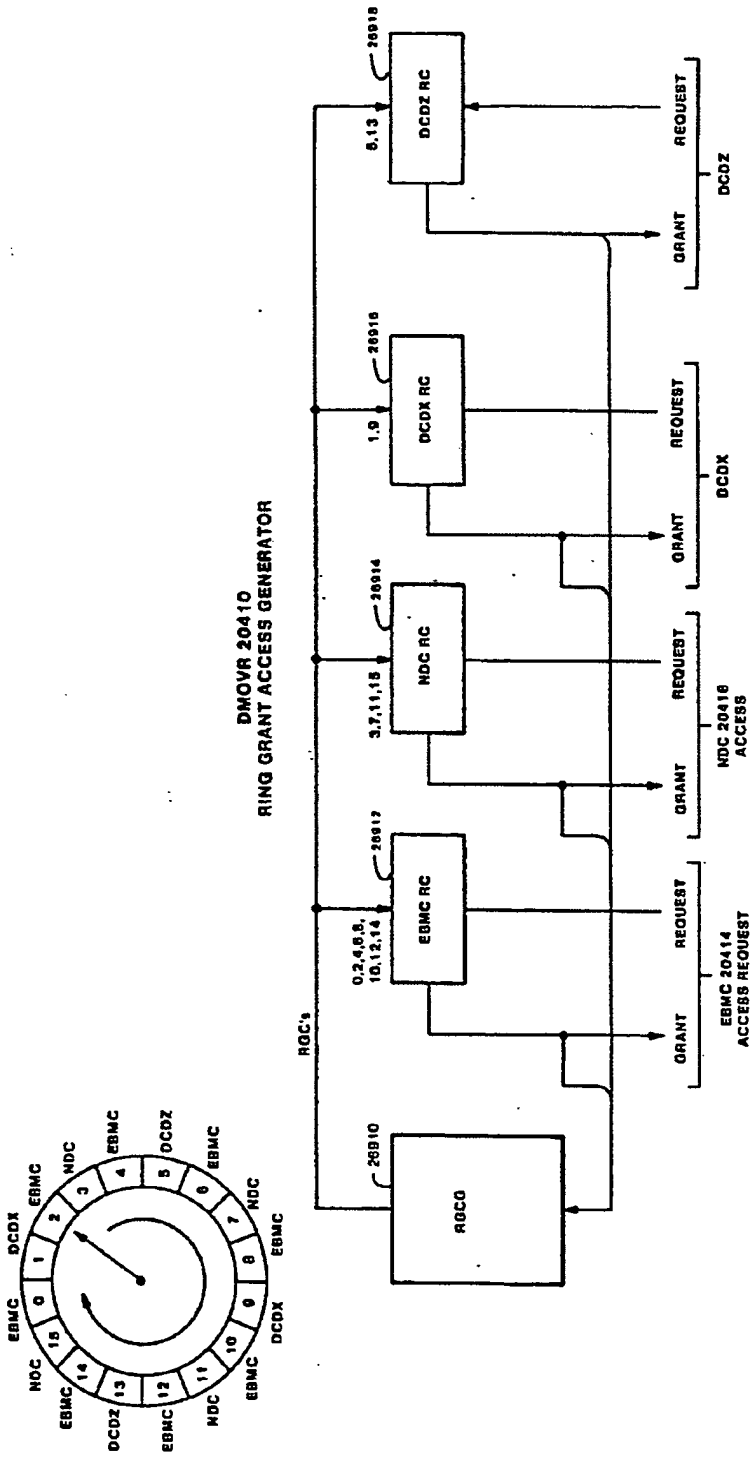


FIG 269

Petitioner Apple Inc. - Exhibit 1004, p. 5405

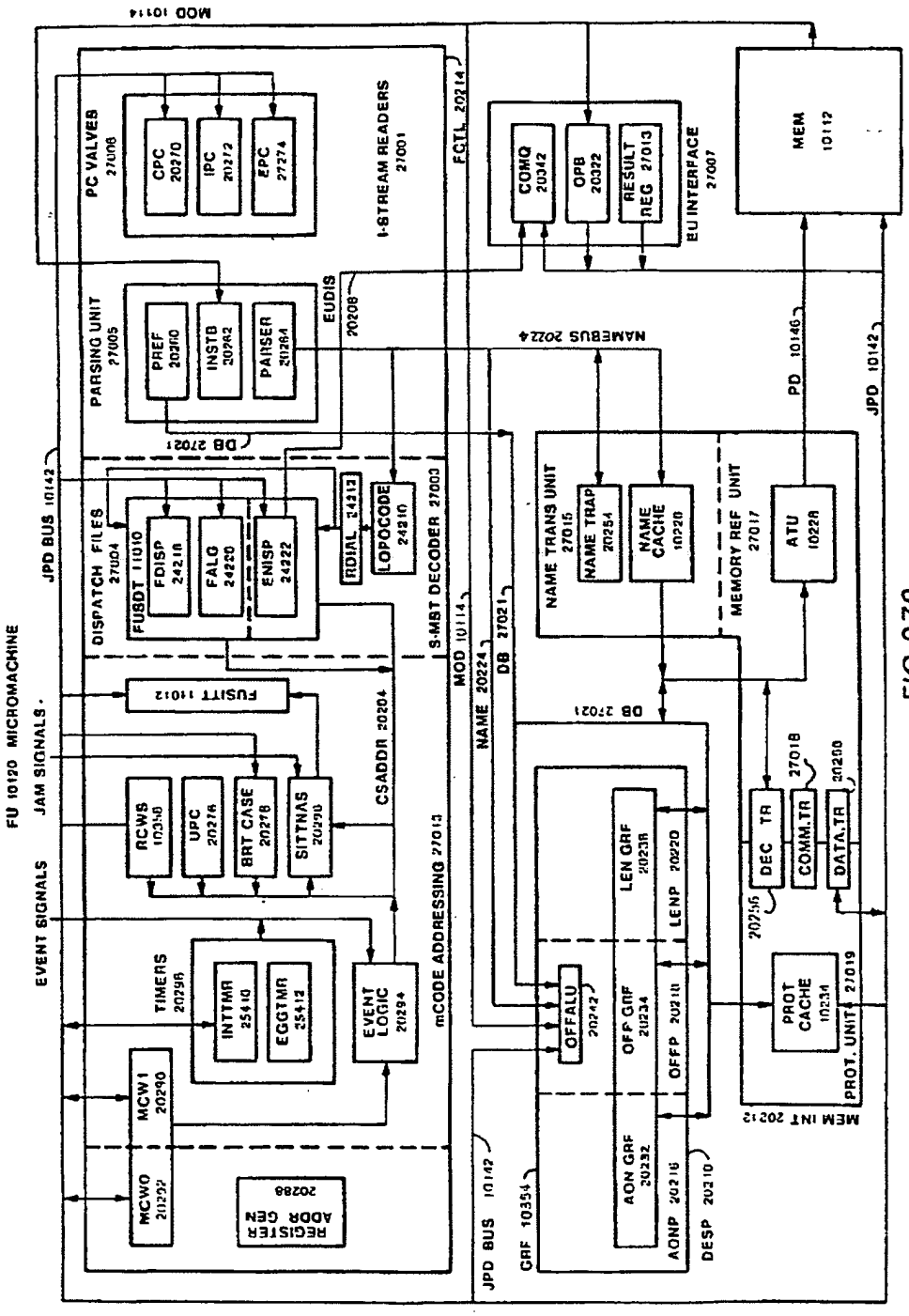


FIG 270

Petitioner Apple Inc. - Exhibit 1004, p. 5406

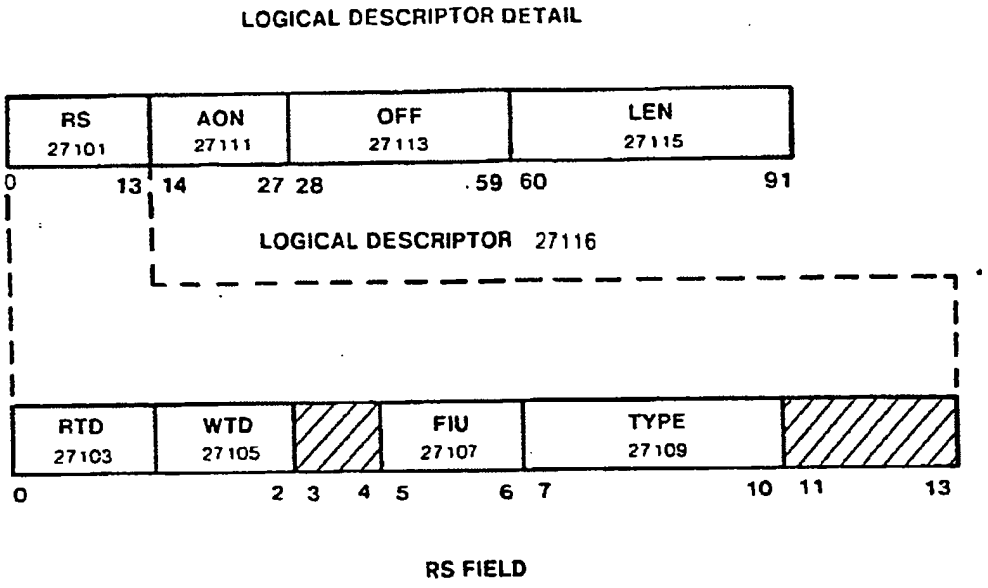


FIG. 271

Petitioner Apple Inc. - Exhibit 1004, p. 5407

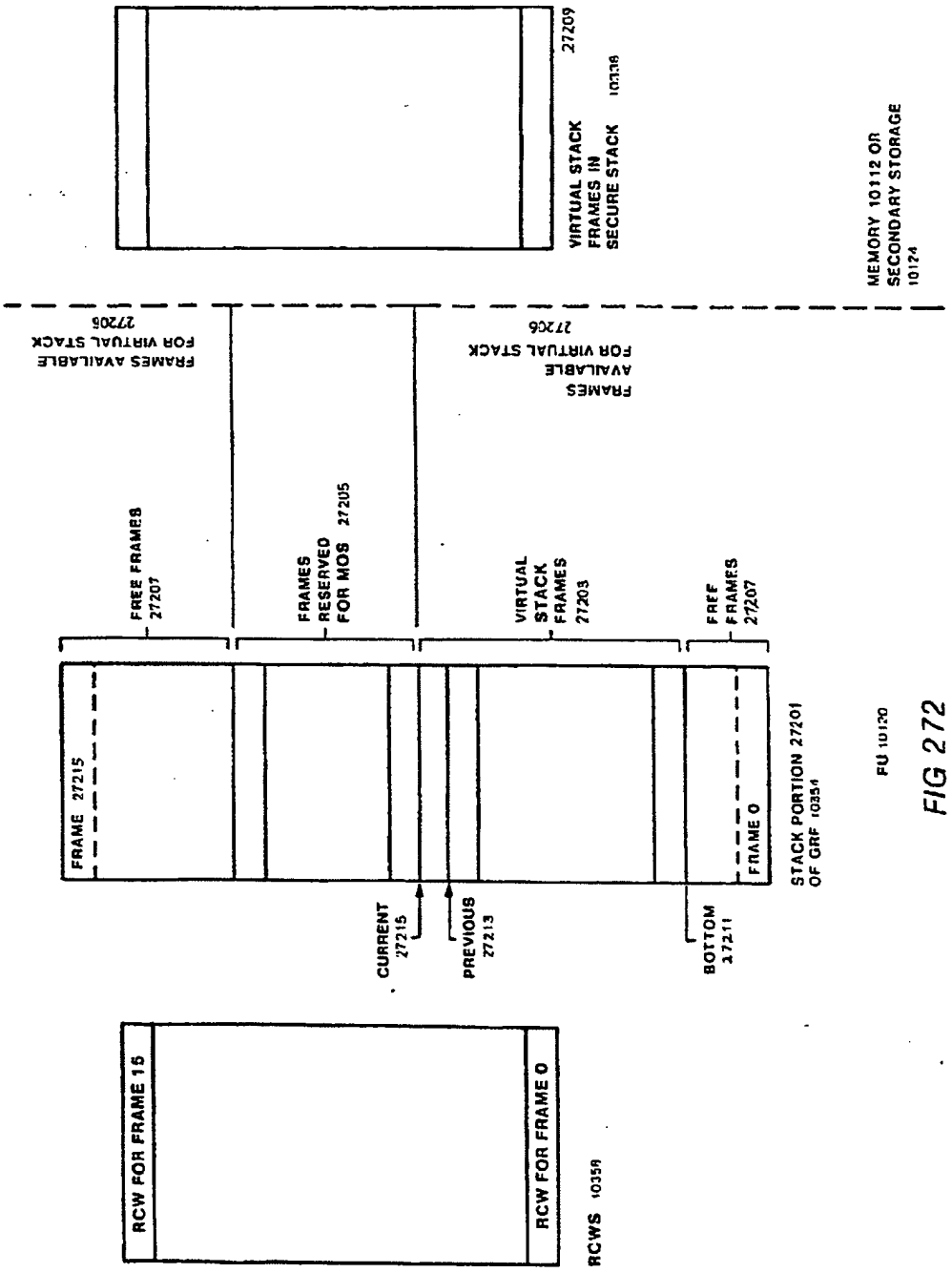


FIG 272

Petitioner Apple Inc. - Exhibit 1004, p. 5408

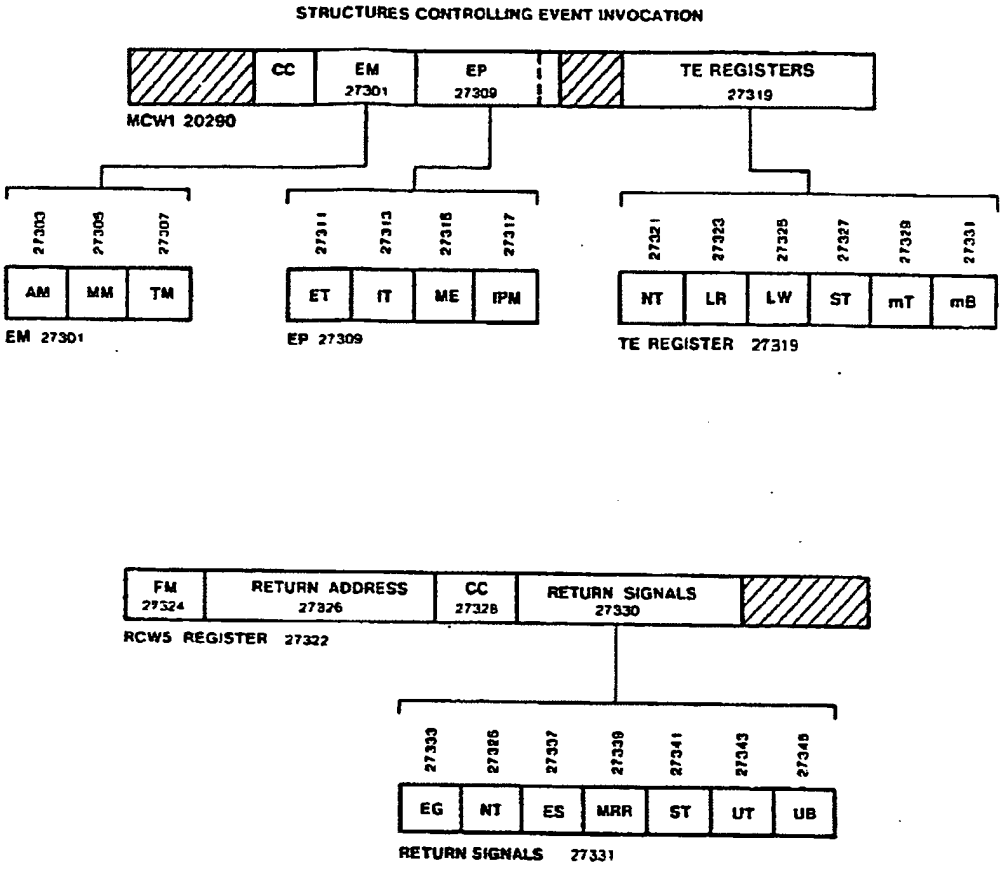


FIG. 273

Petitioner Apple Inc. - Exhibit 1004, p. 5409

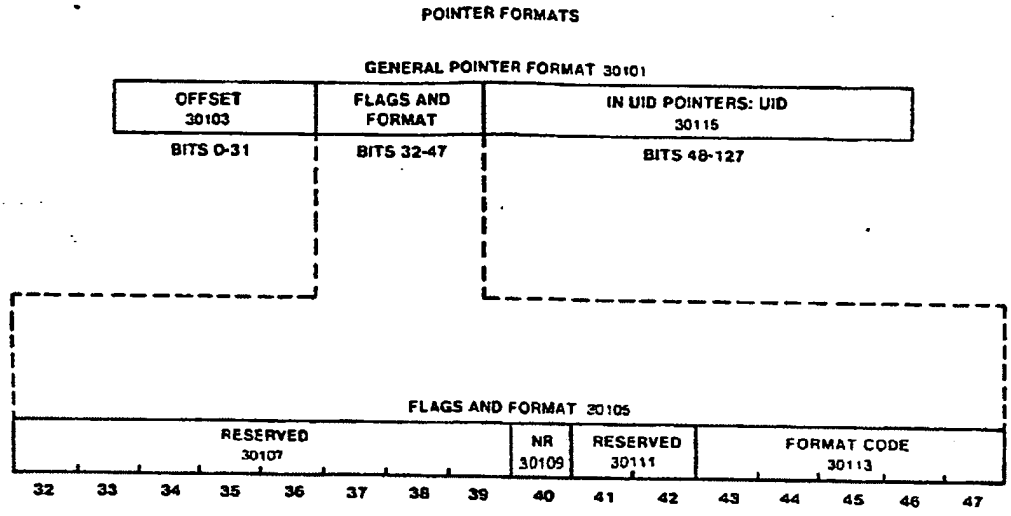


FIG. 301

Petitioner Apple Inc. - Exhibit 1004, p. 5410

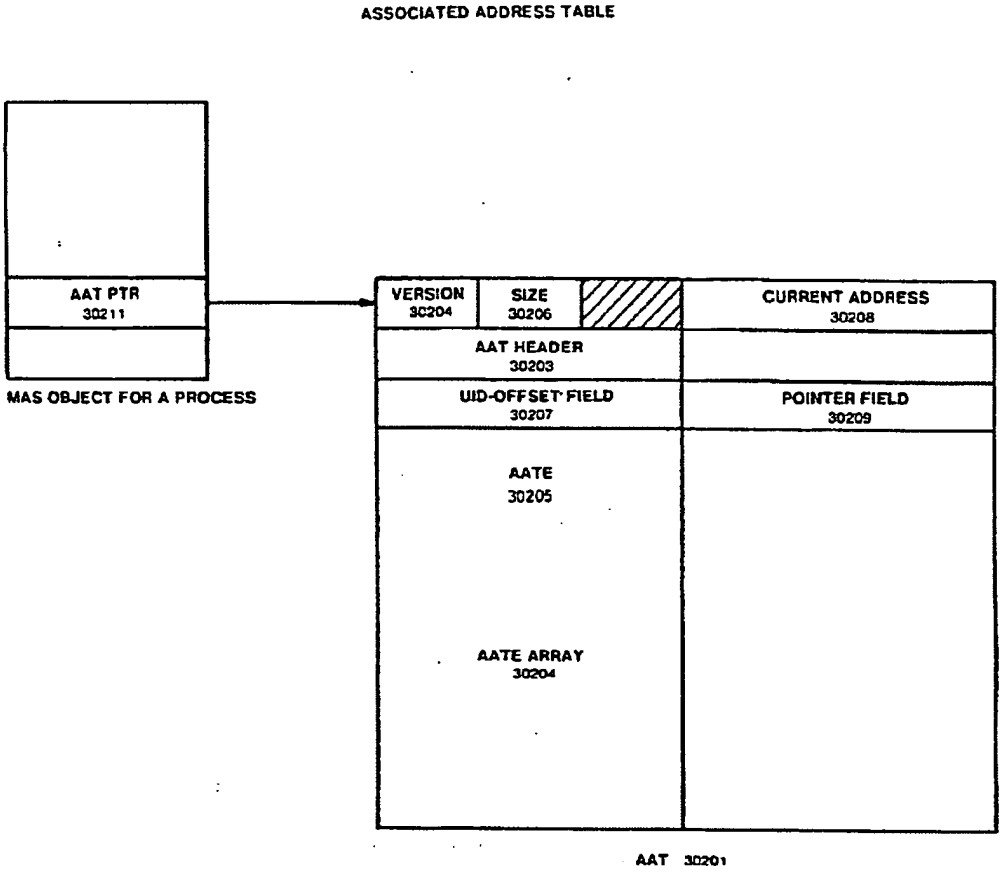


FIG. 302

Petitioner Apple Inc. - Exhibit 1004, p. 5411

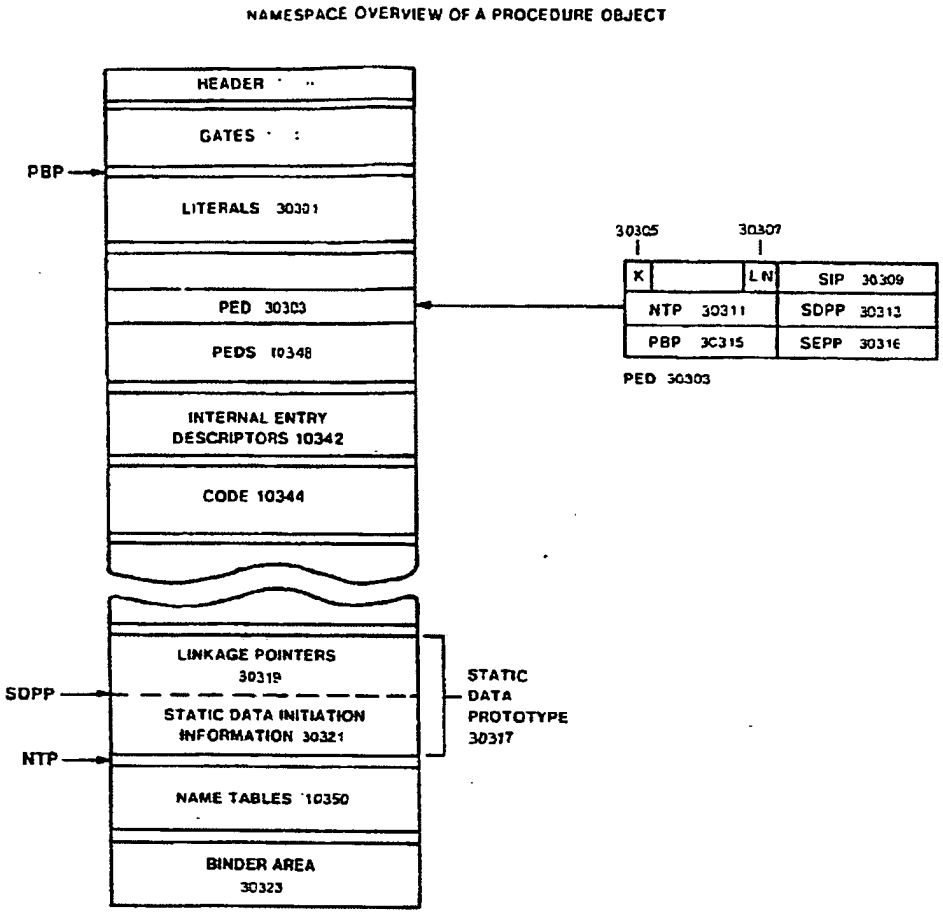


FIG. 303

Petitioner Apple Inc. - Exhibit 1004, p. 5412

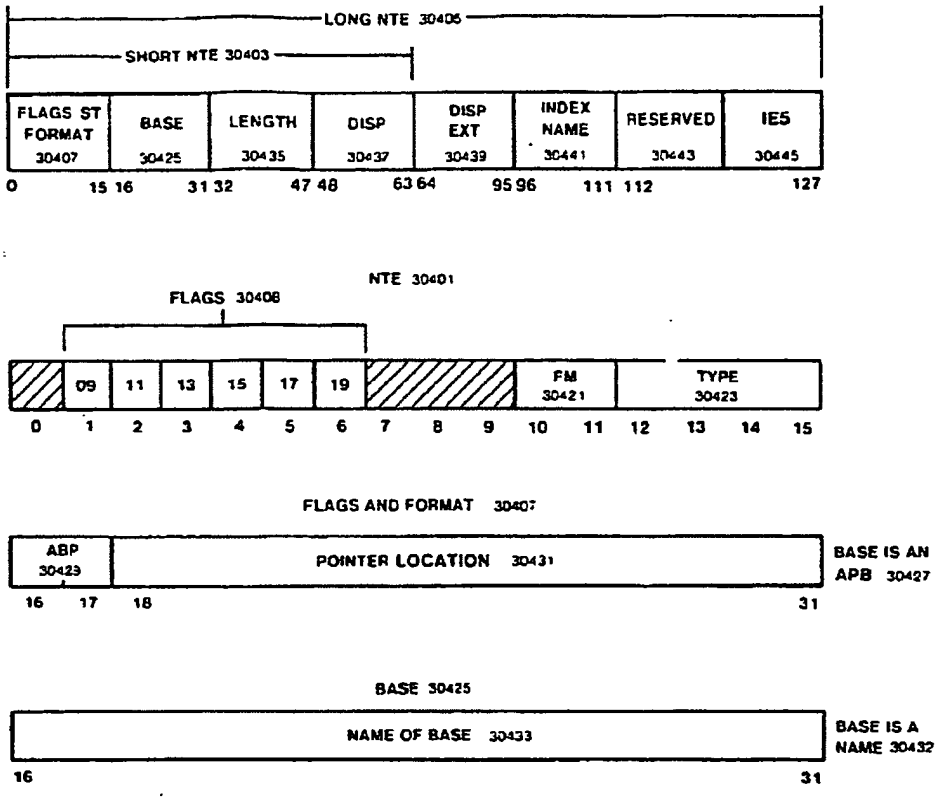


FIG. 304

Petitioner Apple Inc. - Exhibit 1004, p. 5413

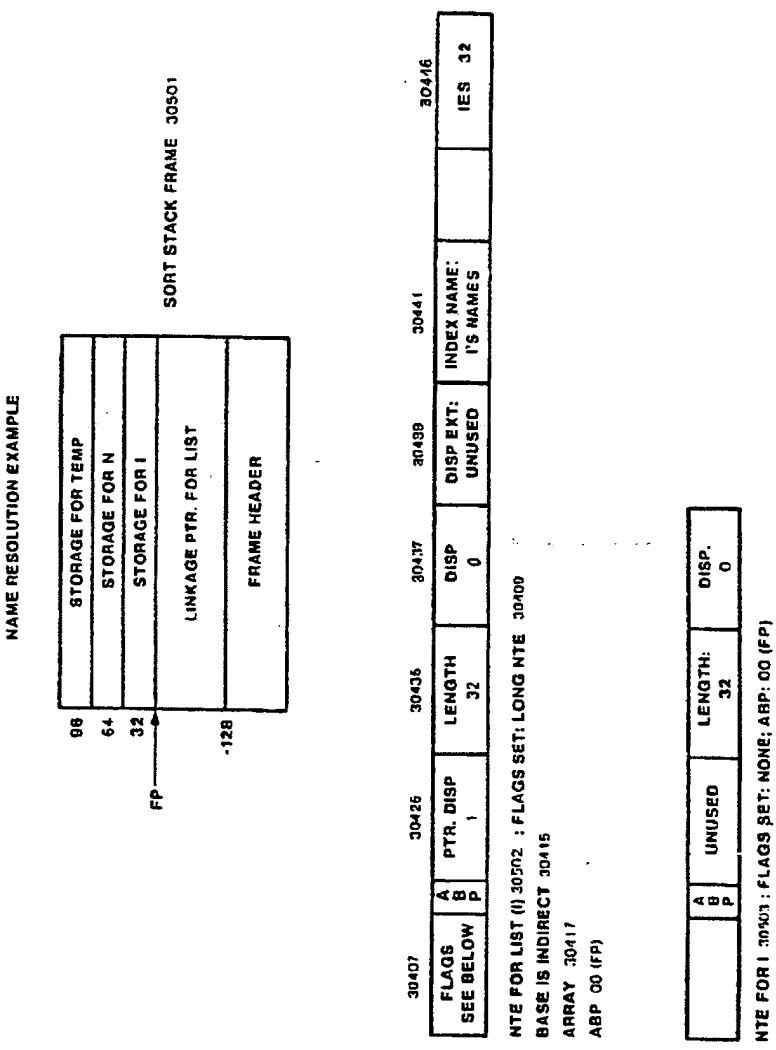


FIG. 305

Petitioner Apple Inc. - Exhibit 1004, p. 5414

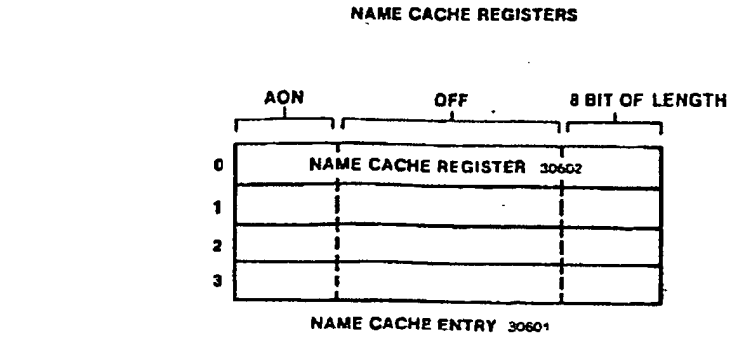


FIG. 306

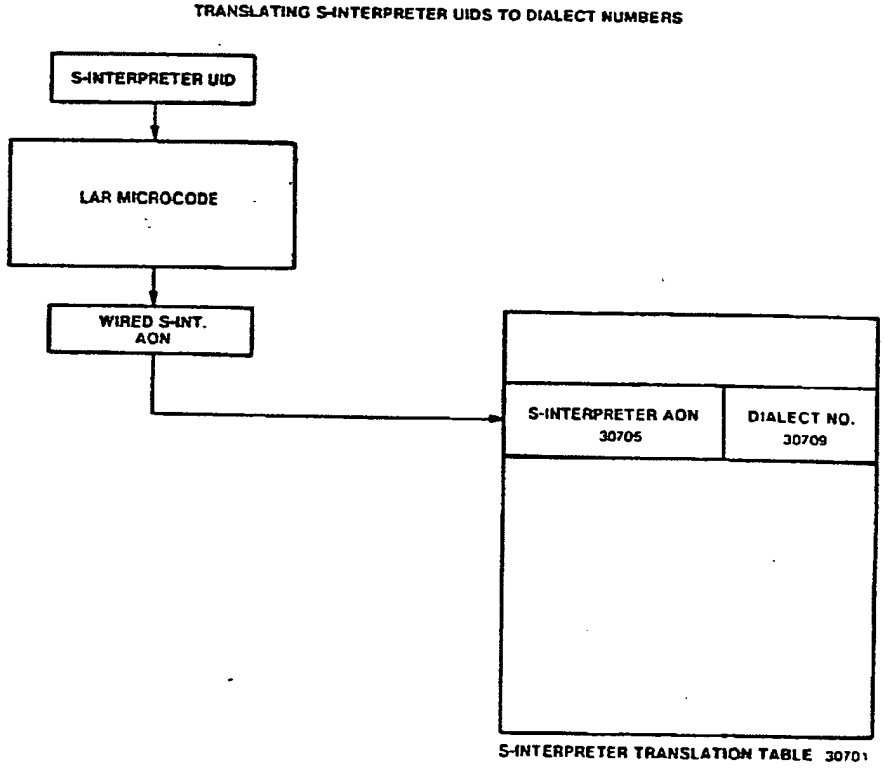


FIG. 307

Petitioner Apple Inc. - Exhibit 1004, p. 5415

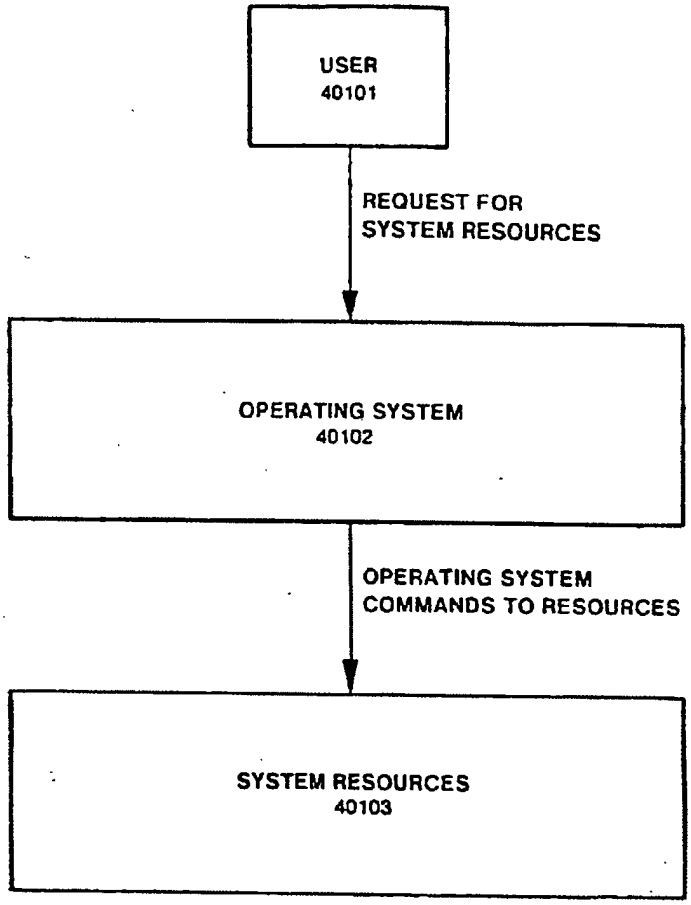


FIG 401

Petitioner Apple Inc. - Exhibit 1004, p. 5416

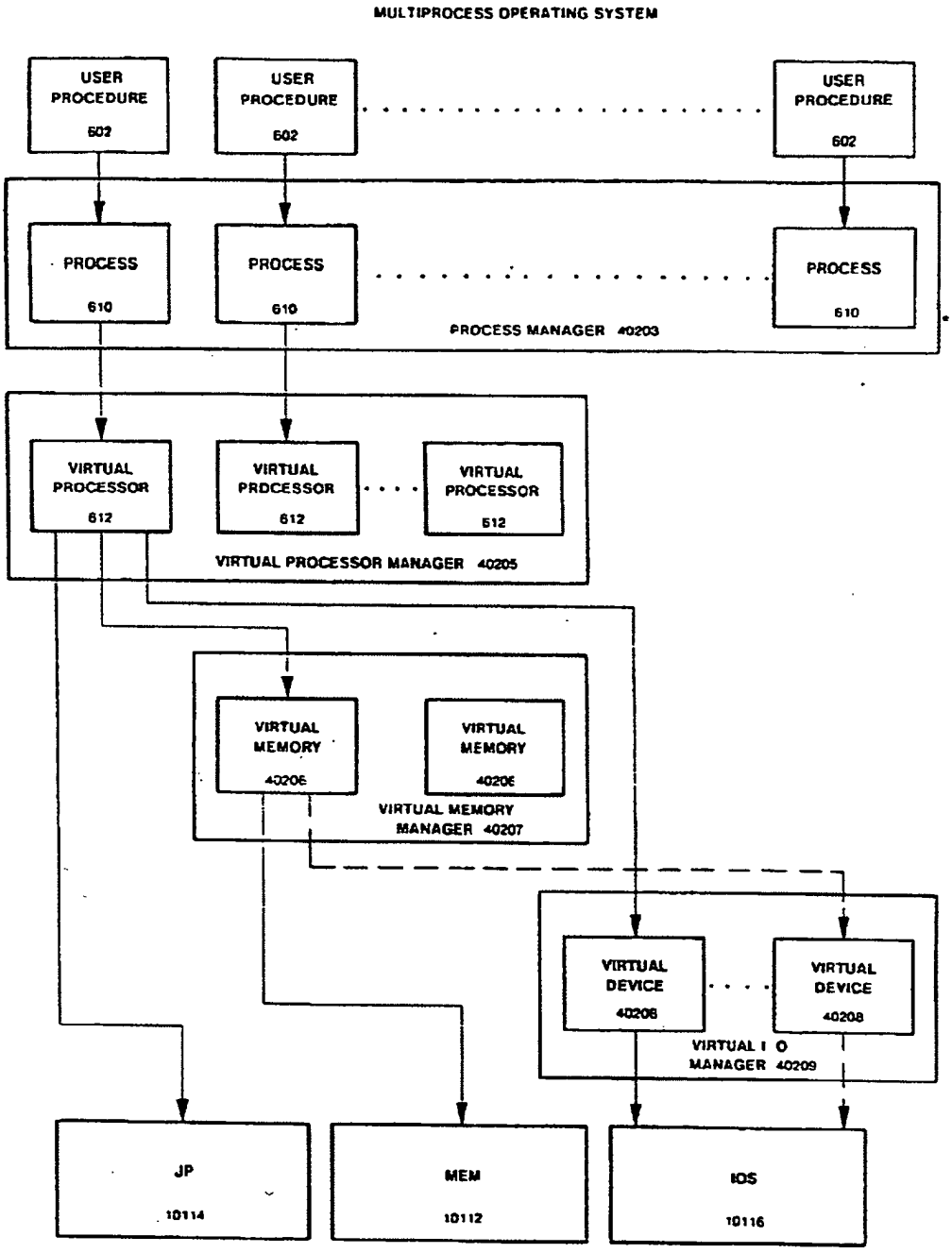


FIG 402

Petitioner Apple Inc. - Exhibit 1004, p. 5417

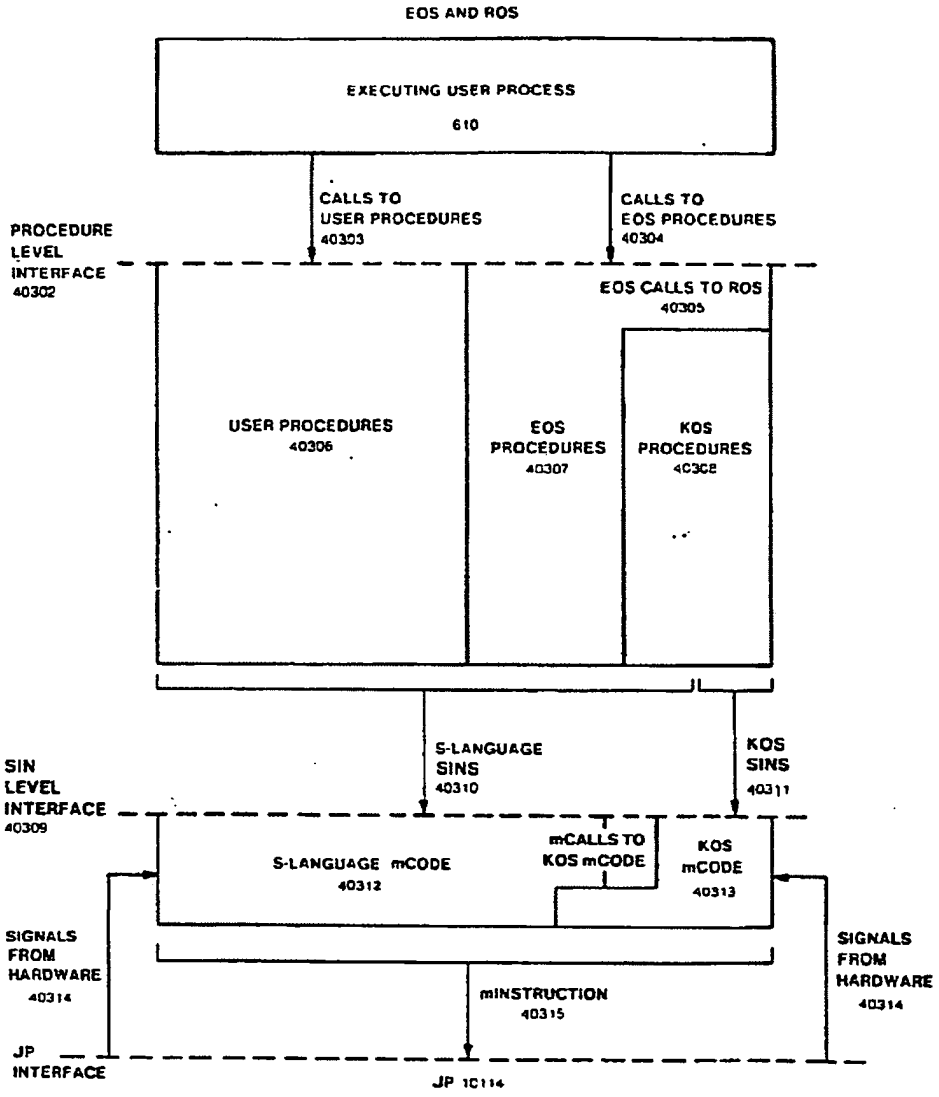


FIG. 403

Petitioner Apple Inc. - Exhibit 1004, p. 5418

EOS VIEW OF OBJECTS

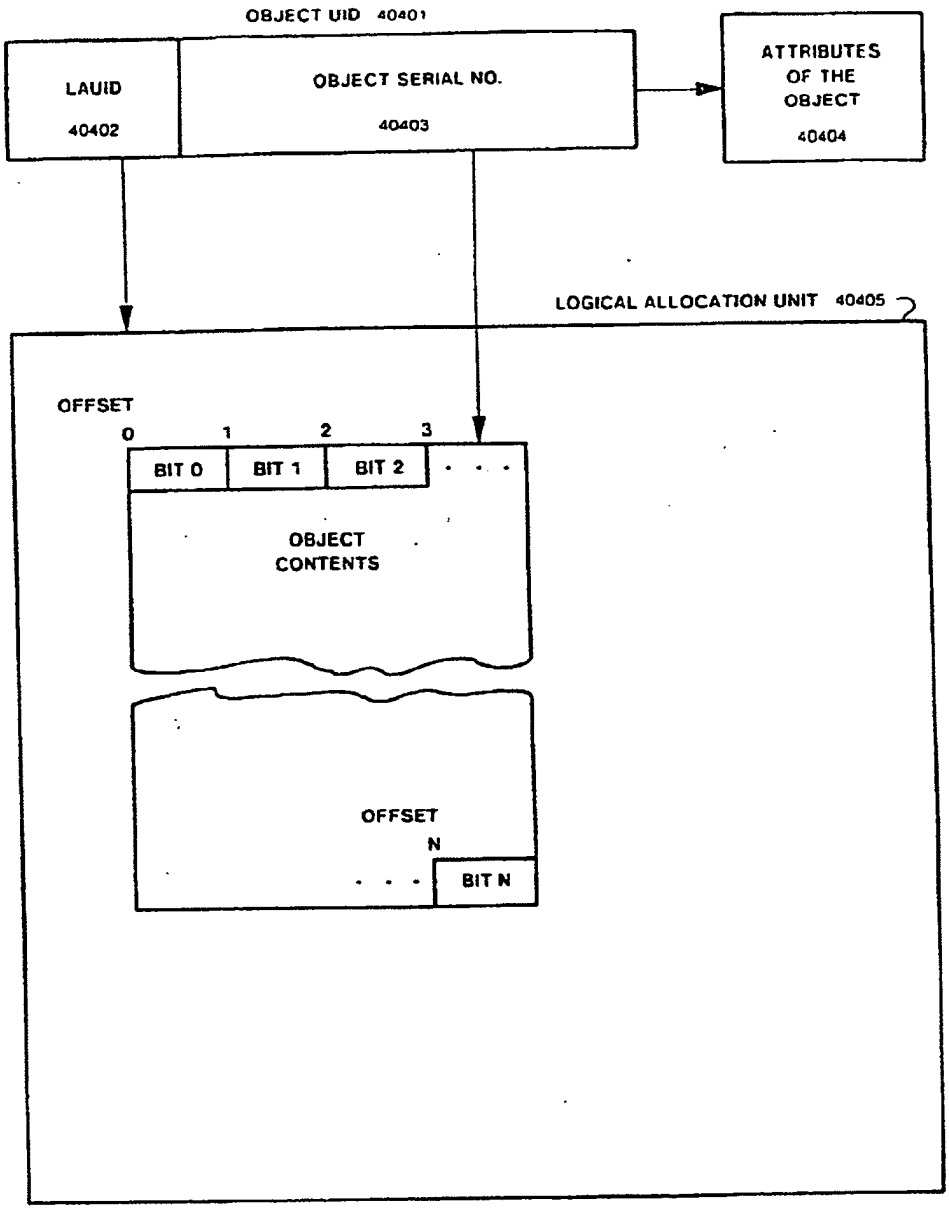


FIG 404

Petitioner Apple Inc. - Exhibit 1004, p. 5419

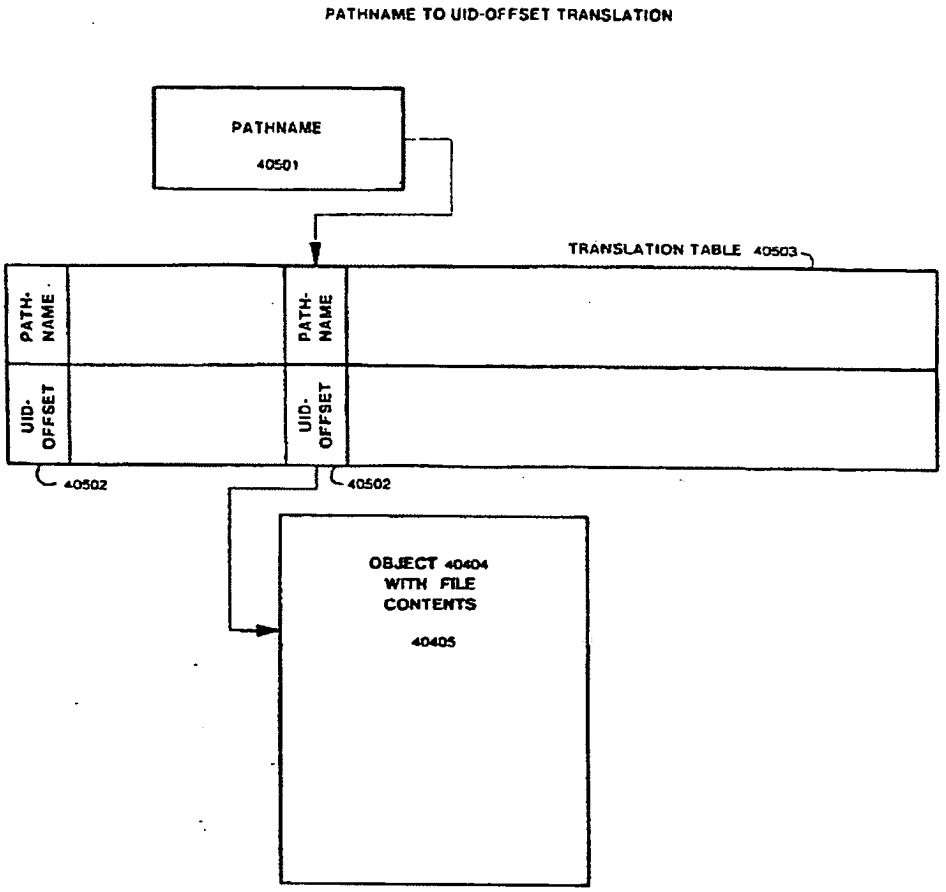


FIG 405

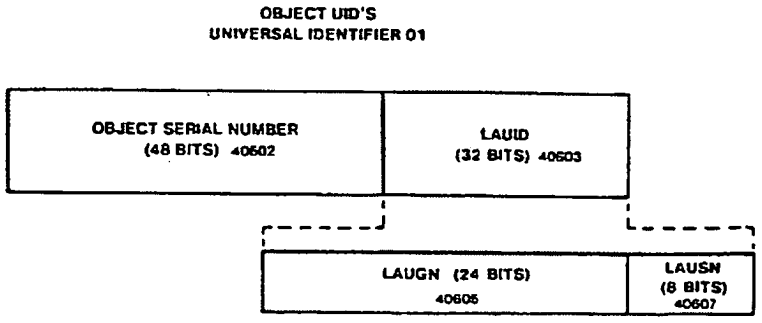


FIG 406

Petitioner Apple Inc. - Exhibit 1004, p. 5420

ATU, MHT, AND MEMORY

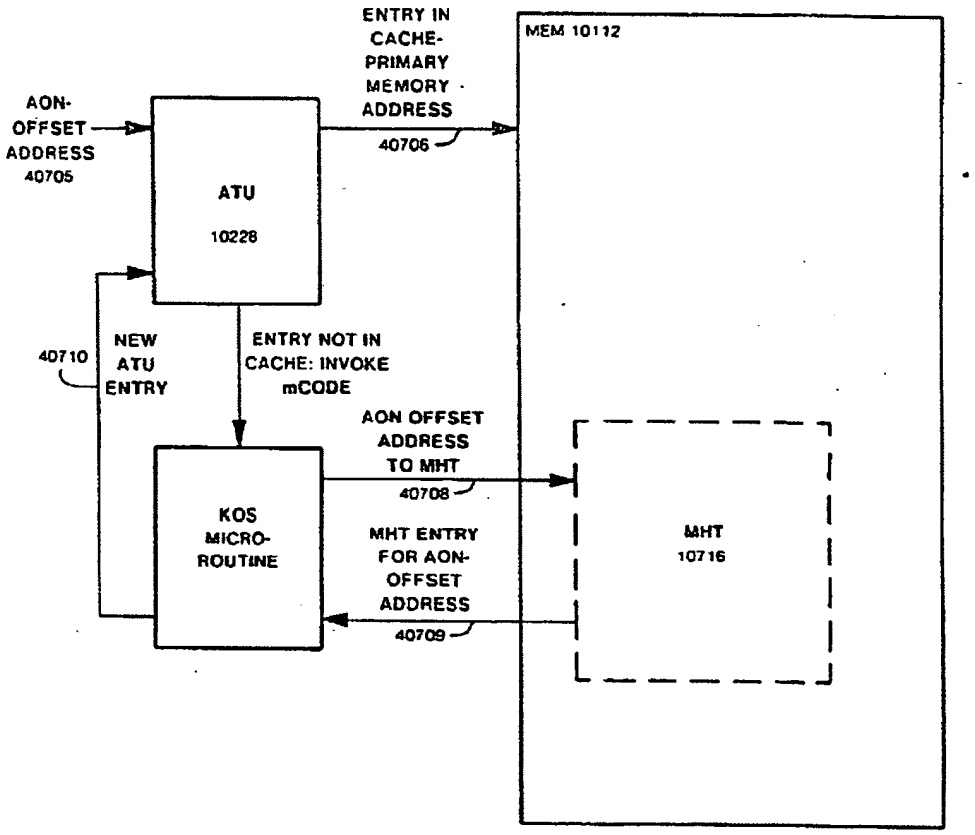


FIG 407

Petitioner Apple Inc. - Exhibit 1004, p. 5421

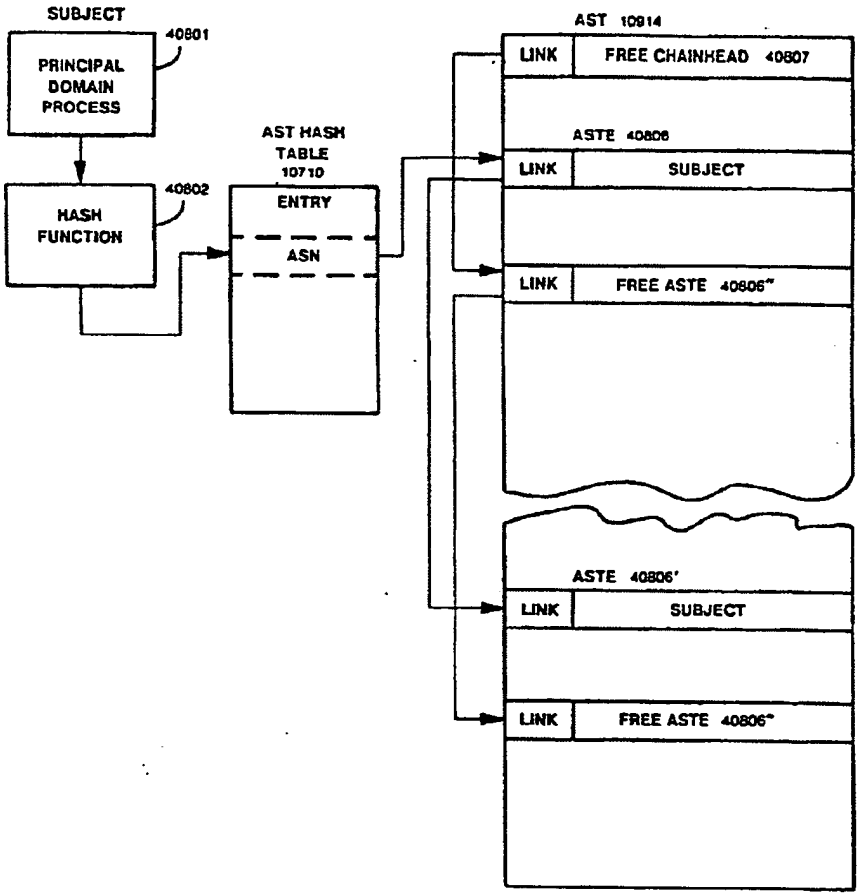


FIG 408

Petitioner Apple Inc. - Exhibit 1004, p. 5422

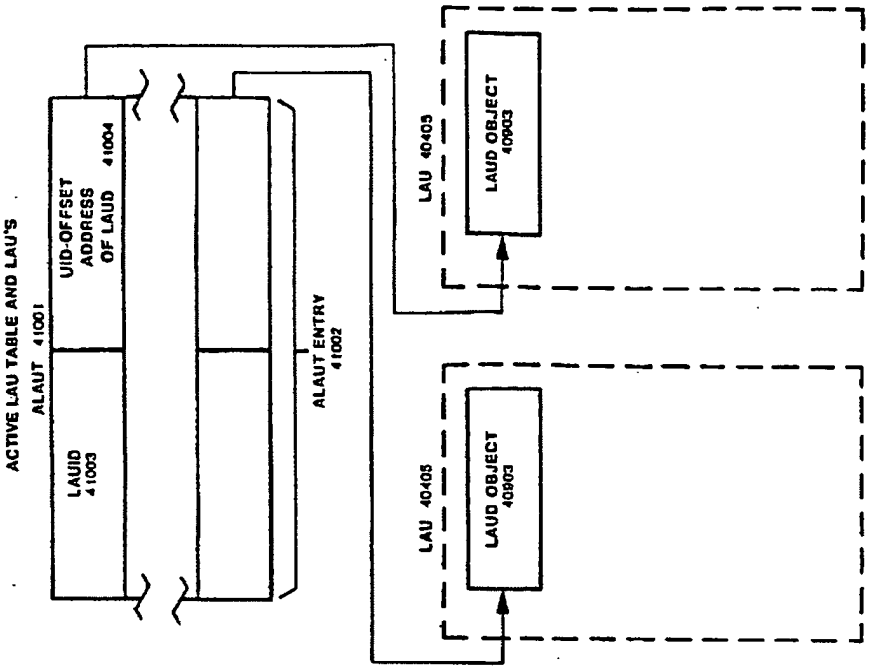


FIG 410

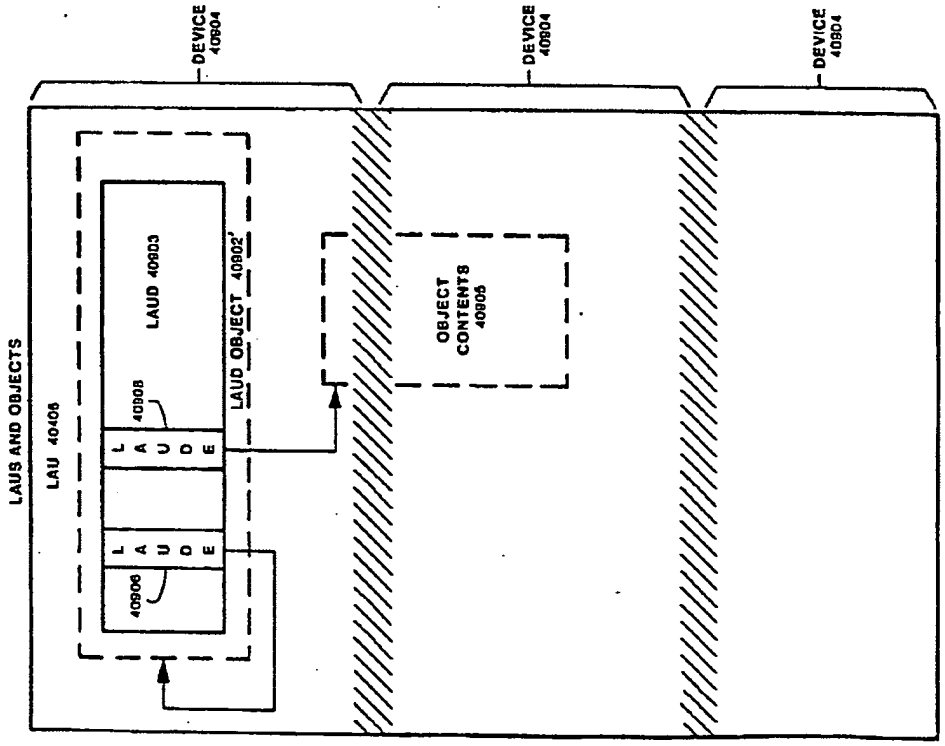


FIG 409

Petitioner Apple Inc. - Exhibit 1004, p. 5423

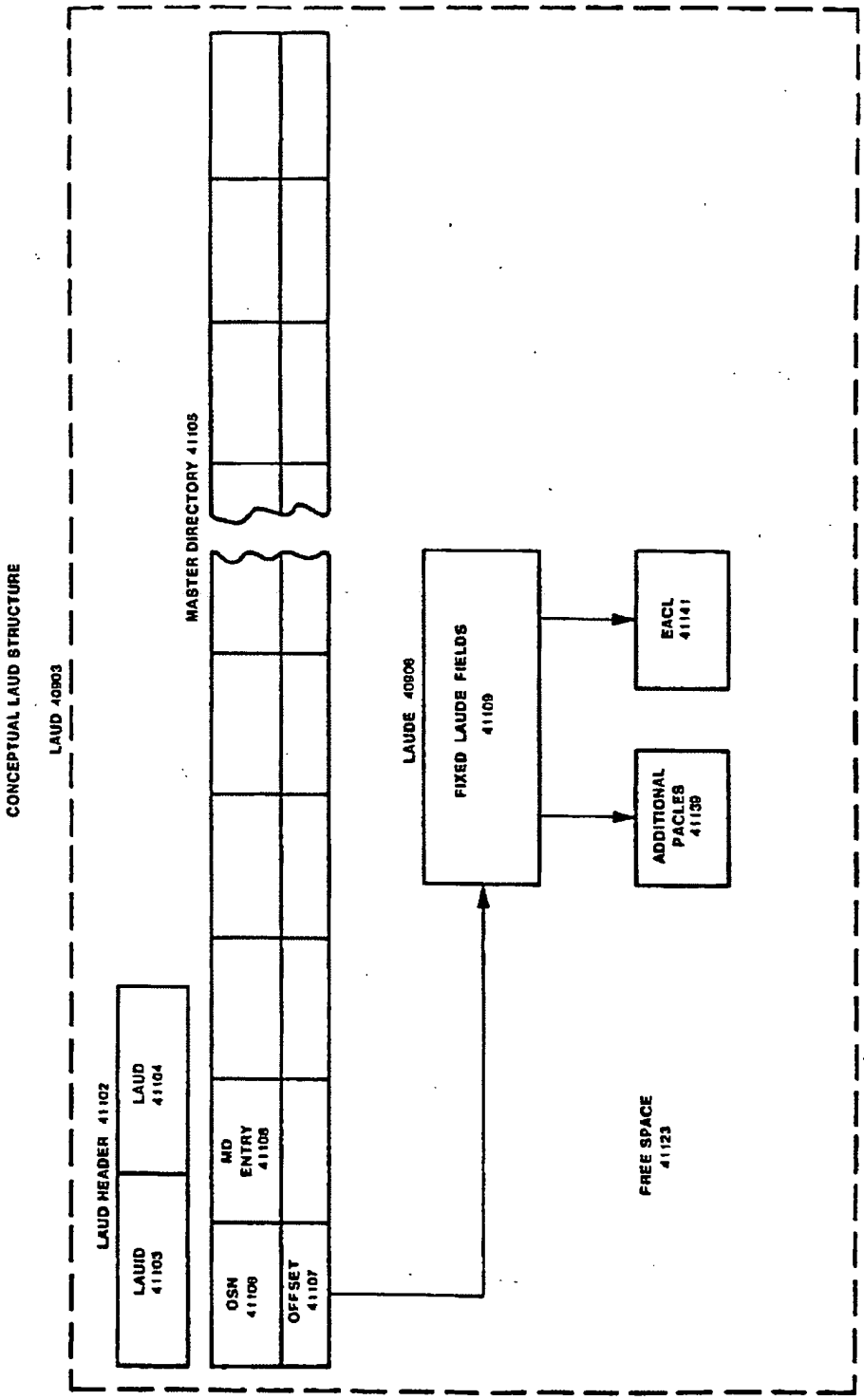


FIG 411

Petitioner Apple Inc. - Exhibit 1004, p. 5424

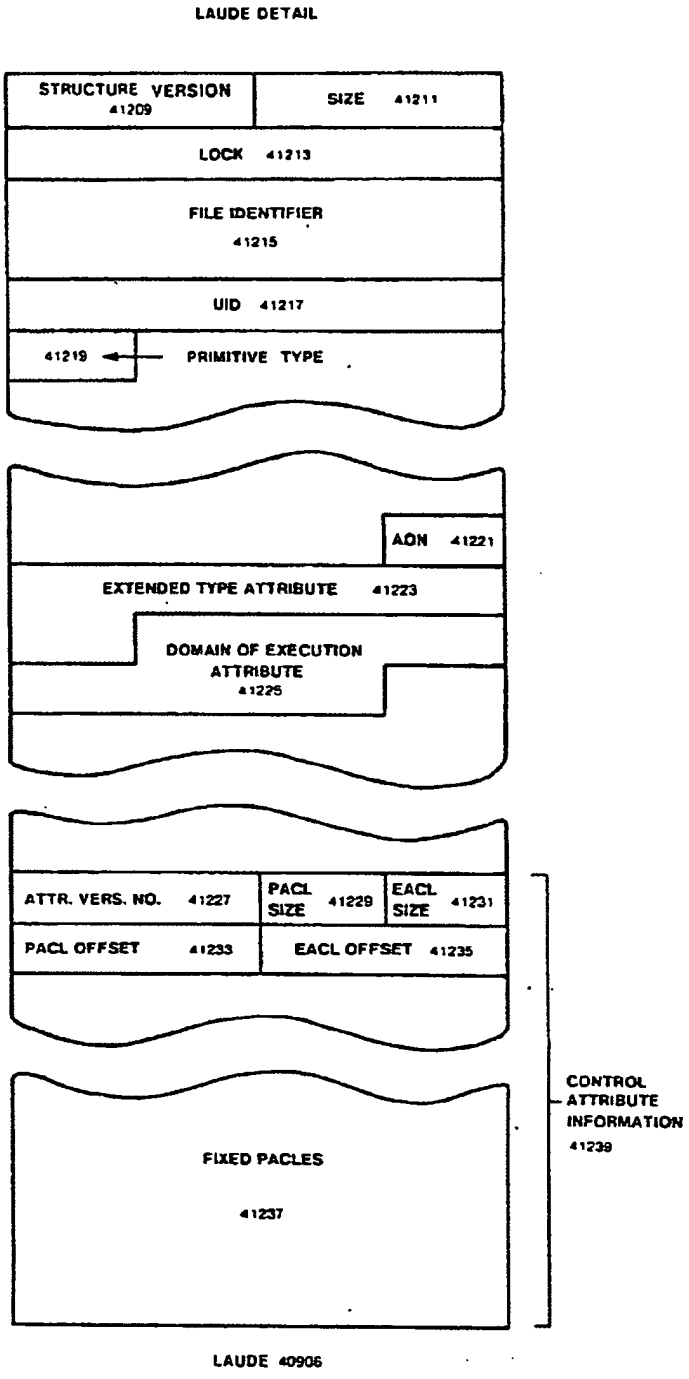


FIG. 412

Petitioner Apple Inc. - Exhibit 1004, p. 5425

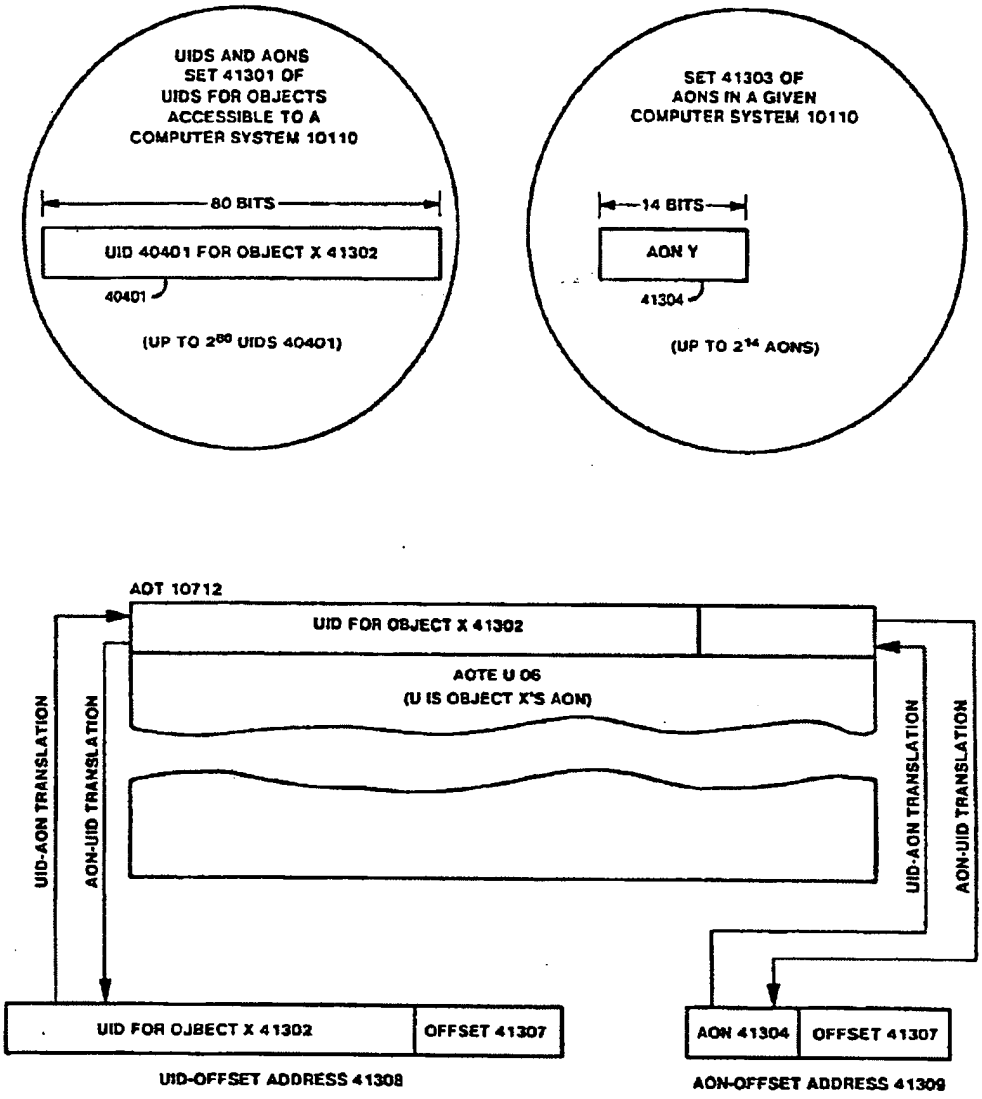


FIG 413

Petitioner Apple Inc. - Exhibit 1004, p. 5426

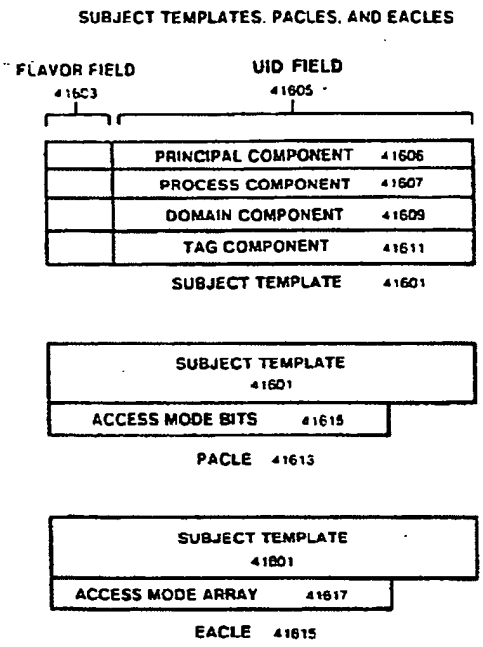


FIG. 416

Petitioner Apple Inc. - Exhibit 1004, p. 5427

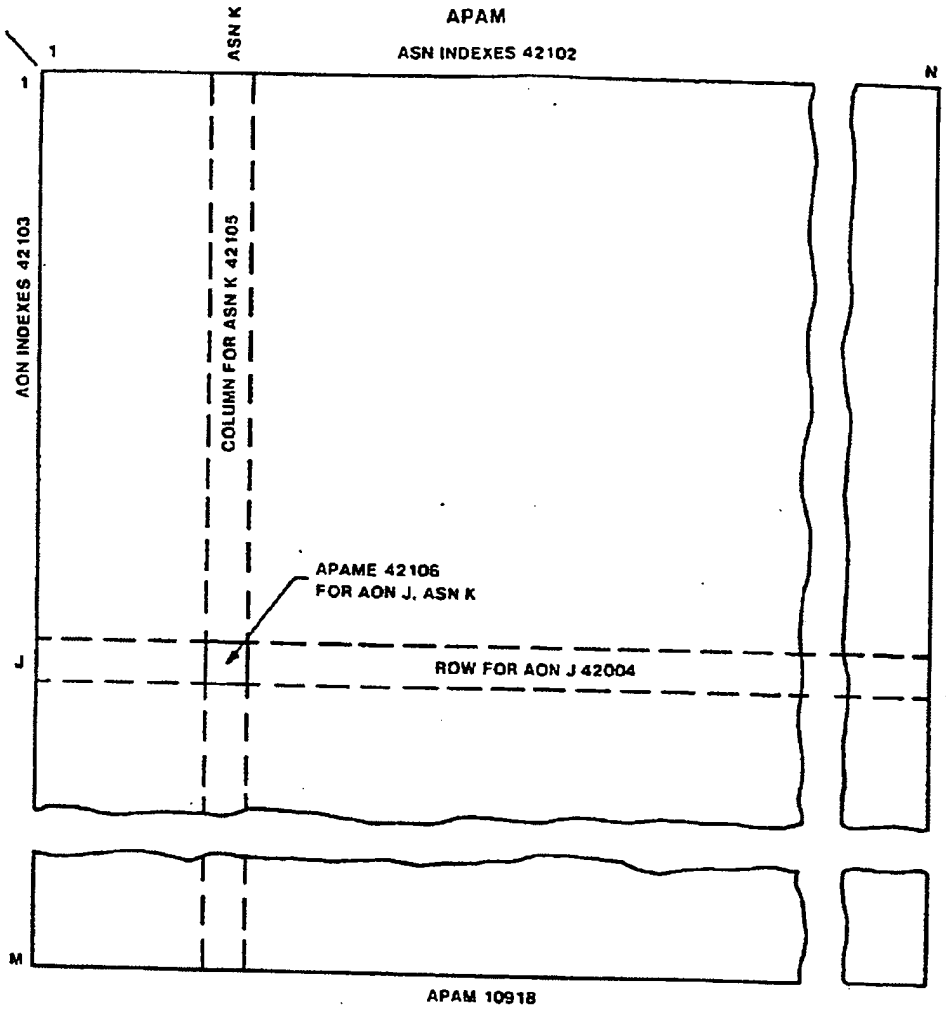


FIG. 421

Petitioner Apple Inc. - Exhibit 1004, p. 5428

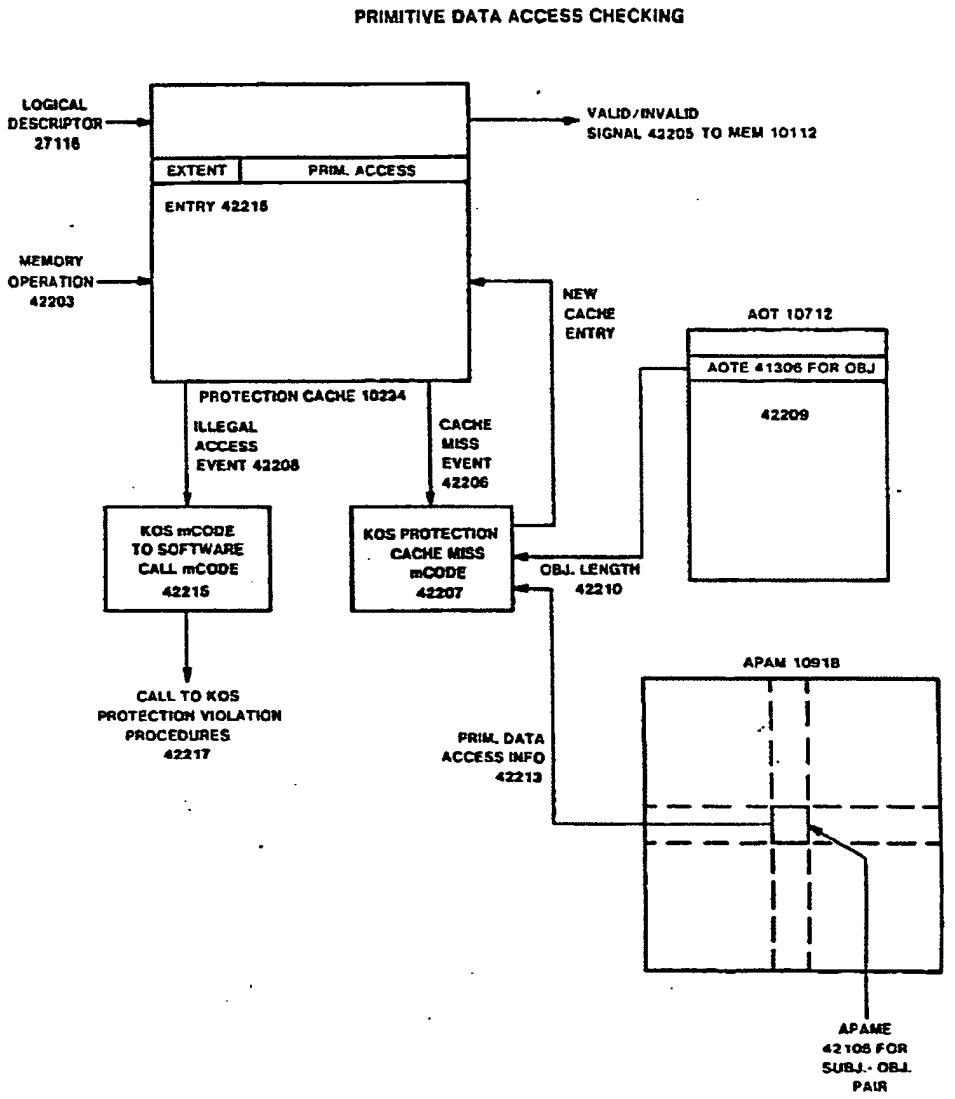


FIG. 422

Petitioner Apple Inc. - Exhibit 1004, p. 5429

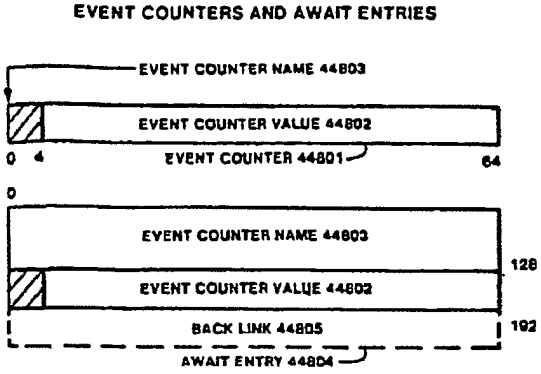


FIG. 448

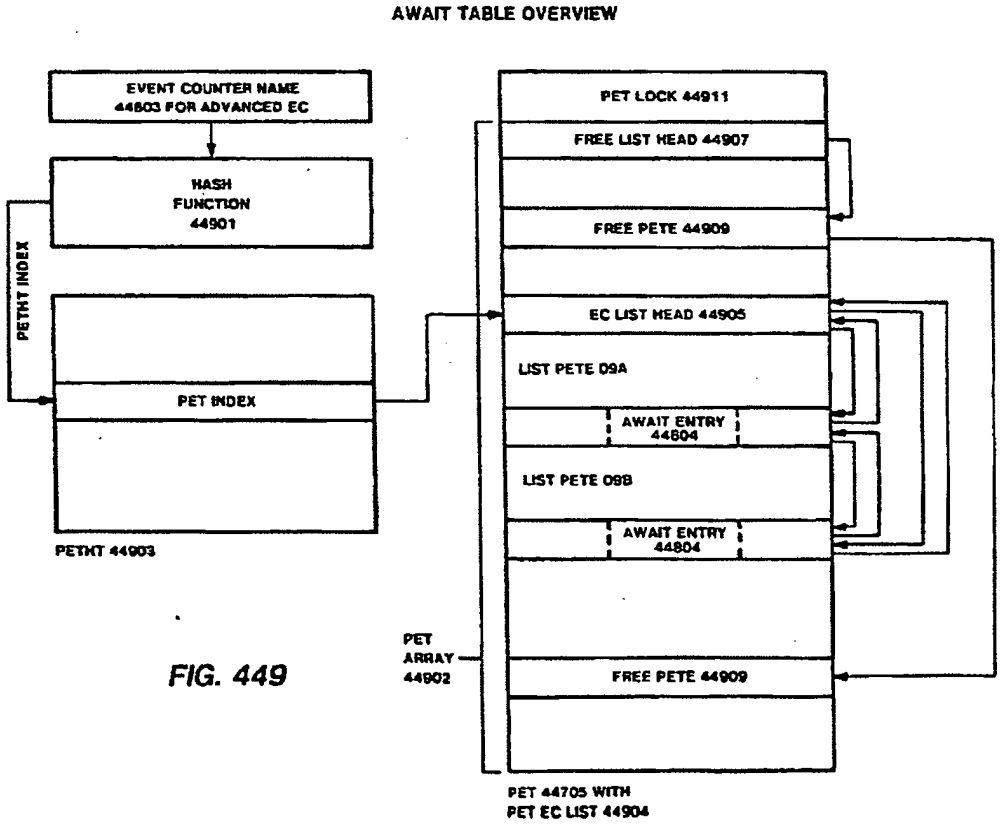


FIG. 449

Petitioner Apple Inc. - Exhibit 1004, p. 5430

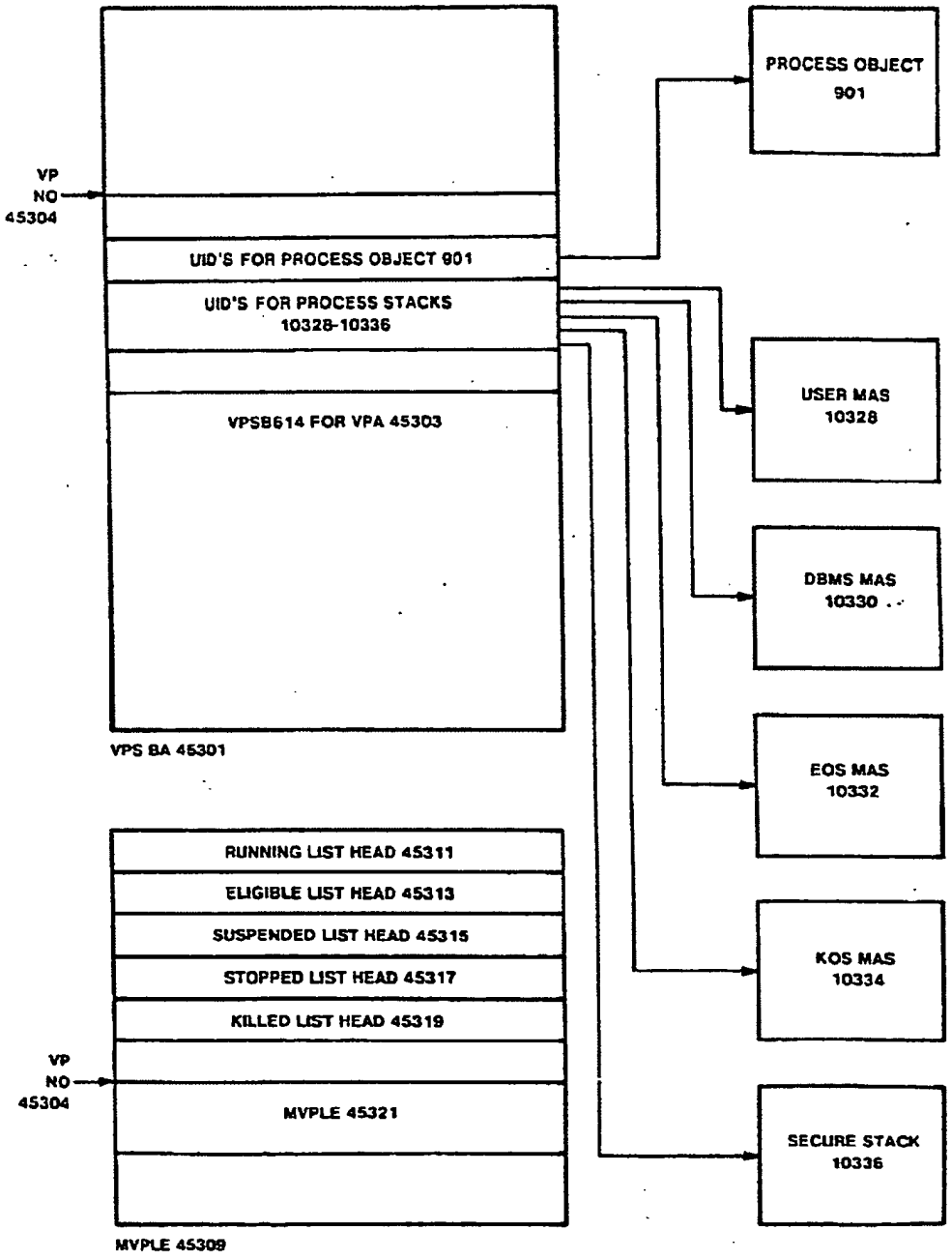


FIG. 453

Petitioner Apple Inc. - Exhibit 1004, p. 5431

VIRTUAL PROCESSOR SYNCHRONIZATION

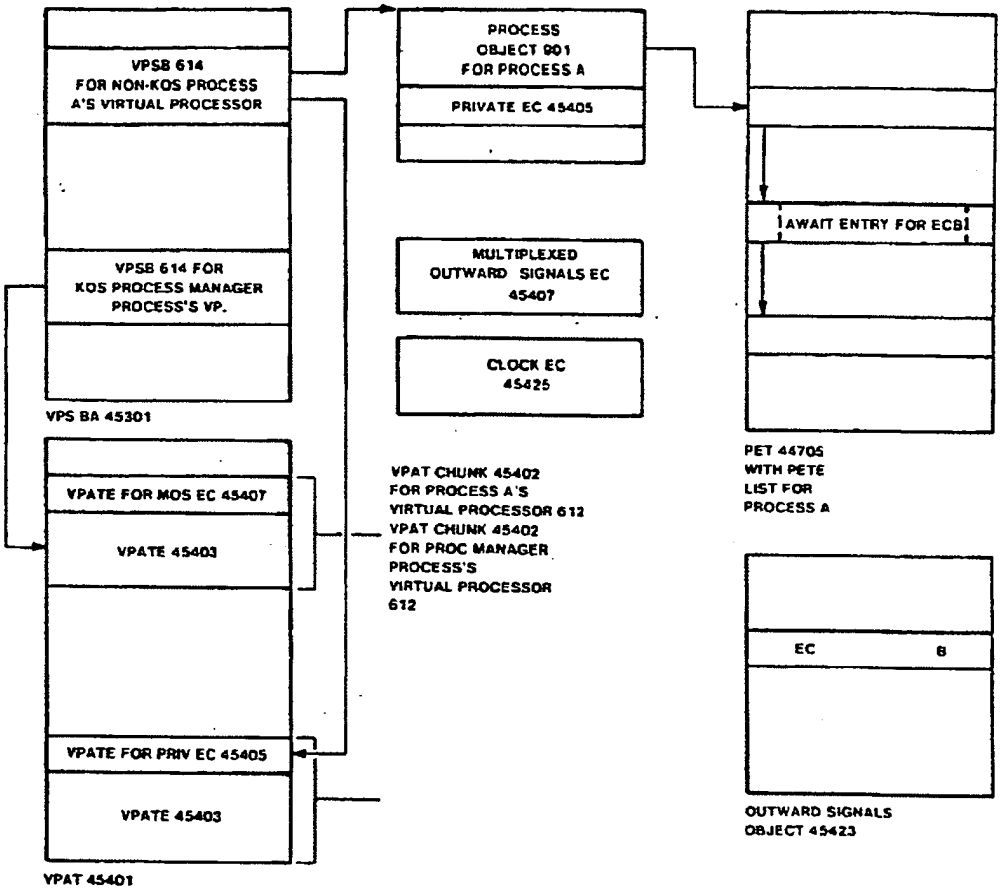


FIG. 454

Petitioner Apple Inc. - Exhibit 1004, p. 5432

MAS OBJECT OVERVIEW

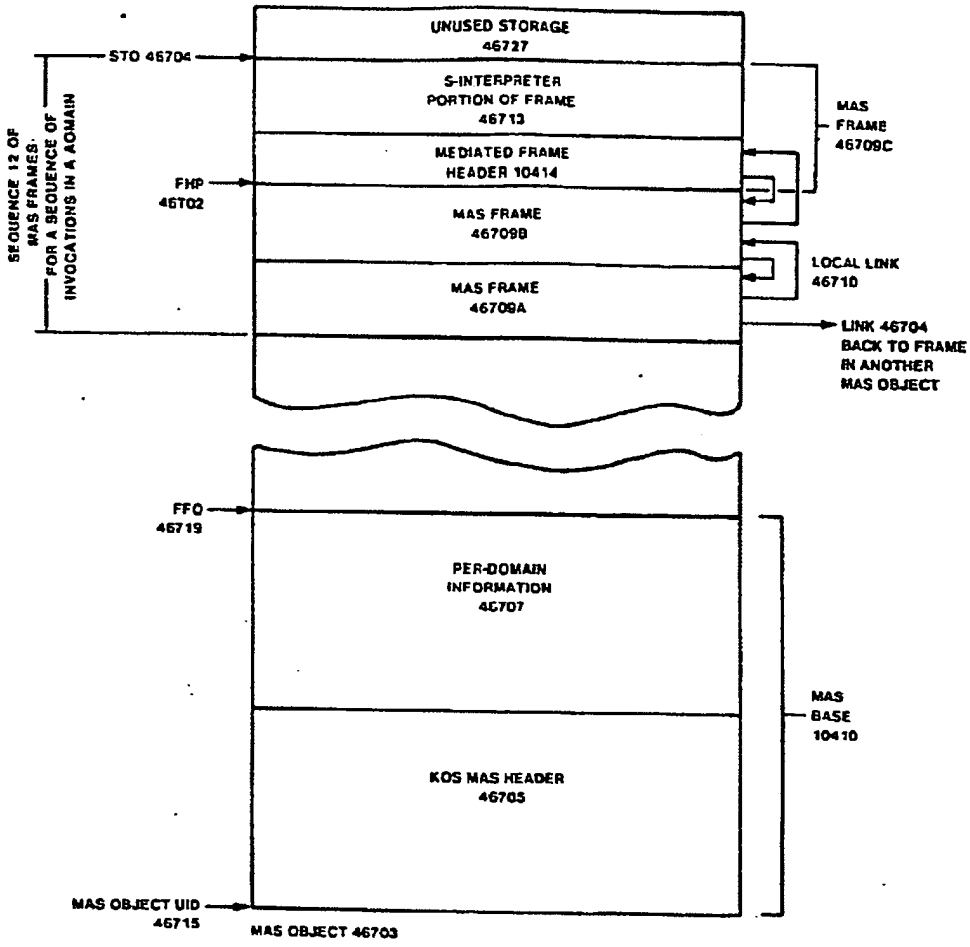


FIG. 467

Petitioner Apple Inc. - Exhibit 1004, p. 5433

MAS BOSE DETAIL

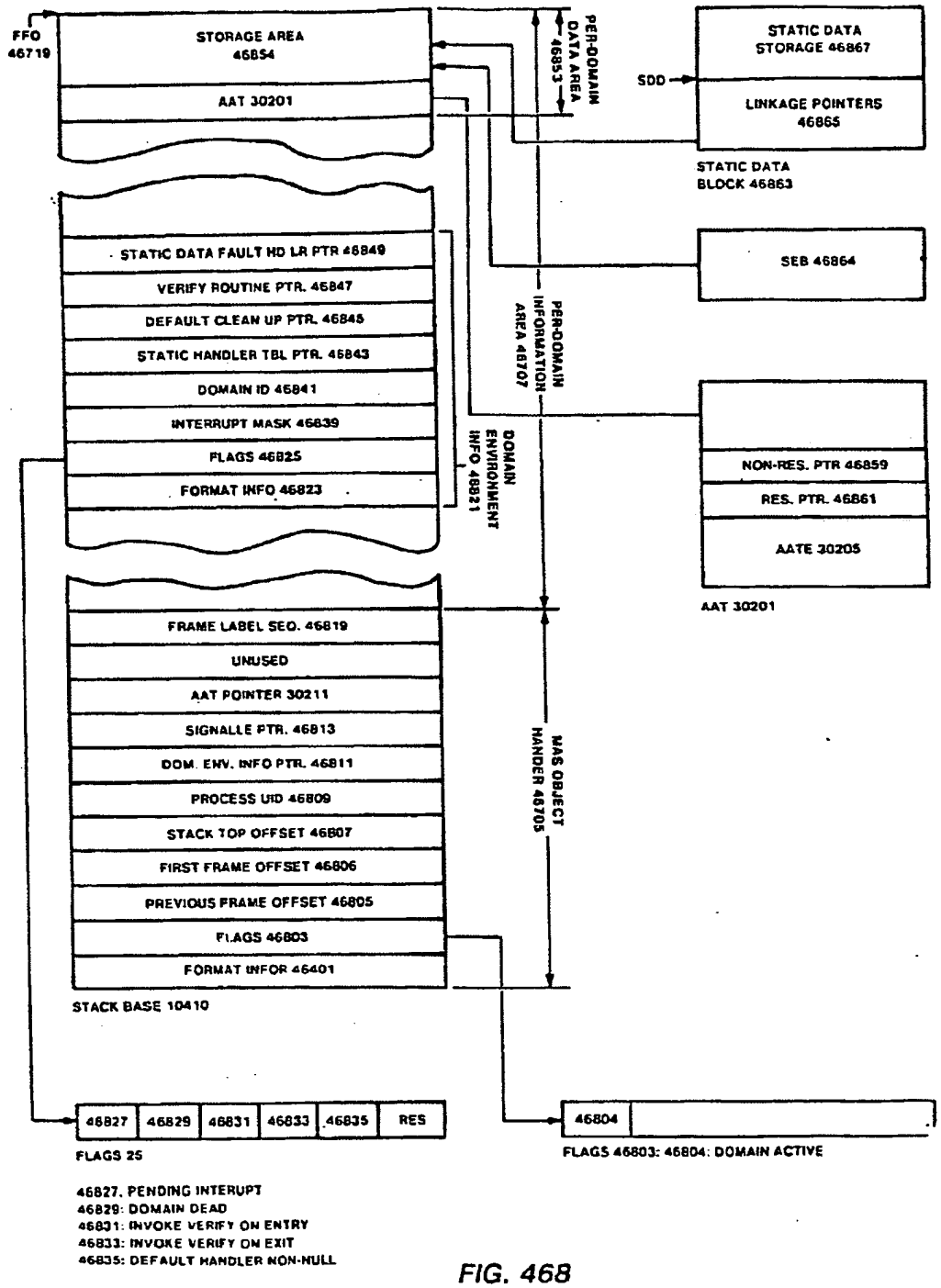


FIG. 468

Petitioner Apple Inc. - Exhibit 1004, p. 5434

MAS FRAME DETAIL

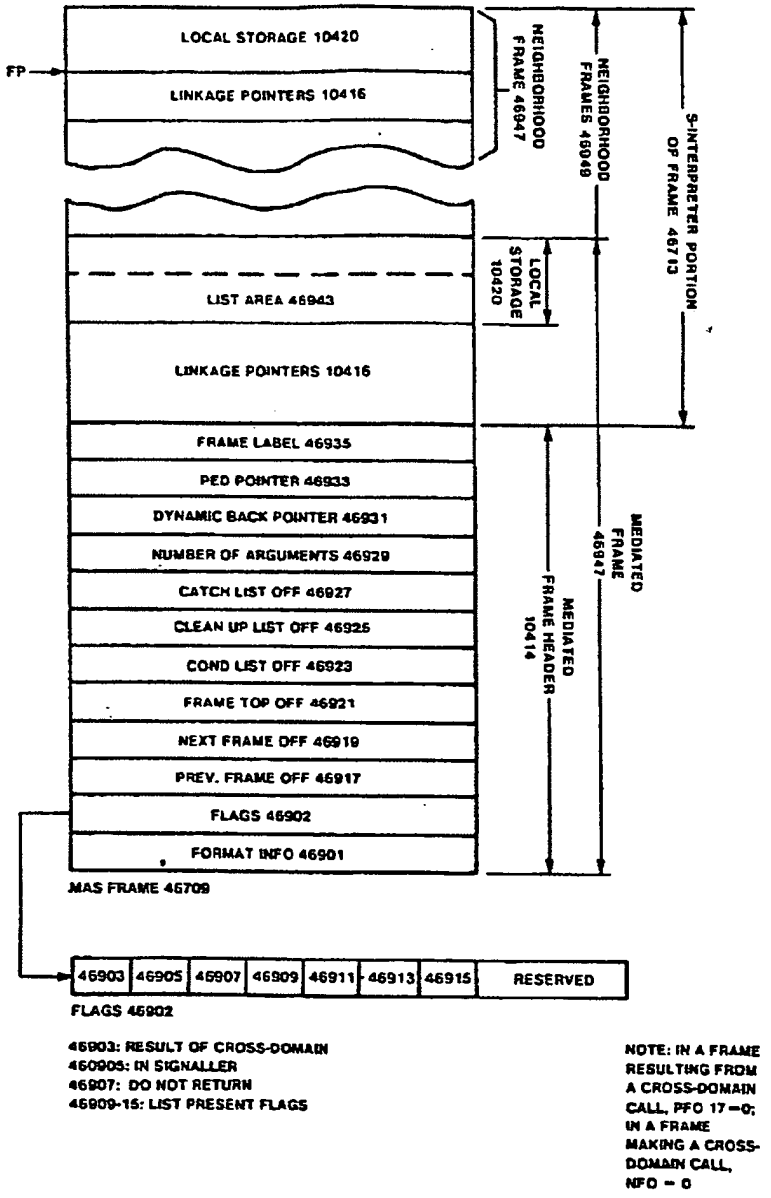


FIG. 469

Petitioner Apple Inc. - Exhibit 1004, p. 5435

SS 10336 OVERVIEW

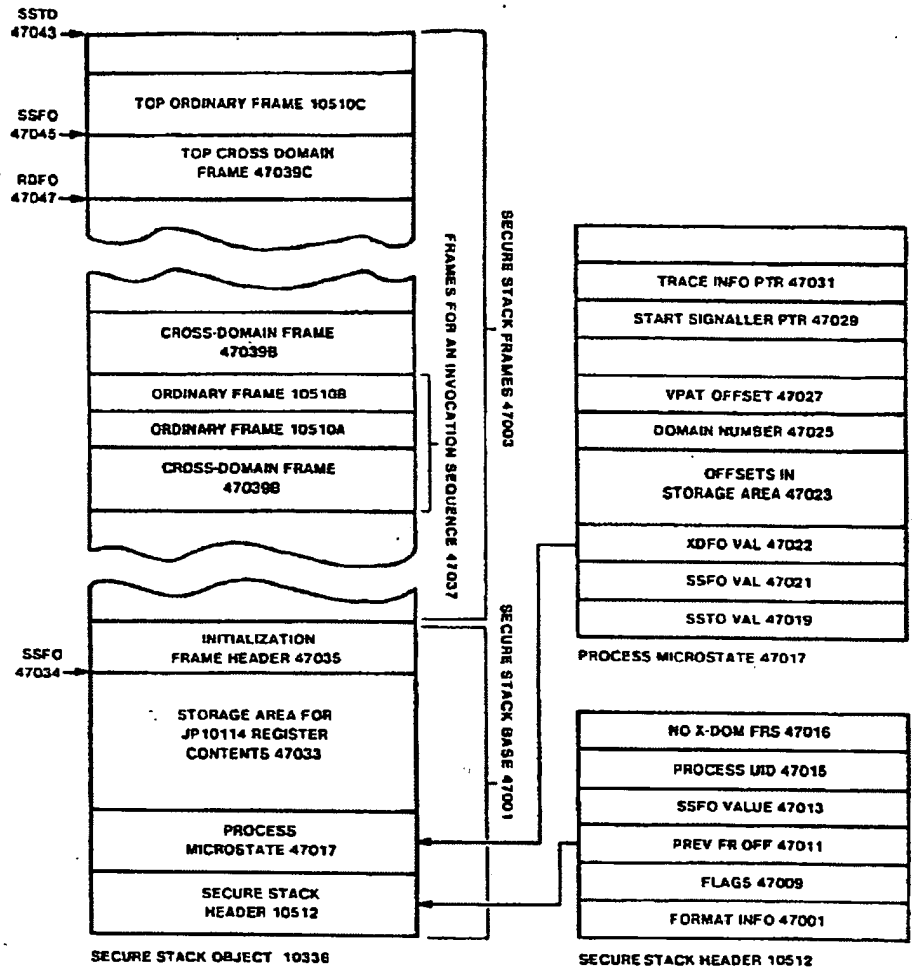


FIG. 470

Petitioner Apple Inc. - Exhibit 1004, p. 5436

SECURE STACK 10336 FRAME DETAIL

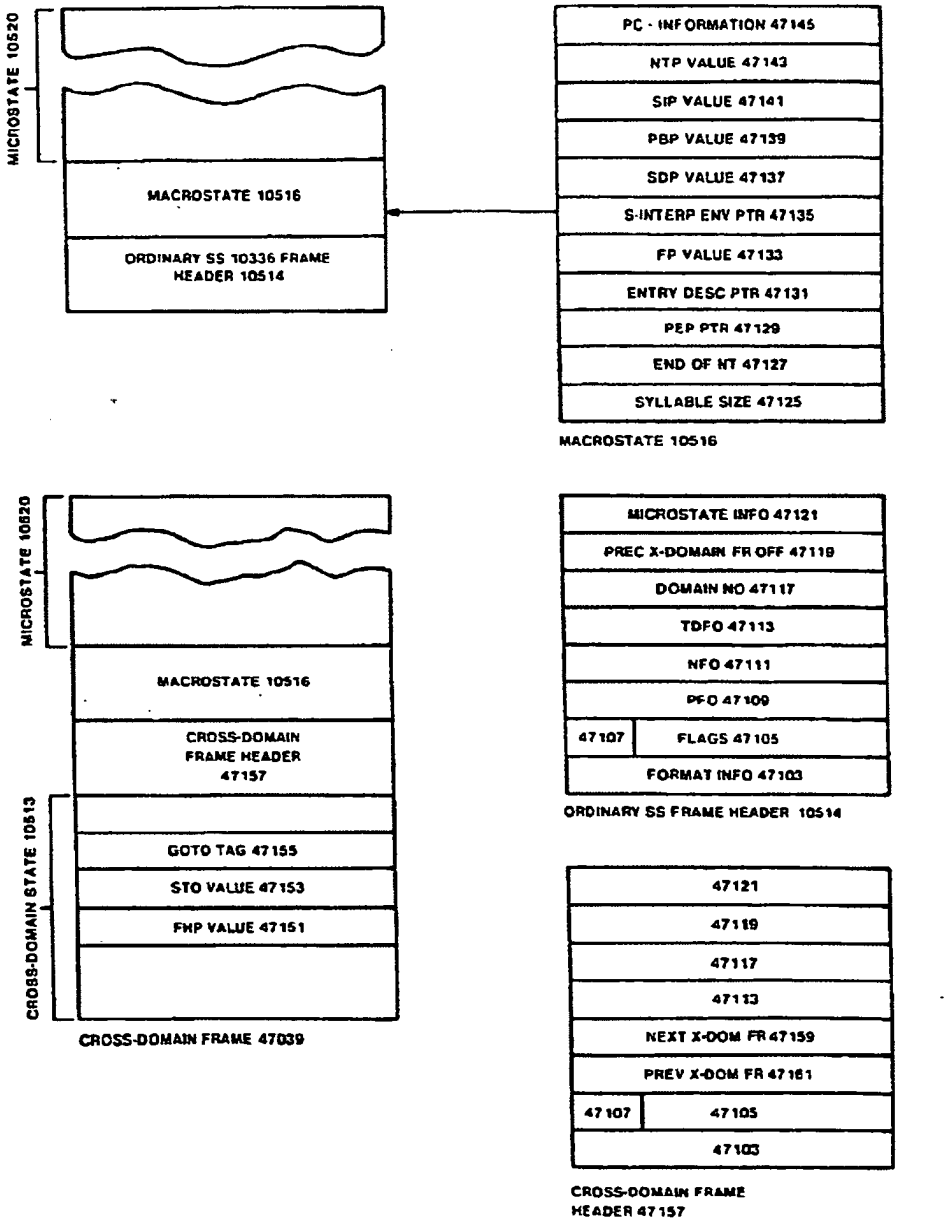


FIG. 471

Petitioner Apple Inc. - Exhibit 1004, p. 5437

PROCEDURE OBJECT OVERVIEW

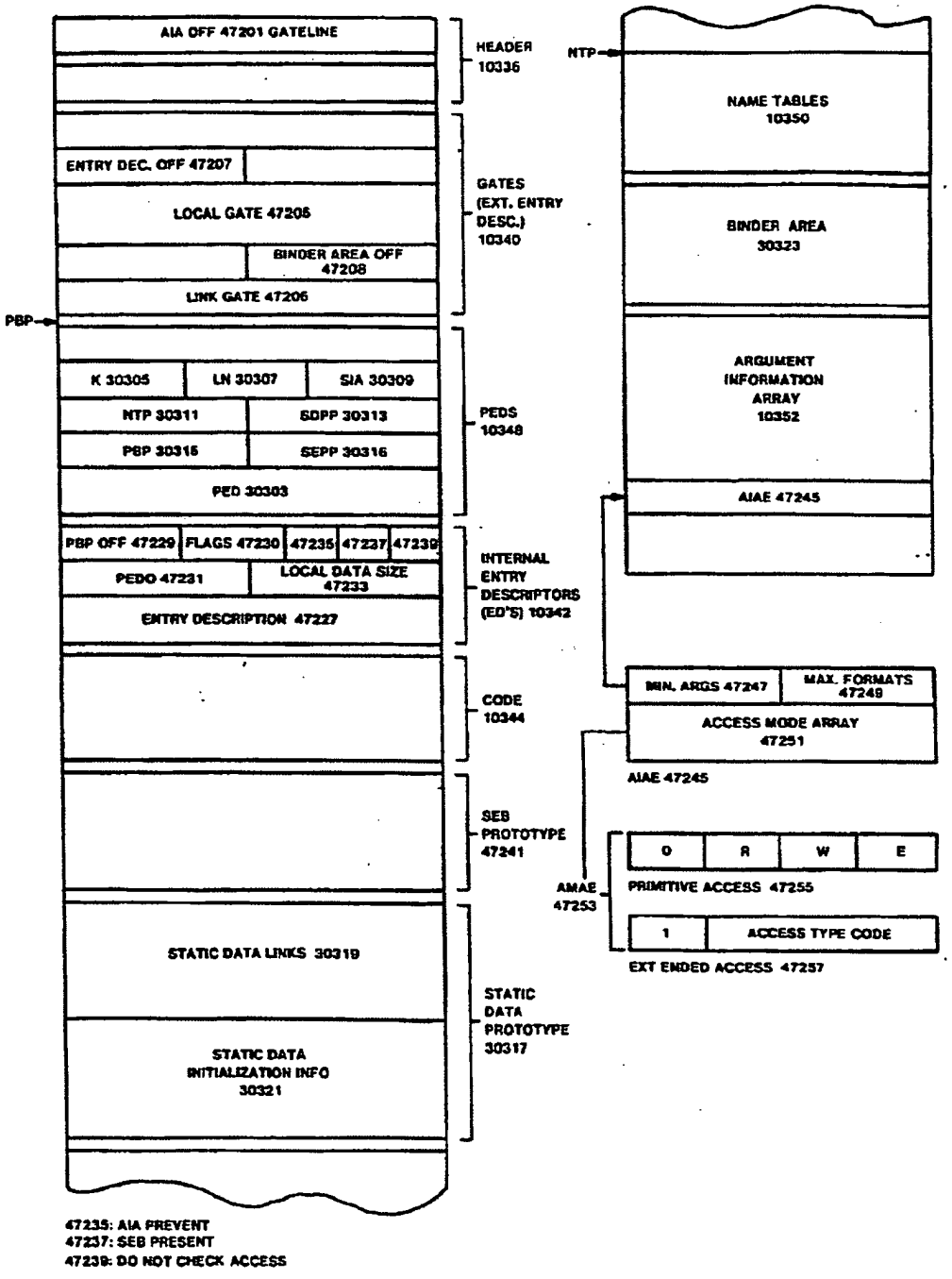


FIG. 472

Petitioner Apple Inc. - Exhibit 1004, p. 5438

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/60 — A2

(11) International Publication Number: **WO 99/24928**

(43) International Publication Date: 20 May 1999 (20.05.99)

(21) International Application Number: PCT/US98/23648

(22) International Filing Date: 6 November 1998 (06.11.98)

(30) Priority Data:
08/965,185 6 November 1997 (06.11.97) US

(71) Applicant: INTERTRUST TECHNOLOGIES CORP. [US/US]; 460 Oakmead Parkway, Sunnyvale, CA 94086 (US).

(72) Inventors: SHEAR, Victor, H.; 5203 Battery Lane, Bethesda, MD 20705 (US). VAN WIE, David, M.; Apartment 216, 965 East El Camino Real, Sunnyvale, CA 94087 (US). WEBER, Robert, P.; 215 Waverley Street #4, Menlo Park, CA 94025 (US).

(74) Agent: FARIS, Robert, W.; Nixon & Vanderhye P.C., 8th floor, 1100 N. Glebe Road, Arlington, VA 22201 (US).

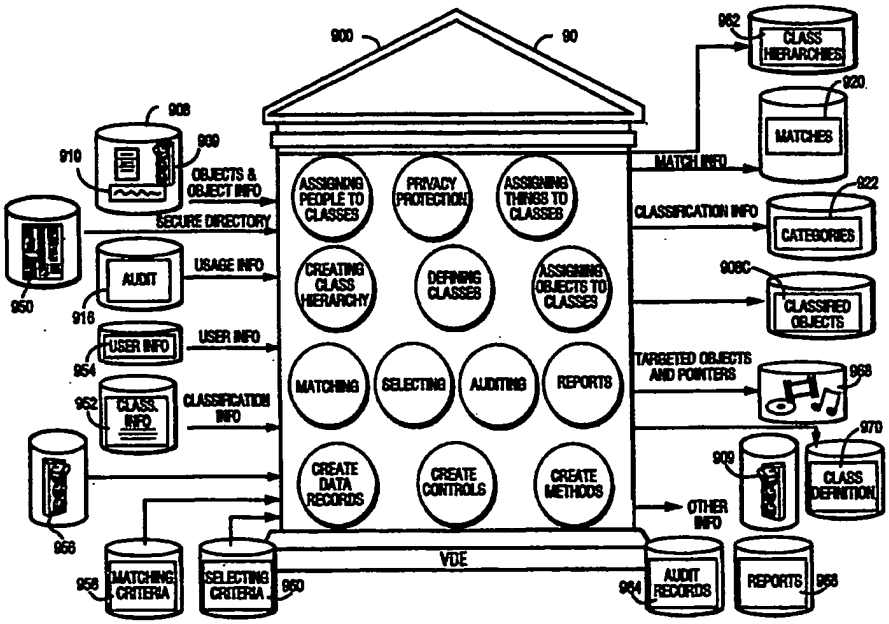
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: SYSTEMS AND METHODS FOR MATCHING, SELECTING, NARROWCASTING, AND/OR CLASSIFYING BASED ON RIGHTS MANAGEMENT AND/OR OTHER INFORMATION

(57) Abstract

Rights management information is used at least in part in a matching, narrowcasting, classifying and/or selecting process. A matching and classification utility system comprising a kind of Commerce Utility System is used to perform the matching, narrowcasting, classifying and/or selecting. The matching and classification utility system may match, narrowcast, classify and/or select people and/or things, non-limiting examples of which include software objects. The Matching and Classification Utility system may use any pre-existing classification schemes, including at least some rights management information and/or other qualitative and/or parameter data indicating and/or defining classes, classification systems, class hierarchies, categroy schemes, class assignments, category assignments, and/or class membership. The Matching and Classification Utility may also use at least some rights management information together with any artificial intelligence, expert system, statistical, computational, manual, or any other means to define new classes, class hierarchies, classification systems, category schemes, and/or assign persons, things, and/or groups of persons and/or things to at least one class.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

SYSTEMS AND METHODS FOR MATCHING, SELECTING, NARROWCASTING, AND/OR CLASSIFYING BASED ON RIGHTS MANAGEMENT AND/OR OTHER INFORMATION

FIELDS OF THE INVENTIONS

The inventions relate to electronic rights and transaction management. More particularly, the inventions relate to automated systems, methods and techniques for efficiently matching, selecting, narrowcasting, categorizing and/or classifying in a distributed electronic rights and/or other event and/or transaction management environment. For example, the inventions provide electronic computer based systems, methods and techniques for matching, classifying, narrowcasting, and/or selecting digital information describing people and/or other things. This matching, classifying, narrowcasting, and/or selecting can be based, at least in part, on elements of rights management information and/or one or more other categories of information -- wherein such information is used for efficient, trusted event management assuring the execution of one or more controls related to, including, for example, consequences of processing such digital information describing people and/or other things. The present inventions also provide systems and methods for efficiently determining class hierarchies, classification schemes, categories, and/or category schemes and/or the assignment of objects, persons and/or things to said class hierarchies, classification schemes,

categories, and/or category schemes using at least some rights management information.

BACKGROUND AND SUMMARY OF THE INVENTIONS

The modern world gives us a tremendous variety and range of options and choices. Cable and satellite television delivers hundreds of different television channels each carrying a different program. The radio dial is crowded with different radio stations offering all kinds of music, news, talk, and anything else one may care to listen to. The corner convenience store carries newspapers from around the country, and a well stocked newsstand allows you to choose between hundreds of magazines and publications about nearly every subject you can think of. Merchandise from all corners of the world is readily available at the shopping mall or by mail order. You can pay by check, in cash, or using any number of different kinds of credit cards and ATM cards.

This tremendous variety is good, but it also presents problems. Sometimes, it is hard or inefficient for us to find what we want and need because there are too many things to evaluate and choose from, and they are often located in too many places. We can waste a lot of time searching for the things we need or want at the right price, with the rights features, and at a particular time.

Sometimes, we never find things that satisfy what we feel we need or want. This happens when we don't know what to look for,

how to look for it, or don't have the necessary assistance or tools to search successfully. For example, we may not know the best way of looking for something. Sometimes, we know what we are looking for but can't express or articulate it in ways that help us look. And sometimes, we don't even know what we are looking for. You may know you need something, know its missing, but never really know how to communicate to others what you are looking for. For example, someone who speaks only English may never find resources using Japanese or Spanish. In general, we often don't have the time or resources to look for all the things that would give us the most benefit or make us the most satisfied.

It's Hard To Find Mass Media Things You Want Or Need.

Figure 1A shows, as one example, how frustrating it can be to find anything to watch on the hundreds of television channels that may be available. The man in Figure 1A spends a lot of time "channel surfing," trying to find something he is interested in watching. He may be moderately interested in golf, but may not like the particular golf tournament or golf players being broadcast at 7 o'clock on a particular channel. After flipping through other channels, he might think an action movie looks interesting only to find out after watching it for a while that he isn't really interested in it after all. A documentary on horses also seems interesting at first, but he finds it boring after watching it awhile because it doesn't give him the kind of information he is interested in. The whole process can be

frustrating and he may feel he wasted a lot of time. Figure 1B shows the man getting so frustrated at the wasted time and energy that he thinks that maybe watching television is just not worth it . What the man really needs is a powerful yet efficient way to find those things that most satisfy his desires -- that is, match his needs and/or his interests.

Our Mail Overloads Us With Things We Don't Want or Need

The same thing can happen with information sent to us in the mail. It can be fun to receive some kinds of mail, such as personal letters, or magazines and catalogs on topics of personal interest. Certain other mail, such as bills, may not be fun but are usually important. Unfortunately, our mailboxes are typically overflowing with yet another kind of mail commonly referred to as "junk mail." The person in Figure 2 finds his mailbox stuffed to the overflowing point with mail he never asked for and has absolutely no interest in. Most of this junk mail ends up unread and in the trash. However, it can take a long time to sort through all this mail to be sure you are only throwing out only the junk mail and not the good mail you are interested in or need. For example, it's sometimes hard to distinguish credit card bills from offers for new credit cards you don't need or want. Wouldn't it be useful if your mail could be automatically "cleaned" of the mail you had no interest in and you received only the mail you wanted or needed?

Sorting through things to identify things you might want, then selecting what you actually want, can be a frustrating and time consuming experience. For example, it wastes the time of the person who receives the junk mail, and it also wastes the time, money and effort of the people who spend their money to send mail to people hoping that they will buy their products.

As frustrating as finding and selecting may be to consumers, they often create even greater problems for businesses and people who want to locate or provide information, goods and services. It is often said, that in the world of business, "Information is Power" and "efficiency is the key to success." To find or sell the most relevant or useful information and to provide the ability to most efficiently allow business to operate at its best, we need easy-to-use tools that can help us navigate, locate, and select what matches our interests. In the modern world, it is often difficult to find out what different people like, and to supply people with the opportunity to select the best or most satisfying choices.

Past attempts outside the computer world to match up people with information, goods and/or services have had limited success. For example, attempts to "target" mass mailings may increase the chance that they will go to people who are interested in them, but the entire process is still very wasteful and inefficient. It is considered a good success rate to match the interests of only a few percent of the recipients of "junk" mail. Telemarketing campaigns that use the

telephone to reach potential consumers can be very expensive, very annoying to consumers who are not interested in the products being marketed, and very costly and inefficient. A much more ideal situation for all concerned is enabling businesses to send information only to individual consumers likely to find the information interesting, desirable, convincing, and/or otherwise useful. That way, businesses save time and money and consumers aren't unproductively hassled by information, phone calls, junk mail, junk e-mail and the like. However, right now it is extremely difficult to accomplish this goal, and so businesses continue to annoy consumers while wasting their own time, money, and effort.

Because of the Vast Amount of Information Available, Even Systems that Provide a High Degree of Organization May Be Difficult to Use or Access

You can find yourself wasting a lot of time finding things -- even in places where finding things is supposed to be easy. For example, a library is a place where you can find all sorts of useful information but can also waste a lot of time trying to find what you are looking for. Modern libraries can be huge, containing tens or even hundreds of thousands or millions of different books, magazines, newspapers, video tapes, audio tapes, disks, and other publications. Most libraries have an electronic or manual card catalog that classifies and indexes all of those books and other materials. This classification system is useful, but it often has significant limitations.

For example, normally a card catalog will classify materials based only on a few characteristics (for example, general subject, author and title). The boy in Figure 3 is looking for information on American League baseball teams during World War II for a high school report. The card catalog led to the general subject of baseball and other sports, but, looking at the catalog, he can't identify any books that seem to provide the specific information he wants to see, so he must rely on books classified as "histories of sports" or "histories of baseball." He can spend lots of time looking through the books on the shelves, going back to the card catalog, and going back to the shelves before he finds a reference that's reasonably helpful. He may need to go ask an expert (the librarian) who is familiar with the books the library has on sports and may know where to look for the information. Even then, the boy may need to flip through many different books and magazines, and look in many different places within the library before he finds the information he is looking for.

Finding Products You Want or Need Can Be Very Difficult and Time Consuming

The same kind of frustrating experience can happen when you shop for a particular kind of item. While some people enjoy shopping, and have fun seeing what is in various stores, many people dislike spending time shopping, searching for the best or most affordable item. And sometimes even people who like to shop don't have the time to shop for a specific item.

For example, the man in Figure 4 goes into a shopping mall looking for a tie to fit very tall people. He didn't wear a tie to work that day, but, at the last minute, an important meeting was scheduled for later that day and he needs to dress up. The shopping mall has a large variety of stores, each selling a range of merchandise. But the man may only have a short time to look. For example, he may be on his lunch break, and needs to get back to work soon. He can't spend a lot of time shopping. He may therefore need to rely on tools to help him identify where he wants to buy the tie. Perhaps he uses a mall directory that classifies the different stores in terms of what kinds of merchandise they sell (for example, clothing, books, housewares, etc.). Perhaps he asks at the malls help desk staffed by "experts" who know what is available in the shopping mall. But even these resources may not tell him where to buy Italian silk ties that are discounted and cost $20. So he does the best he can with the available resources.

These Problems Are Worse in the Digital World

The electronic or digital world offers a rapidly growing, vast array of electronically published products and services. For example, computer superstores have a dizzying array of different software products. Furthermore, music is now published primarily in digital form on optical disks, and video will soon be published that way too. And, of particular interest related to certain of the inventions described by this document, the Internet now has millions of home pages with an overwhelmingly variety and quantity of digital

information, and, these millions of home pages, in turn, point or "link" to millions of other web pages as well.

Today, for example, you can use the Internet to:

- read electronic newspapers, books and magazines and see them on your computer screen;
- get music in electronic form and play it using your computer;
- send and receive electronic mail all over the world;
- download reports and other information compiled by governments, companies, industries, universities, and individuals;
- watch videos and animations;
- play games with "cyber-friends" located around the world;
- chat with individuals and groups who share at least some interests in common;
- participate in "virtual reality" worlds, games, and/or experiences;
- (offer to) buy, and/or (offer to) sell nearly anything; and
- conduct electronic transactions and commerce.

Today on the Internet and you can also find nearly anything and everything you can possibly imagine, although finding exactly what you really want may be time consuming and frustrating. This is

because the Internet and World Wide Web provide perhaps the best example of an environment that is particularly hard to navigate. There are an overwhelming number of choices -- too many to easily relate to or understand -- and many of which are terribly hard to find, even using the various Web searching "engines." The Internet is particularly exciting because it has the potential to provide to nearly everyone access to nearly every kind of information. Information can also come from an almost limitless variety of sources. But today, so much information on the Internet is superficial or useless, and too many choices can be more a curse than a blessing if you don't have meaningful, easy ways to eliminate all but a relatively few choices. And the situation will only become much worse as more Web sites appear, and as digital information is distributed in "objects" or "containers" providing enhanced security and privacy but possibly more difficult access and identifiability.

As time passes, more and more valuable and desirable information will be available in digital containers. However, unless tools are developed to solve the problem, there will be no efficient or satisfying means to sort through the potentially trillions of digital containers available on tens of millions of Web pages, to find containers satisfying a search or fulfilling an information need. Furthermore, existing information searching mechanisms typically provide no way to readily perform a search that matches against underlying commercial requirements of providers and users.

It Will Be Difficult to Find Rights Management Scenarios Matching Your Requirements

If, for example, you have an auto repair newsletter and you want to create an article containing information on auto repair of Ford Bronco vehicles, you may wish to look for detailed, three dimensional, step-by-step "blow-up" mechanical images of Ford Bronco internal components. Perhaps these are available from hundreds of sources (including from private individuals using new, sophisticated rendering graphics programs, as well as from engineering graphics firms). Given the nature of your newsletter, you have decided that your use of such images should cost you no more than one penny to redistribute per copy in quantities of several thousand -- this low cost being particularly important since you will have numerous other costs per issue for acquiring rights to other useful digital information products which you reuse and, for example, enhance in preparing a particular issue. You therefore wish to search and match against rights management rules associated with such products -- non-limiting examples of which include:

- cost ceilings,
- redistribution rights (e.g., limits on the quantity that may be redistributed),
- modification rights,
- class related usage rights,
- category related usage rights,

- sovereignty based licensing and taxation fees,
- import and export regulations, and
- reporting and/or privacy rights (you don't want to report back to the product provider the actual identity of your end users and/or customers.

If you can't match against your commercial requirements, you may be forced to waste enormous amounts of time sifting through all of the available products matching Ford Bronco internal components -- or you may settle for a product that is far less than the best available (settling on the first adequate product that you review).

Computers Don't Necessarily Make It Easier to Find Things

Anyone who has ever used the Internet or the World Wide Web knows that networks, computers and electronics, when used together, do not necessarily make the overall task of finding information easier. In fact, computers can make the process seem much worse. Most Internet users will probably agree that trying to find things you are interested on the Internet can be a huge time drain. And the results can be very unsatisfactory. The rapid growth rate of information available on the Web is continually making this process of finding desired information even harder. You can spend many hours looking for information on a subject that interests you. In most cases, you will eventually find some information of value -- but even using today's advanced computer search tools and on-line directories, it can

take hours or days. With the advent of the technology advances developed by InterTrust Technologies Corp. and others, publishers will find it far more appealing to make their valuable digital information assets available on-line and to allow extractions and modifications of copyrighted materials that will vastly expand the total number of information objects. This will enormously worsen the problem, as the availability of valuable information products greatly expands.

It Is Usually Hard to Find Things On the Internet

There are many reasons why it is difficult to find what you want on the Internet. One key reason is that, unlike a public library, for example, there is no universal system to classify or organize electronic information to provide information for matching with what's important to the person who is searching. Unlike a library, it is difficult on the Internet to efficiently browse over many items since the number of possible choices may be much larger than the number of books on a library shelves and since electronic classification systems typically do not provide much in the way of physical cues. For example, when browsing library shelves, the size of a book, the number of pictures in the book, or pictures on magazine covers may also help you find what you are interested in. Such physical cue information may be key to identifying desired selections from library resources. Unfortunately, most digital experiences typically do not provide such cues without actually loading and viewing the work in digital form.

Thus, another reason why the electronic or digital world can make it even harder to find information than ever before has to do with the physical format of the information. The digital information may provide few or no outward cues or other physical characteristics that could help you to even find out what it is – let alone determine whether or not you are interested in it, unless such cues are provided through special purpose informational (for example, graphical) displays. On the Internet, everyone can be an electronic publisher, and everyone can organize their offerings differently -- using visual cues of their own distinctive design (e.g., location on a web page, organization by their own system for guiding choices). As one example, one publisher might use a special purpose graphical representation such as the video kiosk to support an electronic video store. Other publishers may use different graphical representations altogether.

Historically, there has been no particular need for consistent selection standards in conventional, non-electronic store based businesses. Indeed, it is often the unique display and choice selection support for customers' decision processes that make the difference between a successful store and a failure. But in the electronic world--where your choice is not among a few stores but rather is a choice among potentially thousands or even millions of possibly useful web sites and truly vast numbers of digital containers -- the lack of a consistent system for describing commercially significant variables that in the "real" world may normally be provided by the

display context and/or customized information guidance resource (catalog book, location of goods by size, etc.) seriously undermines the ability of digital information consumers to identify their most desirable choices.

Adding to this absence of conventional cues, the enormity of available choices made available in cyberspace means that the digital information revolution, in order to be practical, must provide profoundly more powerful tools to filter potentially desirable opportunities from the over abundance of choices. In sum, the absence of the ability to efficiently filter from a dimensionally growing array of choices, can completely undermine the value of having such a great array of choices.

In the "real" world, commercial choices are based on going to the right "store" and using the overall arrays of available information to identify one's selection. However, as information in digital and electronic form becomes more and more important, the problem of relating to the vast stores of information will become a nightmare. For example, picture yourself in a store where each shopping aisle is miles long, and each item on the shelf is packaged in the same size and color container. In an actual store, the product manufacturers put their products into brightly colored and distinctively shaped packages to make sure the consumer can readily find and select their product. These visual cues distinguish, for example, between a house brand

and a specific name brand, between low fat and regular foods, and between family size and small size containers.

On the Internet, a digital "store" is likely to be many stores with vast resources integrating products from many parties. If you were limited to conventional classification and matching mechanisms, you would be unable to sift through all the material to identify the commercially acceptable, i.e., an item representing the right information, at the right price, providing license rights that match your interests. Certainly, if each digital package looks the same, you are at a loss in making reasonable decisions. You can't tell one from another just by looking at it.

While information written on the "outside" of a digital package may be useful, you simply don't have the time to read all the packages, and anyway, each packager may use different words to describe the same thing and the descriptions may be difficult to understand. Some people may write a lot of information on the outside of their package, and others may write little or nothing on the outside of the package. If there is no universal system agreed upon by everyone for defining what information should be written on the outside of the package and how it should be formatted, using such a store would be painfully difficult even if you could limit the number of choices you were evaluating.

There is a Need For Efficient and Effective Selection Based, at Least in Part, on Rights Management Information

Unlike a real store where all breakfast cereals are shelved together and all soft drinks are in the same aisle, there may be no single, universal way to display the organization of all of the information in a "digital store" since, by its nature, digital information frequently has many implications and associated rules. For example, there now exist highly developed rights management systems such as described in U.S. Patent application Serial No. 08/388,107 of Ginter et al., filed 13 February 1995, for "Systems And Methods For Secure Transaction Management And Electronic Rights Protection (hereafter "Ginter et al") – the entire disclosure (including the drawings) of which is expressly incorporated into this application as if expressly set forth herein. Many rules associated with any given piece of digital information may, combinatorially, given rise to many, very different, commercial contexts that will influence the use decisions of different potential users in many different ways (e.g., cost, auditing, re-use, redistribution, regulatory requirements, etc.).

No readily available systems developed for the digital information arena provide similarly satisfying means that describe the many commercial rules and parameters found in individual custom catalogs, merchandise displays, product specifications, and license agreements. Further, no readily available mechanisms allow

"surfing" across vast choice opportunities where electronic matching can single out those few preferred items.

As one example, picking an appropriate image may involve any or all of the following:

- price,
- republishing (redistribution) rights,
- rights to extract portions,
- certified usable in certain sovereignties (e.g., pornographic content not allowed in Saudi Arabia),
- size,
- format, etc.,
- use and reuse administrative requirements (e.g., which clearinghouses are acceptable to rightsholders, what is the requirement for reporting usage information – is the name of your customer required, or only the use class(es) or none -- is advertising embedded), and
- other features.

No previously readily available technology allows one to efficiently make selections based on such criteria.

By their nature, and using the present inventions in combination with, amongst other things, "Ginter et al", the packages in a digital store may be "virtual" in nature -- that is, they may be all

mixed up to create many, differing products that can be displayed to a prospective customer organized in many different ways. This display may be a "narrowcasting" to a customer based upon his matching priorities, available digital information resources (e.g., repository, property, etc.) and associated, available classification information. In the absence of an effective classification and matching system designed to handle such information, digital information of a particular kind might be just about anywhere in the store, and very difficult to find since the organization of the stores digital information resources have not been "dynamically" shaped to the matching interests of the potential customer.

These Inventions Solve These Problems

The present inventions can help to solve these problems. It can give you or help you to find the things you like, need or want. For example, it can deliver to you, (including narrowcasting to you), or help you to find:

- things that match your interests;
- things that match your lifestyle;
- things that match your habits;
- things that match your personality;
- things you can afford and/or accept your preferred payment method;
- things that help you in your work;
- things that help you in your play;

- things that help you to help others;
- things that other people who are similar to you have found helpful,
- things that fulfill the commercial objective or requirements of your business activities; and
- things that will make you happy and fulfilled.

The present inventions can expand your horizons by helping you to find interesting or important things, things that you enjoy, things that optimize your business efficiency, and things that help you make the best digital products or services you can -- even if you didn't know precisely what or how to look for what you may need. It can also help you by allowing things you didn't know existed or know enough to look for – but that you may be interested in, want or need – to find you.

The Present Inventions Can Use "Metaclasses" to Take Multiple Classifications Into Account

In some areas, multiple classifications may already exist and thus it is important for a consumer to be able to find what he or she is looking for while taking into account not only that there may be multiple classifications, but also that some classifications may be more authoritative than others. For example, Consumer Reports may be more authoritative on certain topics than more casual reviews published, for example, in the local weekly newspapers.

As another example, consider a book that rates restaurants according several factors, including, for example, quality, price, type of food, atmosphere, and location. In some locations there may be many guides, but they may review different sets of restaurants. One guide may rate a particular restaurant highly while one or more others may consider it average or even poor. Guides or other sources of ratings, opinions, evaluations, recommendations, and/or value may not be equally authoritative, accurate, and/or useful in differing circumstances. One consumer may consider a guide written by a particular renowned expert to be more authoritative, accurate, and/or useful than a guide reflecting consumer polls or ballots. However, another consumer may prefer the latter because the second consumer may perceive the tastes of those contributing opinions to be closer to his or her own tastes than those of the experts.

In accordance with the present inventions, a person may be able to find a restaurant that meets specified criteria – for example, the highest quality, moderately priced Cantonese and/or Hunan Chinese food located in Boston or Atlanta – while weighting the results of the search in favor of reviews from travel books rather than from the local newspapers. As this example indicates, the searching may be according to class of authoritative source (and/or classes sources considered authoritative by the consumer) instead of weighting individual reviewers or sources. Thus in accordance with the present inventions, search may be performed at least in part based on classes of classes, or "metaclasses."

The Present Inventions Can Make Choices Easier

One simple way to look at some examples of the present inventions is as a highly sensitive electronic "matchmaker" that matches people or organizations with their best choices, or even selects choices automatically. The present inventions can match people and/or organizations with things and/or services, things with other things and/or services, and/or even people with other people. For example, the matching can be based on profiles that are a composite of preference profiles of one or more specific users, one or more user groups, and/or organizations -- where the contribution of any given specific profile to the composite profile may be weighted according to the specific match circumstances such as the type and/or purpose of a given match activity.

Figure 5 shows a simplified example of an electronic matchmaker that can match up two people with like interests. Sarah loves hiking, country and western music, gardening, movies and jogging. Mark loves movies, hiking, fast cars, country and western music, and baseball. The electronic matchmaker can look at the interests, personalities and/or other characteristics of these two people and determine that they are compatible and should be together -- while maintaining, if desired, the confidentiality of personal information. That is, unlike conventional matchmaking services, the present inventions can keep personal information hidden from the service provider and all other parties and perform matching within a

protected processing environment through the use of encryption and protected processing environment-based matching analysis.

For example, certain matching of facts that are maintained for authenticity may be first performed to narrow the search universe. Then, certain other matching of facts that are maintained for secrecy can be performed. For example, matching might be based on shared concerns such as where two parties who have a given disability (such as cancer or HIV infection) that is certified by an authority such as a physician who is certified to perform such certification; or the same income level and/or bank account (as certified by an employer and/or financial authority such as a bank). Some or all of such secret information may or may not be released to matched parties, as they may have authorized and/or as may have been required by law when a match is achieved (which itself may be automatically managed within a protected processing environment through the use of controls contributed by a governmental authority).

Figure 5A shows an electronic matchmaker that matches an electronic publisher with mystery stories for his quarterly electronic mystery anthology, where the matching is based on price, redistribution rights, editing rights, attribution requirements (attributing authorship to the author), third party rating of the writers quality, length of story, and/or the topical focus of the story (for example). Here, rule managed business requirements of publisher and writers are matched allowing for great efficiency in matching,

coordination of interests, and automation of electronic business processes and value chain activities.

The convenience of the "electronic matchmaker" provided in accordance with the present inventions extends to commerce in physical goods as well -- as illustrated in Figure 5b. In this non-limiting example, the electronic matchmaker is communicating to the consumer via the Internet and World Wide Web. The matchmaker has found the lowest quoted price for a Jeep sports utility model given, in this one example, a multitude of factors including:

- model,
- color,
- options package,
- availability, and
- discounts resulting from the consumer's membership in certain classes (such as membership in the American Association of Retired Persons, membership in the American Automobile Association, and being a graduate of Stanford University).

Membership in these associations and alumni status may be conveyed or indicated by possession of a special electronic document called a "digital certificate," "membership card," and/or other digital credential that warrants or attests to some fact or facts.

Thus, the electronic matchmaker provided in accordance with these inventions can also match people with things. Figure 6 shows two people, Harry and Tim. Harry loves sports most of all, but also wants to know a little about what is going on in the business world. The business world is most important to Tim, but he likes to keep up with the baseball scores. The electronic matchmaker in accordance with these inventions can learn about what Harry and Tim each like, and can provide information to a publisher so the publisher can narrowcast a newspaper or other publication customized for each of them. A newspaper company can narrowcast to Harry lots of sports information in his newspaper, and it can narrowcast to Tim mostly business information in his newspaper. In another example, Harry's newspaper may be uniquely created for him, differing from all other customized newspapers that emphasize sports over business information. But information that Harry and Tim respectively want to maintain as authentic or secret can be managed as such.

The electronic matchmaker can also match things with other things. Figure 7 shows how the electronic matchmaker can help a student put together a school project about big cats. The electronic matchmaker can help the student locate and select articles and other material about various kinds of big cats. The electronic matchmaker can, for example, determine that different articles about tigers, lions and cheetahs are all about big cats – but that articles about elephants and giraffes are not about big cats. If there is a charge for certain items, the electronic matchmaker can find only those items that the

student can afford, and can make sure the student has the right to print pictures of the big cats. The electronic matchmaker can help the student to collect this information together so the student can make a colorful poster about big cats.

The electronic matchmaker can match up all sorts of different kinds of things. Figure 8 shows the electronic matchmaker looking at three different objects. The matchmaker can determine that even though objects A and C are not identical, they are sufficiently similar that they should be grouped together for a certain purpose. The electronic matchmaker can determine that for this purpose, object B is too different and should not be grouped with objects A and C. For a different purpose, the electronic matchmaker may determine that objects A, B and C ought to be grouped together.

The Present Inventions Can Make Use of Rights Management Information

How does the electronic matchmaker find out the information it needs to match or classify people and things? In accordance with a feature provided by these inventions, the electronic matchmaker gets information about people and things by using automated, computerized processes. Those processes can use a special kind of information sometimes known as rights management information. Rights management information may include electronic rules and/or their consequences. The electronic matchmaker can also use information other than rights management information.

An example of rights management information includes certain records about what a computer does and how it does it. In one simple example, records may give permission to read a particular news article if that the customer is willing to pay a nickel to purchase the article and that the nickel may be paid using a budget provided by a credit card company or with electronic cash. A customer might, for example, seek only news articles from providers that take electronic cash and/or process information with a certain information clearinghouse as described in U.S. Patent application Serial No. 08/699,712 to Shear et al., filed 12 August 1996, for "Trusted Infrastructure Support Systems, Methods And Techniques For Secure Electronic Commerce Electronic Transactions And Rights Management" (hereafter "Shear et al") – the entire disclosure (including the drawings) of which is expressly incorporated into this application as if expressly set forth herein.

The Present Inventions Can Maintain Privacy

Figure 9 shows one way in which the electronic matchmaker can get information about a person. In this example, the electronic matchmaker asks Jill to fill out a computer questionnaire about what she likes. The questionnaire can also ask Jill what information she wishes to be maintained as authentic, and what information (e.g., encrypted by the system) may be used for secure matching only within a protected processing environment and can not be released to another party, or only to certain specified parties. The questionnaire

answering process may be directly managed by a protected processing environment to ensure integrity and secrecy, as appropriate.

For example, the questionnaire may ask Jill whether she likes baseball and whether she is interested in volcanoes. The electronic matchmaker can also ask Jill if it is okay to look at records her computer maintains about what she has used her computer for in the past. These computer records (which the computer can maintain securely so that no one can get to them without Jill's permission) can keep a history of everything Jill has looked at using her computer over the past month and/or other time period – this process being managed, for example, through the use of a system such as described in the "Ginter et al."

Looking at Figure 10, Jill may have used her computer last week to look at information about baseball, volcanoes and Jeeps. With Jill's permission, the electronic matchmaker can employ a protected processing environment 154 (schematically shown here as a tamper-resistant "chip" within the computer – but it can be hardware-based, software-based, or a combination of hardware and software) to look at the computer's history records and use them to help match Jill up with other kinds of things she is or may be interested in. For example, the electronic matchmaker can let an electronic publisher or other provider or information gatherer (e.g., market survey conductor, etc.) know that Jill is interested in team sports, geology and sports utility vehicles with or without more revealing detail -- as managed

by Jill's choices and/or rights management rules and controls executing in her computer's protected processing environment 154. The provider can send information to Jill – either automatically or at Jill's request – about other, related things that Jill may be interested in.

Figure 11 shows an example of how rights management and other information Jill's computer maintains about her past usage can be useful in matching Jill up with things she may need or want. The computer history records can, for example, show that Jill looked at hockey information for three hours and football information for five hours during the past week. They can indicate that Jill uses a Discover credit card to pay for things, usually spends less that $10 per item, averages $40 per month in such expenses, and almost never buys new programs for her computer.

The electronic matchmaker can, with and subject to Jill's permission, look at and analyze this information. As one example, the electronic matchmaker can analyze relevant rules and controls provided by third parties who have rights in such information -- where such rules are controlled, for example, by Jill's computer's protected processing environment 154. It can also look at and analyze Jill's response to computer questionnaires indicating that she likes baseball and football. The electronic matchmaker can, based on all of this information, automatically select and obtain videos and/or other publications for Jill about team sports and that cost less than

$10 and that accept payment using a Discover card, so that Jill can preview and select those in which she may have a particular interest and desire to acquire.

Figure 12 shows that the electronic matchmaker can take into account computer history records for lots of different people. The electronic matchmaker can work with other rights management related computer systems such as "usage clearinghouses" (non-limiting examples of which are described in each of "Ginter et al" and "Shear et al") to efficiently collect rights management related information. The ability to collect history records from many different people can be very useful. For example, this can allow the electronic matchmaker to distinguish between things that are very popular and things that are not so popular.

The present inventions provide great increases in efficiency and convenience. It can save you a lot of time and effort. It can allow computers to do a lot of the work so you don't have to. It can allow you to compete with larger businesses -- and allow large business to function more efficiently -- by allowing the location of resources particularly appropriate for certain business activities. You can delegate certain complex tasks to a computer, freeing you to be more productive and satisfied with electronic activities. These automated processes can be "smart" without being intrusive. For example, they can learn about your behavior, preferences, changing interests, and even your personality, and can then predict your future interests based

on your past behavior and interest expressions. These processes can ensure confidentiality and privacy – so that no one can find out detailed information about you without your consent. Across the full range of personal and business activities, the present inventions allow a degree of basic efficiency, including automation and optimization of previously very time consuming activities, so that interests and possible resources are truly best matched.

The present inventions handle many kinds of important issues and addresses the widest range of information and rights and automation possibilities. For example, the present inventions are capable of handling (but are not limited to):

- consumer information;
- computer information;
- business information;
- entertainment information;
- other content information;
- information about physical products;
- all other kinds of information.

It can reflect and employ all kinds of rights to optimize matching processes, including:

- content rights;
- privacy rights;
- governmental and societal rights;
- provider rights;

- distributor rights;
- consumer rights;
- workflow rights;
- other value chain participant rights;
- work flow rights;
- business and personal rights and processes of all kinds.

It can employ all kinds of parameter information, including:

- budget,
- pricing
- redistribution
- location (of party, item, etc.)
- privacy
- identity authenticity and/or specificity
- any other parameter information.

Pricing (for example the price of a specific item) can be used in matching based upon price per unit and/ or total price for a volume purchase, price for renting, right to redistribute, cost for redistributing items, etc.

Privacy can be used for establishing matching contingent upon usage reporting requirements for viewing, printing, extracting, dedistributing, listening, payment, and/or requiring the reporting of

other information such as personal demographics such as credit worthiness, stored value information, age, sex, marital status, race, religion, and/or usage based generated profiling information based materially upon, for example, a users history of usage of electronic content and/or commercial transactions, etc.

Identity can be used for matching based upon, for example, such as the presence of one or more specific, class, and/or classes of certificates, including, for example, specific participant and/or group of participant, including value chain certificates as described in "Shear et al".

With the inventions described herein, commercial requirement attributes embodied in rules (controls and control parameter data) are employed in classification structures that are referenced by search mechanisms, either, for example, directly through reading rule information maintained in readable (not encrypted) but authentic (protected for integrity) form, through reading rule information maintained securely, through processes employing a protected processing environment 154 of a VDE node, and/or through the creation of one or more indexes and/or like purpose structures, that, directly, and/or through processes employing a protected processing environment 154, automatically compile commercial and other relevant (e.g., societal regulatory information such as a given jurisdiction's copyright, content access and/or taxation regulations) for classification/matching purposes.

The present inventions can employ computer and communication capabilities to identify information, including:

- topical classification such as described by conventional library classification systems,
- commercial characterizations -- including commercial parameter data such as pricing, size, quality, specific redistribution rights, etc.,
- creator (e.g., a publisher or manufacturer), distributor, societal, user, and other participant interests information,
- information generated by automated profiling of any and all of such parties or collections of parties,
- matching (including electronically negotiating a match) between the interests of any of such parties,
- where appropriate, the use of statistical procedures, expert systems, and artificial intelligence tools for profiling creation and/or analysis, matching, and/or negotiation.

The present inventions thus provide for optimal user, provider, and societal use of electronic cyberspace resources (for example, digital information objects available across the Internet, sent by direct broadcast satellite, transmitted over a cable TV system, and/or distributed on optical disk).

Of particular importance is the notion of classes of content, classes of users, and classes of providers. For example, the present inventions can make use of any/all of the following:

- topical identification, for example, such as information represented in typical library subject and/or author and/or catalog and/or keyword search and retrieval information systems;
- any commercial requirements, associated with the use of electronic information (and/or to products, including non-electronic products, and/or to any service), including information embodied in encrypted rules (controls and/or parameter data) governing rights in electronic value chain and electronic interaction contexts, and further including information guaranteed for integrity;
- any information descriptive of an available resource (which may include any information, product, and/or service, whether available in electronic and/or physical forms) such as: the quality of a digital product as evaluated and ranked and/or otherwise specified by one or more third parties and/or independent third parties (e.g., Consumer Reports, a trusted friend, and/or a professional advisor), the size of a product, length in time in business of a service or in the market of a product, a product's or service's

market share, and/or subject governmentally and/or other societally imposed rules and/or integrity guaranteed descriptions, including any associated regulatory requirements, such as societal requirements granting and/or reporting access to information, for example, information on how to create a nuclear bomb to a confidential government auditing agency (this allowing free access to information while protecting societal rights);

- any information descriptive of a user and/or department and/or organization and/or class of users and/or departments and/or organizations (including, for example, such descriptive information encrypted and/or guaranteed for integrity) wherein such information may include, for example, name, physical and/or network and/or cyber-wide logical network location, organizational and/or departmental memberships, demographic information, credit and/or trustworthiness information, and profile preference and usage history information, including any generated profile information reflecting underlying preferences, and/or classes based on said descriptive information and/or profiles.

Some Of The Advantageous Features And Characteristics Provided By The Present Inventions

The classification, matching, narrowcasting, analysis, profiling, negotiation, and selection capabilities of the present inventions include the following capabilities (listed items are not mutually exclusive of each other but exemplary samples):

- Enables highly efficient provision of classes of information, entertainment, and/or services to classes of individuals and/or entities that have (and/or may obtain) the right(s) to such information and are likely to find identified information interesting, useful, and/or entertaining.
- The present inventions also provide systems and methods for efficiently determining class hierarchies, classification schemes, categories, and/or category schemes and/or the assignment of objects, persons and/or things to said class hierarchies, classification schemes, categories, and/or category schemes using at least some rights management information.
- Helps systems, groups, and/or individuals classify, locate, and/or obtain specific information and/or classes of information made available through so-called "publish and subscribe" systems and methods using, among other things, subject-based addressing and/or messaging-based protocol layers.

- Provides fundamentally important commercial and societal rules based filtering to identify desired electronic information and/or electronic information containers through the use of classification structures, profiling technology, and matching mechanisms that harness the vast information opportunities in cyberspace by matching the information needs of users against commercial and/or societal rules related to the use of available information resources, including, for example, commercial and/or societal consequences of digital information use imposed as provider requirements and specified through the use of, and enforced by the use of, a trusted rights management system such as described in "Ginter et al".
- Enables content creators and/or distributors to efficiently "stock the shelves" of retail electronic content outlets and similar merchanisers (both electronic and hard goods) with products and/or services most likely to be purchased and/or used by the customers of such merchanisers. This includes both identifying and "stocking" the most desirable products and/or other user desired resources and optimally presenting such products and/or other

resources in a manner optimized for specific users and/or user classes.

- Matching may be based on history of matching, that is, matching derived at least in part from previous matching, one non-exhaustive example of which includes learned matching for increasing efficiency.
- Enables matching for value chains where the matching is against a plurality of co-participating value chain parties requirements and/or profiles against match opportunities, and/or matching by matches comprised of match input and/or aggregation of match rule sets of providers used to "dock" with one or more user needs, interests, requirements match sets.
- Helps match persons and/or things using fuzzy matching, artificial intelligence (e.g., expert systems), and other methods that that match using plural match sets from providers and/or receivers.
- Makes search easier by using smart agents that match at least in part using at least one class.
- Helps bring buyers and sellers together through cross matching, where both parties offer to provide and/or receive content and/or physical goods for consideration, including barter matching and negotiated barter and other kinds of matching.

- Helps potential customers find those members (e.g., objects such as digital information containers) of any one or more classes of content most useful, entertaining, and/or interesting to them.
- Facilitates organizations securely and efficiently acquiring and distributing for internal use certain classes of content available from external providers and/or more securely and/or efficiently managing classes of their own content, including being able to authorize certain classes of employees to use specified classes of internal and/or external content.
- Efficiently supporting matching between users and digital information where participants in a chain of handling and control have specified rules and usage consequences for such digital information that may depend on class membership, for example, on class(es) of content and/or class(es) of value chain participants and/or classes of electronic events, wherein such participants include, for example, users and/or participants contributing rules and consequences.
- Enables first individuals and/or organzations to locate efficiently other individuals, organizations, products, and/or services who have certain characteristics that corresponds to such first individuals' and/or

organizations' interests, including interests generated by profiling information locally gathered through local event auditing at a VDE installation.

- Facilitates businesses informing a customer about things of special interest to her or him, such as classes of goods, services, and/or content, including directing such information to a customer at least in part based on profiling information locally gathered at a VDE installation through local event auditing at a VDE installation.
- Allows trading companies to match suppliers of certain classes of goods and/or services with those who desire to purchase and/or use those classes of goods and/or services, wherein such matches may include fulling a commercial business interaction and may further include one or more sequences of matches and/or nested matches (a sequence and/or grouping of matches within a given organization or group, wherein such matches may be required to occur in a certain order and/or participate along with other matches in a group of matches before a given match is fulfilled).
- Enhances equity portfolio management by making easier for traders to identify those equities having certain desired characteristics, such as belonging to

the class of equities that will have the greatest positive effect on the value of the trader's portfolio given certain classes of information and assumptions. Such matches may take into account information external to the fulfilment of a given trade, for example, one or more certain other market or specific variable thresholds must be met before an equity is traded, such as a certain rise in the an index stock value of, and/or revenue of, certain one or more network hardware suppliers before a certain quantity of equity is purchased at a certain price for stock of a certain network hardware supplier raw network component manufacturer, and wherein, for example, such determinations can be performed highly efficiently at a user VDE installation as the point of control, where such node receives such trusted information in, for example, VDE containers, as is necessary for a control decision to occur to purchase such equity of such network hardware supplier raw component manufacturer.

- Makes easier automated foreign currency exchange by enabling currency traders to identify members of the class of possible trades and/or conversions that are likely to produce the best returns and/or minimize losses.

- Helps consumers and organizations manage their affairs more efficiently and effectively and helps providers of services by automatically matching users with services that meet certain specified criteria, such as, for example, U. S. and Swiss banks offering the highest interest rates on certain time based classes of bank deposit instruments.
- Enables distributers of software and other content to identify one or more classes of users who are most likely to be interested in purchasing or otherwise using certain classes of software.
- Enables rightsholders to employ rules and/or usage consequences dependent on membership in one or more classes where class membership may be indicated by posession of a special digital document called a "certificate."
- Enables rightsholders to employ rules and/or usage consequences at least partially dependent on roles and responsibilities within an organization, where those roles and responsibilities may be indicated by posession of a digital certificate, digital membeship card, and/or other digital credential.
- Facilitates more efficient automation of manufacturing and other workflow processes by, for example, matching certain manufacturing steps and/or

processes with performance parameter data associated with available classes of equipment capable of performing those steps and/or processes.

- Makes easier the administration and enforcement of government and/or societal rights by, for example, providing matching means for automatically applying certain classes of tax rules to appropriate classes of sales and other transactions.
- Enables altering the presentation of information and/or other content depending on the matching between preferences of the user and one or more classes of content being presented.
- Enables processing or altering (narrowcasting) of an event (e.g., the presentation of information and/or other content), for example, dynamically adjusting the content of an event, in response to a matching among the preferences and/or reactions of a user and/or user group, one or more classes of content being processed through one or more events, one or more classes of one or more users participating in and/or otherwise employing the one or more events, and/or event controls (i.e., rules and/or parameter data).
- Allows the rules and usage consequences and the presentation of information to vary according to the difficulty of the information, including, for example,

adjusting the difficulty of an electronic game so that it is neither too frustratingly difficult nor too easy to use.

- Enables a user to efficiently locate content in one or more particular classes, where class is defined at least in part by weighted topical classification, where, for example, a document or other object is classified in one or more categories where at least one category reflects the absolute or relative attention given to that class in the object being classified.
- Facilitates users' creation of a new document from parts of two or more documents, where at least one of such parts is identified and/or retrieved based upon matching the part's membership in one or more classes identified by trusted, commercial controls employed through the use of a rights management system.
- Enables users to search for, locate, and use only those parts of a document that belong to one or more specified classes, including those parts having certain commercial controls, for example, reflecting acceptable usage restrictions and/or pricing.
- Enhances search and retrieval by creating new classes of content discriptors that incorporate various

dispirate standards for content description and/or location.

- Allows consumers to easily locate services having certain specified characteristics, for example, gambling services offering the most favorable odds and/or specified rules for a particular game or games.
- Helps consumers obtain certain classes of tickets to certain classes of events.

The above capabilities, and others described in this application, are often ideally managed by distributed commerce nodes of a distributed, rights management environment embedded in or otherwise connected to the operating system clients of a distributed computing environment such as described in "Ginter et al" and further described in "Shear et al", and employing, for example, rules, integrity management, container, negotiation, clearinghouse services, and trusted processing capabilities described in "Ginter et al" and "Shear et al".

The Present Inventions Make Use Of Many Kinds Of Information And/Or Data

As discussed above, these inventions provide, among other things, matching, classification, narrowcasting, and/or selection based on rights management and other information. In particular preferred examples, these matching, classification, narrowcasting, and/or

selection processes and/or techniques may be based at least in part on rights management information. The rights management information may be an input to the process, it may be an output from the process, and/or the process can be controlled at least in part by rights management information. Information in addition to, or other than, rights management information may also be an input, an output, and/or a basis for controlling, the process and/or techniques.

Rights management information may be directly or indirectly inputted to the matching, classification and/or selection process. For example, rights management controls, rules and/or their consequences may be an input. Examples of such controls and/or rules include object registration related control set data, user related control set data and/or computer related control set data. In addition or alternatively, information provided based on control sets or rules and their consequences may be inputted. The following are examples of such information that may be provided based, for example, on rules and consequences:

- information exhaust;
- user questionnaires,
- audit trail related information;
- aggregated usage data;
- information measuring or otherwise related to user behavior;
- information measuring or otherwise related to user preferences;

- information measuring or otherwise related to user personality;
- information measuring or otherwise related to group behavior;
- information measuring or otherwise related to group preferences;
- information measuring or otherwise related to group culture
- information measuring or otherwise related to organizational behavior;
- information measuring or otherwise related to organizational preferences;
- information measuring or otherwise related to organizational culture;
- information measuring or otherwise related to institutional behavior;
- information measuring or otherwise related to institutional preferences;
- information measuring or otherwise related to institutional culture;
- information measuring or otherwise related to governmental behavior;
- information measuring or otherwise related to governmental preferences;

- information measuring or otherwise related to governmental culture;
- information measuring or otherwise related to societal behavior;
- information measuring or otherwise related to societal preferences;
- information measuring or otherwise related to societal culture;
- object history related information;
- other types of information;
- any combinations of information including, some, all or none of the information set forth above.

The processes, techniques and/or systems provided in accordance with these inventions may output rights management related information such as, for example:

- one or more control sets;
- various rules and/or consequences;
- information used by control sets;
- certificates;
- other rights management information.

In accordance with various preferred embodiments provided by these inventions, information other than rights management information may also be used, at least in part, as an input, output and/or to control the matching, classification, narrowcasting, and/or

selection processes, systems and/or techniques. Examples of such information include:

- content object information;
  - full text
  - portions of objects
  - portions of sub-objects
  - abstracts
  - metadata
  - other content object related information
- user information
  - census information
  - purchasing habits
  - credit and financial transaction related information
  - governmental records
  - responses to questionnaires
  - survey results
  - other user information
- computer related information
  - identification information
  - configuration information
  - other computer related information
- combinations of information.

Matching/Classifying/Selection

Systems, methods and techniques provided in accordance with these inventions can classify a variety of types of things including, for example:

- people
- computers
- content
- events
- transactions
- objects of all types
- combinations of things;
- combinations of people and things.

The matching, classifying and/or selecting processes provided in accordance with these inventions are very flexible and useful. For example, they may be used to associate people with information, information with other information, people with other people, appliances with people, appliances with information, and appliances with other appliances. The present inventions in their preferred examples can associate any kind of information, object or thing with any other kind of information, object or thing.

Different Associations Between Classes and Rights

The processes, systems and/or techniques provided in accordance with these inventions can provide and/or take into account many different kinds of associations between classes and rights. For

example, they can look at what rights are available to a user, computer, data structure or any other object. They can also look to rights selected by an object (for example, the subset of rights a user has chosen or otherwise identified). Alternatively or in addition, they can look to rights that have been exercised by a user or in conjunction with an object or other thing, and they can look to the consequences of exercising such a right(s).

Embodiments in Accordance With the Present Inventions Can Be Used to Define Classes Based on Uni-Dimensional and/or Multi-Dimensional Attributes and/or Characteristics

Example processes, systems and/or techniques provided in accordance with these inventions can be used to define classes based on uni-dimensional and/or multi-dimensional attributes and/or characteristics. Any one or more attributes can be used. The attributes and/or characteristics can be flexibly defined. They may define groups or classes containing elements sharing certain attributes in common. There can, for example, be a spectrum of classification that takes into account gray areas as to whether a particular person or thing possesses a certain one or a number of particular attributes and/or characteristics. Or classification may have a higher degree of certainty or definition. For example, a process can test to determine whether particular people or things are inside or outside of particular classes or groups based on one or a number of attributes or characteristics (for example, whether you live in Denver, are under the age of 25 and are single). In accordance with additional specific

features provided by these inventions, there may be a minimum number of different classes set up to "cover" a particular situation – with every person or thing either being within or outside of a given, disjoint class or group.

Preferred Examples In Accordance With The Present Inventions Are Extensible to Accommodate Changing Conditions

The systems, methods and/or techniques provided by these inventions are extensible to accommodate changing conditions. For example, they can be made to readily adapt to changes in rules, consequences, topics, areas and/or subjects pertaining to groups such as, for example categories, and any other variable. Furthermore, partially and/or entirely new variables may be introduced to one or more existing sets of variables -- for example, to extend or otherwise modify a model to account for additional variables, to apply a new strategy, to adapt to new network and/or installation circumstances, to adapt to new user factors, to change analysis and/or other processing characteristics, and so on.

Preferred Examples In Accordance With The Present Inventions Are Compatible With Pre-Existing or Any New Classification Techniques or Arrangements

The example systems, methods and/or techniques provided by these inventions can be made fully compatible with any classification and/or categorization means, method, process, system, technique, algorithm, program, and/or procedure, presently known or unknown,

for determining class and/or category structures, definitions, and/or hierarchies, and/or the assignment of at least one object, person, thing, and/or member to at least one class and/or category, that without limitation may be:

- implemented by computer and/or other means; and/or
- based upon discrete and/or continous mathematics; and/or
- using nominal, ordinal, interval, ratio and/or any other measurement scale and/or measurement mode; and/or
- including parameter data; and/or
- entail linear and/or non-linear estimation methods; and/or
- any other methods.

For example, classification can be performed using any or all of the following example classification techniques:

- Statistical techniques that identify one or more clusters of cases sharing similar profiles and/or features, including any of the family of cluster analysis methods, for example, those described in Hartigan (Hartigan, J. A., Clustering Algorithms, New York: Wiley, 1975);
- Methods for numerical taxonomy, for example, as described, for example, by Sneath and Sokal (Sneath, Peter H.A. and Robert R. Sokal, Numerical

Taxonomy: The Principals and Practice of Numerical Classification, San Francisco: W.H. Freeman, 1973);

- Any of the methods for cluster analysis, factor analysis, components analysis, and other similar data reduction/classifiction methods, for example, those implemented in popular statistical and data analysis systems known to those skilled in the arts, for example, SAS and/or SPSS;
- Pattern classification techniques, including components analysis and neural approaches, for example, those described by, for example, Schurmann (Schurmann, Jurgen, Pattern Classification: A Unified View of Statistical and Neural Approaches, New York: John Wiley & Sons, 1966);
- Statistical techniques that identify one or more underlying dimensions of qualities, traits, features, characteristics, etc., and assign parameter data indicating the extent to which a given case has, possesses, and/or may be characterized by the underlying dimension, factor, class, etc. and/or result in the definition of at least one class and/or the assignment of at least one case to at least one class, for example, as described by Harman (Harman, Harry H., Modern Factor Analysis, 3rd ed. rev., Chicago: University of Chicago Press), and/or as implemented

by SAS and/or SPSS and/or other statistical analysis programs.

- Statistical methods that employ fuzzy logic and/or fuzzy measurement and/or whose assignment to at least one class entails probabilities different from 1 or zero.
- Baysian statistical classification techniques that use estimates of prior probabilities in determining class definitions and/or the assignment of at least one case to at least one class;
- Any statistical and/or graphical classification and/or data reduction method that uses rotation of reference axes, regardless of whether orthogonal or oblique rotations are used, for example, as described in Harman, and as implemented in SAS and/or SPSS and/or other statistical programs;
- Statistical methods for two and three way multidimensional scaling, for example, the methods described by Kruskal and Wish (Krusgal Joseph B. and Myron Wish, Multidimensional Scaling, Beverly Hills, CA: Sage Publications, 1978), and/or by Shepard, et al. (Shepard, Roger N., A. Kimball Romney, and Sara Beth Nerlove, Multidimensional Scaling: Theory and Applications in the Behavioral Sciences, New York: Seminar Press, 1972);

- Knowedge based approaches to classification, for example, as described by, for example, Stefik (Stefik, Mark, "Introduction to Knowledge Systems," San Francisco: Morgan Kauffman, 1995); and
- any other classification techniques or arrangements pre-existing or yet to be developed.

Preferred Examples In Accordance With The Present Inventions Are Fully Compatible With A Wide Array of Technologies Including the Distributed Commerce Utility System and the Virtual Distribution Environment

Systems, methods and/or techniques provided in accordance with these inventions build upon and can work with the arrangements disclosed in "Ginter et al"; "Shear et al"; and other technology related to transaction and/or rights management, security, privacy and/or electronic commerce.

For example, the present inventions can make particular use of the security, efficiency, privacy, and other features and advantages provided by the Virtual Distribution Environment described in "Ginter et al".

As another example, a matching and classification arrangement can be constructed as a distributed commerce utility system as described in "Shear et al". The present inventions can work with other distributed commerce utility systems, and can enhance or be a part of other commerce utility systems.

By way of non-exhaustive, more specific examples, the present inventions can be used in combination with (and/or make use of) any or all of the following broad array of electronic commerce technologies that enable secure, distributed, peer-to-peer electronic rights, event, and/or transaction management capabilities:

- a "VDE" ("virtual distribution environment") providing, for example, a family of technologies by which applications can be created, modified, and/or reused;
- a standardized control and container environment which facilitates interoperability of electronic appliances and efficient creation of electronic commerce applications and models;
- a programmable, secure electronic transaction management foundation having reusable and extensible executable components;
- seamless integration into host operating environments of electronic appliances or direct employment of such technologies in electronic commerce applications;
- cyberspace digital content rights and transaction management control systems that may operate in whole or in part over Internets, Intranets, optical media and/or over other digital communications media;
- support of an electronic "world" within which most forms of electronic transaction such as content usage,

distribution, auditing, reporting, and payment activities can be managed;

- Transaction Operating Systems (operating systems that have integrated secure, distributed, and programmable transaction and/or event management capabilities);
- Rights Operating Systems (operating systems that have integrated, distributed, and programmable rights management capabilities);
- secure content container management;
- clearinghouse functions related to content usage;
- overall electronic commerce architectures that provide electronic commerce automation through the use of secure, distributed digital events management;
- the general enablement of traditional commerce behavior in the digital commerce world;
- enhanced inherent, distributed efficiencies of conventional commerce practices with powerful, reliable electronic security, and with the programmability and electronic automation efficiencies made possible by modern computing;
- trusted operation of a freely configurable, highly efficient, general purpose digital marketplace in which

parties "come together" to establish commercial relationships;

- support of "real" commerce in an electronic form (that is, the progressive creation of commercial relationships that form, over time, a network of interrelated agreements representing a value chain business model);
- enabling content control information to develop through the interaction of (and/or negotiation between) securely created and independently submitted sets of content and/or appliance control information;
- interconnection of appliances providing a foundation for much greater electronic interaction and the evolution of electronic commerce;
- a variety of capabilities for implementing an electronic commerce environment;
- a neutral, general purpose platform for commerce;
- an architecture that avoids reflecting specific distribution biases, administrative and control perspectives, and content types;
- a broad-spectrum, fundamentally configurable and portable, electronic transaction control, distributing, usage, auditing, reporting, and payment operating environment;

- systems and methods that uniquely enable electronic commerce participants to protect their interests during the sequence of activities comprising an electronic commerce model;
- ability of commerce participants to assure protection by specifying rules and controls that monitor and enforce their interests during the processing of remote commerce events;
- permitting commerce participants to efficiently participate in, and manage, the distributed electronic activities of a digital value chain;
- allowing commerce model participants to, for example, securely and cooperatively govern and automate the distributed electronic activities comprising their collective electronic business models;
- allowing commerce model participants to securely contribute electronic rules and controls that represent their "electronic" interests;
- rules and controls that extend a "Virtual Presence™" through which the commerce participants govern remote value chain activities according to their respective, mutually agreed to rights;

- a Virtual Presence taking the form of participant specified electronic conditions (rules and controls) that must be satisfied before an electronic event may occur;
- rules and controls that enforce the party's rights during "downstream" electronic commerce activities;
- control information delivered by, and/or otherwise available for use with, the VDE content containers constituting one or more "proposed" electronic agreements which manage the use and/or consequences of the use of such content and which can enact the terms and conditions of agreements involving multiple parties and their various rights and obligations;
- rules and controls from multiple parties forming aggregate control sets ("Cooperative Virtual Presence™") that ensure that electronic commerce activities will be consistent with the agreements amongst value chain participants;
- control sets defining the conditions which govern interaction with protected digital content (disseminated digital content, appliance control information, etc.);
- conditions used to control not only digital information use itself, but also the consequences of such use to protect the individual interests of commerce participants

and form cooperative, efficient, and flexible electronic commerce business models;

- true, efficient electronic cooperative governance of value chain activities;
- empowering each commerce model participant to securely deliver, and persistently maintain control over, the rules and controls they contributed specifying constraints on, and consequences of, electronic conduct;
- extending Cooperative Virtual Presence over time and involving the execution of controls, and the use of content, at physically dispersed locations, such as Internet user sites;
- a chain of handling and control in which dispersed locations are bound together through the use of secure communication techniques and unique, secure digital container technology;
- ability to preserve the rights of parties through a series of transactions which may occur at different times and different locations;
- extending the ability of electronic content providers to control the use of proprietary information;
- allowing content providers to limit use to authorized activities and amounts;

- allowing participants (e.g., actors, directors, script and other writers, musicians, studios, publishers, distributors, retailers, advertisers, credit card services, content end-users, and others) involved in a business model to have the ability to embody their range of agreements and requirements, including use limitations, into an "extended" agreement comprising an overall electronic business model;
- representing such an extended agreement by electronic content control information which can automatically enforce agreed upon rights and obligations;
- a competitive, general purpose electronic commerce architecture supporting the distributed, secure "unmanned" electronic interaction;
- distributing such capabilities across networks and involving the sequence (or web) of distributed activities underlying electronic value chains;
- cooperative electronic governance of distributed electronic commerce processes that optimizes electronic commerce value propositions;
- the capability of electronically, remotely representing the interests of commerce participants to support efficient, flexible, commerce model automation;

- enabling rules and controls that are independently contributed by multiple parties to securely merge together and form the collective rules and controls sets that reflect the electronic commerce agreements between parties;
- using rules and controls sets to collectively, automatically, govern remote electronic conduct;
- securely managing the integration of control information provided by two or more parties;
- constructing electronic agreements between VDE participants that represents a "negotiation" between the control requirements of two or more parties and enacts the terms and conditions of a resulting agreement;
- ensuring and/or enforcing the rights of each party to an electronic agreement regarding a wide range of electronic activities related to electronic information and/or appliance usage;
- the ability to broadly support electronic commerce by securely managing independently delivered VDE component objects containing control information (normally in the form of method, data, or load module VDE objects);
- using independently delivered control information to negotiate with senior and other pre-existing content

control information to securely form derived control information;

- ensuring that all requirements specified by derived control information are satisfied before VDE controlled content is accessed or otherwise used;
- ensuring that all load modules and any mediating data which are listed by the derived control information as required are available and perform their required function;
- use of independently delivered control components to allow electronic commerce participants to freely stipulate their business requirements and trade offs;
- allowing electronic commerce, through the various control requirements stipulated by VDE participants, to evolve into forms of business which are the most efficient, competitive and useful -- much as with traditional, non-electronic commerce;
- providing commerce participants with the ability to freely fashion the chains of handling and control pathways that protect data and processes and the freedom to shape the models within which their Virtual Presence operates -- allowing commerce participants to optimally formulate their electronic commerce value propositions;

- VDEs configured to support the various underlying agreements between parties that define important electronic commerce pathways of handling for electronic content, content and/or appliance control information, content and/or appliance usage information and payment and/or credit;

- allowing content creators and other providers to specify the pathways that, partially or fully, must be used to disseminate commercially distributed property content, content control information, payment administrative content, and/or associated usage reporting information;

- empowering commerce participants, subject to the rules and controls previously set in a value chain, to freely fashion control models implementing their Virtual Presence by using GUI templates or rights programming languages employing commerce/rights management components;

- component based control methods that allow the present inventions to efficiently operate as a highly configurable content control system;

- content control models that can be iteratively and asynchronously shaped, modified, and otherwise updated to accommodate the needs of VDE participants;

- iterative and/or concurrent multiple participant processes through the submission and use of secure, control information components (e.g., executable code such as load modules and/or methods, and/or associated data);
- control information for Virtual Presence employed in protected processing environment nodes located at user sites to ensure that digital events are governed in accordance with the collective rights of commerce model participants;
- digital events that launch or require other digital events;
- digital events that may include, for example, content use consequences such as collection of audit information, secure communication of such information, payment for content use, or satisfaction of any other electronically stated condition;
- events that occur within either the secure setting of a local node, or more widely within the secure environment of a distributed system of nodes;
- the association of Virtual Presence rules and controls with protected information enclosed within one or more electronic content containers to achieve a high order of configurability for Virtual Presence chains of handling and control;

- distribution using VDE that may package both the electronic content and control information into the same VDE container, and/or may involve the delivery to an end-user site of different pieces of the same VDE managed property from plural separate remote locations and/or in plural separate VDE content containers and/or employing plural different delivery means;
- content control information that is partially or fully delivered separately from its associated content to a user VDE installation in one or more VDE administrative objects;
- delivery of portions of said control information from one or more sources;
- making control information available for use by access from a user's VDE installation secure sub-system to one or more remote VDE secure sub-systems and/or VDE compatible, certified secure remote locations;
- use of delivery means that may include electronic data storage means such as optical disks for delivering one portion of said information and broadcasting and/or telecommunicating means for other portions of said information;

- allowing a content provider to deliver different business rules to a large corporate customer, compared with rules delivered to "retail" customers;
- supporting separation of content and Virtual Presence controls to allow a provider to associate different control sets with the same content – and not requiring the provider to create one set of content controls that apply to all types of customers;
- allowing content provider modification over time of rules and controls to reflect sales, new pricing, special discounts, etc. – while limiting this right by rules and controls provided by other parties having more senior rights;
- employing secure object container technology to efficiently implement Virtual Presence chains of handling and control;
- use of software container technology to significantly facilitate the organized dissemination of digital content, including the specialized form of digital content constituting rights control information;
- employing object software technology and using object technology to form containers for delivery of at least in part encrypted or otherwise secured information;

- using containers that contain electronic content products or other electronic information and some or all of their associated permissions (control) information;
- distributing container objects along pathways involving content providers and/or content users;
- securely moving containers between nodes of a VDE arrangement, which nodes operate VDE foundation software and execute control methods to enact electronic information usage control and/or administration models;
- employing delivered containers both for distributing VDE control instructions (information) and/or to encapsulate and electronically distribute content which has been at least partially secured;
- supporting the essential needs of electronic commerce value propositions by uniting fundamental configurability with secure Virtual Presence;
- virtual presence across virtual networks in accordance with the underlying agreement amongst commerce model participants to allow each participant to enjoy secure, reliable electronic automation of commerce models;
- allowing each rights holder's Virtual Presence at a remote site to possess the sole authority to administer or delegate the participant's electronic rights;

- capabilities that contribute to establishing an environment of trusted cooperative governance;
- practical enhancements relating to the establishment of secure event management and the maintenance of secure audit, encryption, budget, and other relevant information;
- control structures for an overall, distributed, secure rights/event administration environment;
- processes for interaction between independently delivered rules and controls, including electronic negotiation;
- creating distributed rights operating systems;
- integrating control processes into host operating environments;
- secure semiconductors to support protected processing environments;
- a secure, programmable, digital event management component architecture in which components are fully assembleable and reusable;
- differing assemblages of components formed to reflect an exhaustive array of commerce model functional capabilities, overall model implementations, and ad hoc event management scenarios;

- support for the full range of digital content types, delivery modes, and reporting and other administrative activities;
- traveling objects;
- smart agents;
- "atomic" load module operation to support "sparse space," cost-effective, secure processing semiconductors;
- smart card and other traveling client nodes;
- creating rights management software container technologies, including extraction, embedding, and other secure container content management processes;
- Chain of Handling and Control generation of secure objects (containers) and associated control information;
- audit reconciliation and usage pattern evaluation processes;
- specialized cryptographic implementations;
- use of a specialized electronic rights and commerce language, unique applications for fingerprinting and/or watermarking technologies, secure control structures, the formulation of new types of metering technologies, reciprocal event management (employing dispersed user sites) for automating web-like commerce models, and many other designs and capabilities;

- mechanisms to persistently maintain trusted content usage and reporting control information through both a sufficiently secure chain of handling of content and content control information and through various forms of usage of electronic information;
- rights management technology supporting persistent, distributed controls;
- means enabling continuing Virtual Presence through Chains of Handling and Control;
- persistency of control as a unique and fundamentally important attribute underlying Virtual Presence and Chain of Handling and Control for enabling true commerce behavior in cyberspace including ad hoc relationships and activities, distributed processes, and reliable enforcement of agreements between parties;
- Persistent Virtual Presence controls that continue to be enforced -- to the extent required by the controls themselves -- as protected digital content is, for example, used and reused, copied and further distributed, extracted and embedded, audited and reported;
- persistency responsive to rules and controls associated with electronic events, that causes new secure content containers to be created automatically by systems and methods supplying the procession of secure transport

vehicles required by Chain of Handling and Control for conveying disseminated content, associated rules and controls, and audit information and payment;

- container creation to carry extracted content, payment tokens, control information, audit information, and the like;
- securely generated containers carrying with them rules and controls stipulated by rules and controls associated with one or more triggered electronic events;
- capabilities for persistency and independent secure delivery and merging of rules and controls that provide technical means for ensuring that dynamic user behavior can be encouraged, rather than discouraged;
- dynamic user behavior encouraged as a critical link in building ad hoc relationships and cost-effectively distributing content, while simultaneously ensuring that rights holders are protected and retain control over their business models;
- enabling ad hoc behavior that frees users from constraints on their conduct resulting from inflexible, first generation technologies;
- support for enterprising behavior that is characteristic of traditional commerce resulting in more efficient and more satisfying electronic commerce experiences;

- general purpose character electronic commerce technologies provided by a combination of important capabilities including component, object oriented, programmable control language; secure specialized container technology; independent delivery of secure control information mechanisms; Chain of Handling and Control persistency of control mechanisms; event driven operating system functions; and the advanced security architecture – allowing multiple simultaneous models to evolve, and practically and efficiently operate;
- general purpose rights and event management architecture that is intrinsically reusable for many simultaneous models -- providing enormous competitive economic advantages over technologies that are essentially single model by design;
- commerce architecture client nodes that are basic pieces of reusable cyberspace infrastructure;
- generalized configurability resulting, in part, from decomposition of generalized requirements for supporting electronic commerce and data security into a broad range of constituent "atomic" and higher level components (such as load modules, data elements, and methods) that may be variously aggregated together to form control methods for commercial electronic agreements and data security arrangements;

- a secure operating environment employing VDE foundation elements along with securely deliverable VDE components that enable electronic commerce models and relationships to develop;
- the unfolding of distribution models in which content providers, over time, can expressly agree to, or allow, subsequent content providers and/or users to participate in shaping the controls for, and consequences of, use of electronic content and/or appliances;
- a very broad range of the functional attributes important for supporting simple to very complex electronic commerce and data security activities;
- electronic information and/or appliance usage control (including distribution), security, usage auditing, reporting, other administration, and payment arrangements;
- capabilities that rationalize the support of electronic commerce and electronic transaction management stemming from the reusability of control structures and user interfaces for a wide variety of transaction management related activities;
- content usage control, data security, information auditing, and electronic financial activities that can be

supported with tools that are reusable, convenient, consistent, and familiar;

- a general purpose Rights Operating System employing a reusable kernel and rights language components that provides the capabilities and integration needed for the advanced commerce operating systems of the future;
- a general purpose, reusable electronic commerce capabilities that all participants can rely on will become as important as any other capability of operating systems;
- such a rights operating system providing rights and auditing operating system functions and other operating system functions -- the rights and auditing operating system functions securely handling tasks that relate to virtual distribution environment;
- secure processing units and/or protected processing environments that provide and/or support many of the security functions of the rights and auditing operating system functions;
- an overall operating system designed from the beginning to include the rights and auditing operating system functions plus the other operating system functions -- or incorporation of the rights and auditing operating system

functions as an add-on to a preexisting operating system providing the other operating system functions;

- operating system integration and the distributed operating systems; and
- a rational approach - a transaction/distribution control standard - allowing all participants in VDE the same foundation set of hardware control and security, authoring, administration, and management tools, for widely varying types of information, business market model and/or personal objectives;

Any or all of these features may be used in combination with the inventions disclosed herein.

Brief Description of the Drawings

These and other features and advantages will be better and completely understood by referring to the following detailed description of presently preferred example embodiments in accordance with the drawings, of which:

Figures 1A-4 show "prior art" examples of how it is hard to find things you need or want;

Figures 5-12 are simplified examples of what example systems, methods and techniques in accordance with these inventions can do;

Figures 13, 14 and 14A show an example matching and classification utility system architecture;

Figures 15-15G show examples of how a matching and classification utility system can interact with other commerce utility systems;

Figures 16A-16C show examples of distributed matching and classification utility system organizations;

Figure 17 shows example matching and classification utility system functionality definitions;

Figures 18-46(B) show example steps that may be performed by the example matching and classification utility system; and

Figures 47-70 show some example matching and classification utility system applications.

Detailed Description Of Presently Preferred Example Embodiments

Figures 5-12 and the discussion above provide an introduction to the following detailed description of presently preferred embodiments in accordance with these inventions. The "electronic matchmaker" shown in Figures 5-12 is implemented in these more detailed embodiments by a matching and classification utility system 900.

Example Matching And Classification Utility

Figure 13 shows an example matching and classification utility system 900 as including:

- an object classifier 902;
- a user (people) classifier 904; and
- a matching engine 906.

Object classifier 902 classifies things. User classifier 904 classes people. Matching engine 906 matches things with other things, things with people, and/or people with other people.

In more detail, object classifier 902 receives information about objects and uses that information to classify those objects into groups based on the qualities or characteristics of the objects. For example, the object classifier 902 may classify objects of the type described in in "Ginter et al". Such objects may comprise information and/or associated rules for using the information. For example, object classifier 902 may receive as inputs:

- rights management information 909 such as rules and/or associated consequences;
- things 908 controlled or affected by such rights management information including, for example content objects or other information subject to such rules;
- items 910 such as metadata, abstracts or the like that describe the things 908; and/or

- other information of any type.

Object classifier 902 classifies and/or selects things based at least in part on these inputs.

In this example, user classifier 904 is a type of object classifier that is specially adapted to classify people. User classifier 904 can classify people based, for example, on:

- audit trails 912 indicating how people have used their computers and other electronic appliances;
- profiles 914 developed by asking users questions about their preferences;
- controls 909' that are associated, at least in part, with the user or things the user uses;
- object descriptors 910' that describe objects used by the user; and/or
- other information about and/or relating to the user.

User classifier 904 classifies and/or selects people based at least in part on these inputs.

Matching engine 906 receives, as inputs, the classifications and/or selections made by the object classifier 902 and/or the user classifier 904. Matching engine 906 matches things with things, things with people and/or people with people (or any combination of these) based on these selection and/or classification inputs.

Example More Detailed Architecture

Figure 14 shows a more detailed architectural diagram of matching and classification utility 900. In this example, matching and classification utility 900 receives a variety of inputs including, for example, some or all of the following:

- objects 908 and/or information about objects including controls 909 and/or object descriptors 910;
- content 950;
- audit trail information 916;
- user information such as profiles 914;
- class information 952;
- user information 954;
- other rights management information 956;
- matching criteria 958;
- selection criteria 960; and/or
- other information.

Matching and classification utility 900 in this example can provide a variety of different outputs including, for example, some or all of the following:

- matching information 920;
- class hierarchies 962;
- category definitions 922 and class definitions 970;
- classified objects 908C;
- audit records 964 indicating the results of classification, matching, and or selecting processes;

- reports 966 indicating the results of classification, matching, and/or selecting processes;
- targeted objects and/or pointers 968;
- controls 909;
- other rights management information; and
- other classification, matching and/or selection related information.

A Preferred Embodiment Matching and Classification Utility 900 is a VDE-Aware Commerce Utility System

In the preferred embodiment, matching and classification utility 900 is constructed as a commerce utility system 90 as described in "Shear et al", and may comprise one or more processes securely distributed over one or more secure electronic appliances within a "Virtual Distribution Environment" as described in "Ginter et al". Furthermore, the present inventions can be used in combination with and/or make use of a wide array of distributed electronic administrative and support services that may be referred to as the "Distributed Commerce Utility." Such a Distributed Commerce Utility may be, among other things, an integrated, modular array of administrative and support services for electronic commerce and electronic rights and transaction management. The Distributed Commerce Utility provides, among other advantages, comprehensive, integrated administrative and support services for secure electronic commerce and other forms of electronic interaction. These

administrative and support services can be used to supply a secure foundation for conducting financial management, rights management, certificate authority, rules clearing, usage clearing, secure directory services, and other transaction related capabilities functioning over a vast electronic network such as the Internet and/or over organization internal Intranets, or even in-home networks of electronic appliances. Such electronic interactions supported by the Distributed Commerce Utility may, for example, entail the broadest range of appliances and distribution media, non-limiting examples of which include networks and other communications channels, consumer appliances, computers, convergent devices such as WebTV, and optical media such as CD-ROM and DVD in all their current and future forms.

These administrative and support services can, for example, be adapted to the specific needs of electronic commerce value chains in any number of vertical markets, including a wide variety of entertainment applications. Electronic commerce participants can, for example, use these administrative and support services to support their interests, and/or they can shape and reuse these services in response to competitive business realities. Non-exhaustive examples of electronic commerce participants include individual creators, film and music studios, distributors, program aggregators, broadcasters, and cable and satellite operators.

The Distributed Commerce Utility can, for example, make optimally efficient use of commerce administration resources, and

can, in at least some embodiments, scale in a practical fashion to optimally accommodate the demands of electronic commerce growth. The Distributed Commerce Utility may, for example, comprise a number of Commerce Utility Systems. These Commerce Utility Systems can provide a web of infrastructure support available to, and reusable by, the entire electronic community and/or many or all of its participants. Different support functions can, for example, be collected together in hierarchical and/or in networked relationships to suit various business models and/or other objectives. Modular support functions can, for example, be combined in different arrays to form different Commerce Utility Systems for different design implementations and purposes. These Commerce Utility Systems can, for example, be distributed across a large number of electronic appliances with varying degrees of distribution.

Such a "Distributed Commerce Utility" provides numerous additional capabilities and benefits that can be used in conjunction with the particular embodiments shown in the drawings of this application, non-exhaustive examples of which include:

· Enables practical and efficient electronic commerce and rights management.

· Provides services that securely administer and support electronic interactions and consequences.

· Provides infrastructure for electronic commerce and other forms of human electronic interaction and relationships.

· Optimally applies the efficiencies of modern distributed computing and networking.

· Provides electronic automation and distributed processing.

· Supports electronic commerce and communications infrastructure that is modular, programmable, distributed and optimally computerized.

· Provides a comprehensive array of capabilities that can be combined to support services that perform various administrative and support roles.

· Maximizes benefits from electronic automation and distributed processing to produce optimal allocation and use of resources across a system or network.

· Is efficient, flexible, cost effective, configurable, reusable, modifiable, and generalizable.

· Can economically reflect users' business and privacy requirements.

· Can optimally distribute processes -- allowing commerce models to be flexible, scaled to demand and to match user requirements.

· Can efficiently handle a full range of activities and service volumes.

· Can be fashioned and operated for each business model, as a mixture of distributed and centralized processes.

· Provides a blend of local, centralized and networked capabilities that can be uniquely shaped and reshaped to meet changing conditions.

· Supports general purpose resources and is reusable for many different models; in place infrastructure can be reused by different value chains having different requirements.

· Can support any number of commerce and communications models.

· Efficiently applies local, centralized and networked resources to match each value chain's requirements.

· Sharing of common resources spreads out costs and maximizes efficiency.

· Supports mixed, distributed, peer-to-peer and centralized networked capabilities.

· Can operate locally, remotely and/or centrally.

· Can operate synchronously, asynchronously, or support both modes of operation.

· Adapts easily and flexibly to the rapidly changing sea of commercial opportunities, relationships and constraints of "Cyberspace."

Any or all of these features may be used in combination with the inventions disclosed herein.

In more detail, as shown in Figure 14A, matching and classification utility 900 may include one or more rights operating system layers 90-1; one or more commerce utility support service layers 90-4; one or more service application connect layers 90-3; and one or more service functions 90-B. One or more protected processing environments 154 may be used to support secure functions 90-D. Matching and classification utility 900 may be controlled, at least in part, by rights management information such as for example:

- VDE-compatible controls 909;
- rules and/or their consequences; and/or
- other rights management information.

Matching and Classification Utility Can Interact With Other Commerce Utility Systems

Figure 15 shows that matching and classification utility 900 can interact and interrelate with other commerce utility systems described in "Shear et al" including for example:

- financial clearinghouses 200,
- usage clearinghouses 300,
- rights and permissions clearinghouses 400,

- certifying authorities 500,
- secure directory services 600,
- transaction authorities 700,
- VDE administrators 800, and/or
- other commerce utility systems 90.

Figures 15A-15G show example detailed interactions between matching and classification utility 900 and these various other commerce utility systems 90.

Figure 15A shows interactions between matching and classification utility 900 and a financial clearinghouse 200. For example, matching and classification utility 900 may send the financial clearinghouse 200:

- requests for information,
- class information such as classes and/or class assignments,
- bills and charges, and/or
- other information.

Financial clearinghouse 200 may send matching and classification utility 900:

- money,
- audit records,
- payment data,
- user data, and/or
- other information.

Figure 15B shows example interactions between matching and classification utility 900 and usage clearinghouse 300. Matching and classification utility 900 may send the usage clearinghouse 300:

- requests for information,
- class information such as classes and/o class assignments,
- audit information, and/or
- other information.

Matching and classification utility 900 may receive from usage clearinghouse 300:

- requests for class information,
- usage and/or rights management information,
- audit records, and/or
- other information.

Figure 15C shows example interaction between matching and classification utility 900 and rights and permissions clearinghouse 400. In this example, rights and permissions clearinghouse 400 sends matching and classification authority 900:

- controls sets and/or object information;
- requests for class information;
- clearinghouse usage data; and/or
- other information.

In this example, matching and classification utility 900 sends the rights and permissions clearinghouse 400:

- rights management information such as control sets,
- requests for information,
- class related information such as classes and/or class assignments, and/or
- other information.

Figure 15D shows example interaction between matching and classification utility 900 and certifying authority 500. In this example, certifying authority 500 sends matching and classification utility 900:

- revocation lists,
- certificates,
- certifying authority usage information,
- requests for classification information, and/or
- other information.

In this example, the matching and classification utility 900 sends the certifying authority 500:

- revocation list checks,
- requests for certificates,
- requests for usage information,
- classification related information such as classes and/or class assignments, and/or
- other information.

Figure 15E shows an example interaction between the matching and classification utility 900 and a secure directory services 600. In

this example, the matching and classification utility 900 sends the secure directory services 600:

- directory lookup information,
- class related information such as classes and/or class assignments,
- requests for information, and/or
- other information.

In this example, the secure directory services 600 sends the matching and classification utility 900:

- directory services usage information,
- directory information,
- requests for classification information, and/or
- other information.

Figure 15F shows an example interaction between the matching and classification utility 900 and a transaction authority 700. In this example, the matching and classification utility 900 sends the transaction authority 700:

- class related information such as classes and/or class assignments,
- requests for transaction usage information,
- requests for control sets, and/or
- other information.

In this example, the transaction authority 700 sends the matching and classification utility 900:

- transaction usage information,
- transaction control sets,
- requests for classification information, and/or
- other information.

Figure 15G shows an example interaction between the matching and classification utility 900 and a VDE administrator 800. In this example, the matching and classification utility 900 sends the VDE administrator 800:

- requests for administration,
- class related information such as classes and/or class assignments,
- requests for node and/or web information, and/or
- other information.

In this example, the VDE administrator 600 sends the matching and classification utility 900:

- requests for classification information,
- administrative information,
- node and/or user data, and/or
- other information.

Matching and Classification Utility System Can Be In a Hierarchy of Commerce Utility Systems

Figure 16A shows an example of an administrative and support service hierarchy including matching and classification utility system(s) 900. In this example, a number of centralized overall

matching and classification utility systems 900 and/or other Commerce Utility Systems 90 delegate some or all of their work responsibilities to other Commerce Utility Systems 90. In the particular example shown, Commerce Utility Systems 154 may provide services to one or more members of one or more classes, for example, to members of the class "manufacturing companies in the Pacific rim." Organizations, such as companies, non-profit groups or the like may have their own Commerce Utility Systems 156. Certain electronic commerce or other activities (the entertainment industry, for example) might have their own vertically-specialized Commerce Utility Systems 158. Certain geographical, territorial or jurisdictional groups (e.g., Commerce Utility Systems services provided with a particular nation or state within nation, one example of which might be all purchasers of particular products within the state of Wisconsin) may have their own territorial/jurisdictional specialized Commerce Utility Systems 160. Commerce Utility Systems 154, 156, 158, 160 lower in the hierarchy may, in turn, further delegate authorities or responsibilities to particular consumers, organizations or other entities.

In one example arrangement, the Commerce Utility Systems 90 to which authority has been delegated may perform substantially all of the actual support work, but may keep the delegating Commerce Utility Systems 90 informed through reporting or other means. In another arrangement, the delegating Commerce Utility Systems 90 have no involvement whatsoever with day to day activities of the

Commerce Utility Systems to whom they have delegated work. In still another example arrangement, the more specialized Commerce Utility Systems do some of the work and the more overarching Commerce Utility Systems do other parts of the work. The particular division of work and authority used in a particular scenario may largely depend on factors such as efficiency, trustedness, resource availability, the kinds of transactions being managed, and a variety of other factors. Delegation of clearing authority may be partial (e.g., delegate usage aggregation but not financial or rights management responsibilities), and may be consistent with peer-to-peer processing (e.g., by placing some functions within consumers' electronic appliances while keeping some other functions centralized).

Matching and Classification Utilities Can Provide Services to Classes of Nodes, Users, Content Services and/or Transaction Services

Figure 16B shows an example of how Matching and Classification Utilities 900 can provide services to classes of nodes, users, content services and/or transaction services. In this example, matching and classification utility systems 900(1), ... 900(N) provide horizontally specialized matching and/or classification services for different purposes. For example, matching and classification utility 900(1) serves VDE administrative type functions by classifying VDE deployment related information and associated objects. Matching and classification utility 900(2) specializes in higher education classification tasks. Matching and classification utility 900(3)

specializes in business information related tasks, and matching and classification authority 900(N) specializes in trading transactions. Any of these specialties can be combined together, so that a single utility system 900 can perform multiple functions or portions of functions.

Multi-Function Commerce Utility Systems Can be Organized Hierarchically or Peer-to-Peer

Figure 16C shows a still different, more complex Matching and Classification Commerce Utility System 900 environment including elements of both a hierarchical chain of command and a high degree of cooperation in the horizontal direction between different multi-function matching and classification utility systems 900. In this example, there are five different levels of responsibility with a master or overarching matching and classification utility system 900(1) on level 1 having the most authority and with additional matching and classification utility systems on levels 2, 3, 4, and 5 having successively less power, authority, control, scope and/or responsibility. Figure 16C also shows that different matching and classification utility systems 900 on the same level may have different functions, scopes and/or areas of responsibility. For example:

- a Matching and classification utility system 900(2)(1) may be a "type A" Matching and classification utility system,
- Matching and classification utility system 900(2)(2) might be a "type B" Matching and classification utility system, and

- Matching and classification utility system 900(2)(3) might be a "type C" Matching and classification utility system.

On the next level down, Matching and classification utility systems might be type A Matching and classification utility system (such as, 900(3)(1) and 900(3)(2)), they might be type B Matching and classification utility systems (such as, 900(3)(4)), they might be type C Matching and classification utility systems (such as, 900(3)(5), 900(3)(6)), or they might be hybrids -- such as, Matching and classification utility system 900(3)(3) which is a hybrid having type A and type B functions. Figure 16C also shows that additional clearinghouses on levels 4 and 5 might have sub-types as well as types.

A matching and classification utility 900 might break out along content classes (e.g., movies; scientific, technical and medical; and software). Subtype A might include first run movies, oldies, and art films; subtype B might handle journals and textbooks; and type C might be responsible for games, office, educational content. Peer-to-peer communications between clearinghouses could involve differing classes of consumers, differing jurisdictional classes, differing payment methods classes, and/or any other class distinction.

Matching and Classification Utility System Can Be Constructed From Object-Oriented Service Functions

Figure 14A shows Matching and Classification Utility 900 can be constructed from service functions. Figure 17 shows in more

detail how a matching and classification utility system 900 can be constructed based on service functions such as for example:

automatic class generation,

automatic matching,

automatic class assignment,

class based searching,

class based directory,

audit by class,

market research,

rights management language processing,

other service functions.

Example Detailed Steps Carried Out By Matching and Classification Utility System 900

The next section of the specification describes some example steps performed by the matching and classification utility 900.

Example Steps to Categorize Objects and/or Users and/or Appliances

Figure 18 shows example steps to categorize objects, and Figure 19 shows example steps to categorize users 95 and/or

appliances 100. The overall categorization steps in these examples are -- at this level -- similar to one another. The processes begin by getting input data (Figure 18, block 1840, Figure 19, block 1840′). Next, a classification and/or categorization method is selected (Figures 18, block 1842; Figure 19, block 1842′). The process then assembles a data matrix and applies the selected classification method to the data matrix (Figure 18, blocks 1844, 1846; Figure 19, blocks 1844′, 1846′). In addition or alternatively, other data reduction methods may be used (Figure 18, block 1848; Figure 19, block 1848′). Next, the process assigns objects and/or users and/or appliances to the categories developed by the classification method that has been applied (Figure 18, block 1849; Figure 19, block 1849′). Finally, the process stores the results in electronic and/or non-electronic storage in the "write output data" step (Figure 18, block 1850; Figure 19, block 1850′).

The "get input data" step 1840, 1840′ may involve obtaining attribute and/or parameter data from various sources including, for example:

- electronic appliance related attribute data;
- user demographic data;
- user psychographic data;
- available rights management rules and/or consequences (e.g., permissions records);

- exercised rights management rules and/or consequences (e.g., permissions records);
- rights management and/or other audit and/or usage records;
- any third party source of any information, including rights management, usage, audit, statistical, personal, organizational, political, economic, social, religious, business, government, medical, research, academic, literary, military, and/or information and/or data in any format known or unknown concerning any and all other topics that may contribute to the definition of at least one class and/or the assignment of at least one object to a class.

Detailed example steps for harvesting this data are detailed below in connection with Figures 24-46B. This resulting attribute data may be accumulated and aggregated together to form a composite record used as the input to the classification process.

Figure 20 shows an example composite record 1852. This composite classification record may contain attributes derived from any or all of a variety of rights management and/or other data "harvesting" processes. For example, composite record 1852 may include demographic and/or psychographic data obtained by querying the user 95. It may contain usage data obtained by monitoring audit information produced by various usage transactions. It may contain information reflecting user choices concerning rights management

information, the rights management information available to particular users and/or objects, and rights management processes actually performed with respect to particular users and/or particular objects. The information may be analyzed first to provide statistical and/or other summary information, or individual, more granular information may be provided. The composite record 1852 may also contain attributes of particular electronic appliance 100 installations. The particular example composite record 1852 shown in Figure 20 is one non-limiting example composite attribute record containing attributes obtained through a number of different "harvesting" processes. The composite record 1852 may be organized in a way to allow easy and efficient selection of desired attributes in the course of a database lookup, for example, and to allow easy and efficient selection and/or coding as input to any aspect of a classification and/or the assignment of one or more objects to at least one or more classes.

The Figure 21 example cluster analysis process is one example of steps that may be performed as part of the "apply classification method(s)" block 1846, 1846′ of Figures 18, 19. (A classification method, or any other method described in these processes, may be utilized as part of a "knowbot", "agent", "traveling agent", and/or "smart agent", a non-limiting example of which is described in "Ginter et al", for example, Figure 73.) In this particular example, the process selects variables and cases (blocks 1860, 1862, Figure 21), and then assembles an appropriate data matrix (block 1864). A

conventional cluster analysis is then applied (block 1866, Figure 21). The clusters may be interpreted to determine what they mean (Figure 21, block 1868), or they may be compared with previous results and if sufficiently similar, they may be assumed to reflect the same classes as the earlier classification procedure thus minimizing the need for additional interpretation of the clustering results. Step 1868 may be performed automatically or manually, or a combination of automatic and manual processing may be used. Finally, individual cases may be assigned to individual clusters to complete the classification process (Figure 21, block 1870).

Figures 22, 23 show two examples of classification outputs produced by the Figure 21 process. In the Figure 22 example, information from several individuals has been used to create two example categories that reflect differing use profiles. More classes may have been defined than those example classes shown here. Users assigned to the same class have many more features, behavior, and/or other attributes in common than each of them does with members assigned to other classes.

In example Figure 22, members of class 1 tend to spend more per content item purchased, travel abroad more frequently, are more interested in national and international news, business and travel information, and generally do not participate in "pay per view" events and/or content consumption. Members of class 1 also tend to add new rights and/or modify existing rights management controls for

content, for instance, to add a markup and redistribute the content in one example, may be less likely to express a religious preference and/or affiliation, and tend to use the Internet as an area for "surfing" and exploration.

Members of class 2 tend to pay less for content purchased, seldom travel abroad, tend to be interested in sports, religious content and events, and are more often consumers of movies than are members of class 1. Members of class 2 are more likely to "pay per view" than are members of class 1, and are much less likely to add new controls to content and/or modify rights acquired. Members of class 2 are more likely to express a religious preference and among those that do, Protestant denominations are more frequently mentioned. Members of class 2 may use the Internet, but tend to do so as part of their work role and responsibilities rather than as entertainment, hobbies, and other leisure-time pursuits.

Some methods of classification produce parameter data rather than assignment of objects to more discrete (or fuzzy or other kinds of) classes. Instead, this parameter data may indicate the extent to which an object possesses one or more traits, attributes, or class characteristics. For instance, a person may have been assigned to class 1 (call it "the cosmopolitan class") or class 2 (call it "the parochial class") as shown in Figure 22; however, using other procedures the same example persons may be assigned parameter data

reflecting the extent or degree to which they are "cosmopolitan" or "parochial" or some of each.

In the example process that generates the information shown in Figure 23A, data for several individuals has been arranged in a case (row) by variable (column) matrix and using means known to those skilled in the arts, subjected to principal components analysis with subsequent Varimax axis rotation. Components with eigenvalues >1.0 were retained for subsequent rotation and use. After rotation, each case was assigned a score on each retained (and rotated) component. Each score indicates the extent to which the case has the characteristic represented by the component.

The hypothetical data in Figure 23A shows how strongly each variable (the column of the input matrix) is correlated with the underlying characteristic or component. For example, "region of the US" and "Family income" are highly correlated while "owns a sports utility vehicle" is not.

Using results such as these plus the input data matrix, a score is assigned to each case indicating the extent to which they posses the trait, attribute, characteristic indicated by each factor or component. The hypothetical data in Figure 23B shows how strongly each case -- a person or thing – is a member of the class, and/or possesses the underlying variable represented by each component. A higher score shows that example case 1 has more of the underlying component 1 than does example case 3, whose score is close to zero. Components

(factors) may be bipolar with a zero point and cases whose scores may be positive, negative or zero. Hypothetical example case 5 has a negative score on this component.

This component score information may be used by the matching and classification utility 900 to define certain other classes, such as "the class consisting of the top 5% of those who are cosmopolitan," that is, the 5% with the highest scores on example component 1. The original scores and/or derivative class assignments may be included on attribute records with attribute and/or class information harvested from other sources and/or through other processes.

DATA HARVESTING

Example Steps For Collecting Appliance Related Data

Figure 24 shows example steps performed by the matching and classification utility 900 to collect appliance attribute data. In this example, an electronic appliance 100 may have certain information associated with it. For example, a VDE administrator 800 may initialize appliance 100 with certain information upon appliance installation. In this example, the matching and classification utility 900 can collect this appliance attribute data and use it as part of a matching and/or classification and/or selection process. As shown in Figure 24, the matching and classification utility 900 may initially specify desired appliance attribute fields or other information characteristics the utility is going to collect (Figure 24, block 1502).

The information to be collected depends upon the purpose and use to which the matching and classification utility 900 is to put the information to. The matching and classification utility 900 may use a data dictionary or other mechanism for specifying the desired types of appliance information it is going to collect.

The matching and classification utility 900 next determines whether it already possesses the desired information for this particular appliance 100 (Figure 24, block 1504). For example, the information may have been previously gathered as part of a prior process. If the information is already available, the matching and classification utility 900 sends one or more events to a "create appliance attribute record" method to process the previously gathered data (Figure 24, block 1506). (In all these processes, if the appropriate method is has been sent previously to a VDE installation, only the associated administrative events necessary to activate the method need to be sent in the VDE container.) Alternatively, if the desired data is not already available ("no" exit to decision block 1504, Figure 24), the matching and classification utility 900 performs the other steps shown in Figure 24 to collect the appliance attribute data.

These collecting steps shown in Figure 24 may include sending a VDE container 152 with a "create appliance attribute record" method, and one or more associated administrative events to activate the method, to the VDE administrator 800 (Figure 24, block 1508). The next step (Figure 24, block 1510) may be performed by the VDE

administrator 800 processing the administrative event(s) using the "create appliance attribute record" method to determine whether the administrator already has the desired information for the particular electronic appliance 100. If the operation is successful ("yes" exit to decision block 1512, Figure 24), the VDE administrator 800 may send, to the matching and classification utility 900, a VDE container 152 containing one or more administrative events and the appliance attribute record (Figure 24, block 1514). If the operation is not successful ("no" exit to decision block 1512, Figure 24), the "create appliance attribute record" method operating at VDE administrator 800 may, in this example, collect the data directly from the electronic appliance 100 by sending a VDE container to the appliance, the container containing a "create appliance attribute record" method and one or more associated administrative events (Figure 24, block 1516). The appliance 100 may itself process the administrative event(s) using the "create appliance attribute record" method (Figure 24, block 1518) to produce the required appliance attribute record. Appliance 100 may then send a VDE container 152 containing the appropriate administrative event(s) and the appliance attribute record to the matching and classification utility 900 (Figure 24, block 1520).

In another example, blocks 1508-1514 may be bypassed entirely, and the matching and classification utility 900 may (assuming appropriate authorizations are in place) perform block 1516 to send a container 152 with one or more administrative events

and the "create appliance attribute record" method directly to the electronic appliance 100.

Figures 25(A) and 25(B) together show example steps performed by the "create appliance attribute data" method shown in Figure 24, blocks 1506, 1510 and 1518. As disclosed in "Ginter et al", the actual processing steps are performed by one or more load modules associated with the method. This example method (which, as explained above, may be performed by the matching and classification utility 900, the VDE administrator 800, the electronic appliance 100, any other electronic appliance, or a combination of any or all of these) first locates the site configuration record(s) corresponding to the electronic appliance for which appliance attribute data is to be collected (Figure 24A, block 1522). This site configuration record(s) may, for example, be stored in the electronic appliance secure database. The method next locates the permissions record for the site configuration record(s) (Figure 24A, block 1523). The SPE next determines, based upon the permission record(s), whether the method has permission to access and/or use the site configuration record(s) (Figure 25A, block 1524). If the method does not have the appropriate permission ("no" exit to decision block 1524, Figure 25A), the protected processing environment 154 reports the failure and reason for the failure, and the method writes an associated audit record (Figure 25A, block 1525, 1526) and goes on to process a user configuration record(s). On the other hand, if the method does have permission to use the site configuration record(s) ("yes" exit to

decision block 1524, Figure 25A), the method copies the required fields from the site configuration record(s) to create an appliance attribute record, and may then write an appropriate audit record (Figure 25A, block 1527).

After completing processing of site configuration records, the method then locates the user configuration record(s) corresponding to the electronic appliance for which appliance attribute data is to be collected (Figure 25B, block 1528). This user configuration record(s) may, for example, be stored in the electronic appliance secure database. The protected processing environment 154 next locates the permissions record for the user configuration record(s) (Figure 25B, block 1529). The protected processing environment 154 determines next, based upon the permission record(s), whether it has permission to access and/or use the user configuration record(s) (Figure 25B, block 1530). If the method does not have the appropriate permission ("no" exit to decision block 1530, Figure 25B), the protected processing environment 154 reports the failure and reason for the failure, and the method writes an associated audit record (Figure 25B, block 1531, 1532) and exits the process. On the other hand, if the method does have permission to use the user configuration record(s) ("yes" exit to decision block 1530, Figure 25B), the method copies the required fields from the user configuration record(s) to create an appliance attribute record, and may then write an appropriate audit record (Figure 25B, block 1533). The method may then, if desired, create a new permissions record corresponding to the appliance

attribute record (Figure 25B, block 1534). If a new permissions record is desired, the method may include appropriate "shared secrets," expiration interval(s), and/or other data in an associated MDE to, for example, provide a basis for controlling access, use, and modification of the permissions record.

Figures 26A-26C show examples of appliance attribute records created by Figure 25B, block 1532. Figure 26A shows an example appliance attribute record that may include, for example, an appliance identification field 1536(1) and any number of attribute fields 1538(1)...1538(n). Figure 26B shows a more specific appliance attribute record example including an appliance ID field 1536(1), an operating system field 1538(A), a country field 1538(B), a state field 1538(C), a VDE administrator organization field 1538(D), a VDE version field 1538(E), and a VDE maintenance level field 1538(F). Figure 26C shows that different encodings may be used for any/all of the various attribute fields 1538.

Example Steps for Collecting Demographic Data

Figures 27A, 27B show example steps for collecting demographic data. In this example, the matching and classification utility 900 initially specifies demographic data fields it is interested in (Figure 27A, block 1540). The matching and classification utility 900 next determines whether the required data is already available to it (e.g., based on previous inquiries responded to by the user 95) (block 1542, Figure 27A). If the required data is already available ("yes"

exit to decision block 1542, Figure 27A), the matching and classification utility 900 may send one or more events to a "create demographic attribute record" method to process the data (block 1544, Figure 27A).

On the other hand, if the required data is not available to the matching and classification utility ("no" exit to decision block 1542, Figure 27A), the matching and classification utility may send a container 152 to another commerce utility system 90, the container including one or more administrative events associated with a "demographic data query" method and a "create demographic attribute record" method (Figure 27A, block 1546). The other commerce utility system 90 may then process the one or more events using the "demographic data query" method, and write an associated audit record (Figure 27A, block 1548). It may determine whether the required demographic data is available (Figure 27A, block 1550). If the information is available ("yes" exit to decision block 1550, Figure 27A), the commerce utility system 90 may process one or more events using a "create demographic attribute record" method in order to analyze the available demographic data, and write a corresponding UDE audit record (Figure 27A, block 1552). The other commerce utility system 90 may then send appropriate one or more administrative events and the demographic data attribute record within a container 152 to the matching and classification utility 900 (Figure 27A, block 1554)).

If the required demographic data is not available ("no" exit to decision block 1550, Figure 27A), the commerce utility system 90 may send an administrative event to the matching and classification utility system 900 within a container 152 informing the matching and classification utility that the required data is not available (Figure 27B, block 1556). The matching and classification utility 900 may then send a "demographic data query" method and a "create demographic attribute record" method within a container 152 (along with appropriate administrative events to activate such methods) directly to the user 95 about which demographic information is to be collected (Figure 27B, block 1558). The user's electronic appliance 100 may, in response, process the one or more events using the "demographic data query" method, which may write an associated audit record (Figure 27B, block 1560). If the required data is not collected ("no" exit to decision block 1562, Figure 27B, the user's appliance 100 may send a "failure" message associated with the appropriate administrative event to the matching and classification utility 900, and write an associated audit record (Figure 27B, block 1564, 1566). If the required demographic data is successfully collected ("yes" exit to decision block 1562, Figure 27B), the user's electronic appliance may process one or more events using the "create demographic record" method supplied by step 1558, which may write an associated audit record (Figure 27B, block 1568). The electronic appliance may then send appropriate administrative events and the

demographic attribute record to the matching and classification utility within one or more containers 152 (Figure 27B, block 1570).

Figure 28 shows an example questionnaire "pop-up" screen that may be displayed by the user's appliance 100 as a result of processing events using the "demographic data query" method of block 1548, Figure 27A, and/or block 1560, Figure 27B. In this example, information is collected directly from a user 95 by displaying a questionnaire on a display device that is part of the user's appliance 100. The questionnaire may ask for various demographic information such as:

- name
- address
- city
- state
- zip code
- gender
- date of birth
- education level
- marital status
- number of children

- age of first child
- gender of first child
- other information

The user is requested to provide the information by filling in the various fields within the questionnaire. The questionnaire may assure the user that all information the user provides will be treated as confidential, by, for example, disclosing the rules that will be associated with access to and use of the information.

Steps similar to those shown in Figure 25A, 25B may be performed to create a demographic attribute record based on the results of a demographic data query. Figure 29A-29C show examples of different user demographic attribute information records resulting from this process. Figure 29A shows an example demographic attribute record 1572 including a user ID field 1574 and any number of attribute fields 1576(1), ... 1576(n). Figure 29B shows a more specific example of a demographic attribute record including, for example, a user ID number 1574, a gender attribute field 1576(A), an age field 1576(B), a highest educational level field 1576(C), a citizenship field 1576(D), a country of residence field 1576(E), a district field 1576(F), a city field 1576(G), and a street address field 1576(H). Figure 29C shows a different detailed encoding example for demographic attribute record 1572-1.

Example Steps for Collecting Psychographic Data

Figure 20 shows example steps that may be performed to collect user psychographic data. In this particular example, the matching and classification utility 900 initially specifies desired psychographic data it requires in order to perform a particular classification/matching process (Figure 30, block 1580). The matching and classification utility 900 determines if the required data is already available to it (Figure 30, block 1582). If the required data is already available ("yes" exit to decision block 1582, Figure 30), the matching and classification utility 900 sends one or more events to a "create psychographic attribute record" method in order to analyze the available data and provide appropriate psychographic attributes (Figure 30, block 1584). If, on the other hand, the required data is not available to the matching and classification utility 900 ("no" exit to decision block 1582, Figure 30), appropriate steps are performed to collect the required data. In this example, the matching and classification utility 900 may send a "psychographic data query" method and a "create psychographic attribute record" method within one or more containers 152 (along with appropriate administrative events to activate such methods), to appropriate repositories that may contain the required data (Figure 30, block 1586). If the required data is available from the repositories ("yes" exit to decision block 1588, Figure 30), then an electronic appliance at the repository (in this example) processes one or more events using the "create psychographic attribute record" method supplied by block 1586 in

order to generate an appropriate attribute record(s) containing the attribute information the matching and classification utility 900 is interested in (Figure 30, block 1590). This information, and associated one or more events, may be sent to the matching and classification utility 900 within one or more containers 152 (Figure 30, block 1592).

If the required data is not available from the repository ("no" exit to decision block 1588, Figure 30), then the repository may send a "failure" message associated with one or more administrative events to the matching and classification utility 900 within a container 152 (Figure 30, block 1594). The matching and classification utility 900 may, in response, send one or more administrative events, a "collect psychographic data" and "create psychographic attribute record" method directly to the user's electronic appliance 100 within one or more containers 152 (Figure 30, block 1596). The user's electronic appliance 100 may, in turn, process the events using the "collect psychographic data" and "create psychographic attribute record" methods (Figure 30, block 1598, 1600), and send the resulting attribute data record(s) to the matching and classification utility (Figure 30, block 1592).

Figure 31 shows an example psychographic questionnaire "pop-up" screen that may be displayed to the user 95 upon performance of Figure 30, block 1598. This questionnaire may

collect various psychographic information from the user, including for example:

- mood information
- emotion information
- habit information
- behavioral information
- cognitive information
- medical information
- physical information
- patient information
- counseling information
- aptitude information
- testing information
- other information
- combinations of types of information.

The questionnaire may inform the user that all information collected will be treated as "confidential," and may also, if desired, indicate that the user will be compensated for providing the information.

Figures 32A-32C show some example user psychographic attribute information records 1602 that may be created by Figure 30, block 1584, 1590 and/or 1600. Figure 32A shows that a psychographic attribute record 1602 may include a user ID field 1604 and any number of attribute fields 1606(1), ... 1606(n). Figure 32B shows a more detailed user psychographic attribute record 1602 example including a user ID field 1604, a field 1606a indicating whether the user is introverted or extroverted, a field 1606b indicating whether the user is a sensing or intuitive person, a field 1606c indicating whether the user is primarily a thinking person or a feeling person, a field 1606(d) indicating whether the user is primarily a judging person or a perceiving person, and a field 1606(e) indicating an overall psychographic / behavioral profile such as, for example, the iVALS standard provided by SRI. Figure 32C shows a different kind of encoding (in this case, binary) for the various attributes 1606.

Example Method for Determining Attributes Based on Available Rules and Consequences

Figure 33 shows an example method for determining attributes based on available rules and consequences. The matching and classification utility 900 may first send one or more administrative events and a "send permission records" method request to an electronic appliance 100 within one or more containers 152 (Figure 33, block 1610). In response, the appliance may process the events using the method, which may write an associated audit record (Figure 33, block 1612). If this step is performed successfully ("yes" exit to

Figure 33, decision block 1614), the appliance sends appropriate administrative events and the requested permission records to the matching and classification utility 900 within one or more containers 152, and the method writes an associated audit record indicating it has performed this transaction (Figure 33, block 1616). The matching and classification utility may process events using a corresponding "create attribute record from permission records" method to obtain attributes from these provided permission records (Figure 33, block 1618). If the step of block 1612 failed (as indicated by the "no" exit to decision block 1614, Figure 33), the method may send a "failure" message to the matching and classification utility 900, and write an associated audit record (Figure 33, block 1620).

Figure 34 shows a variation on the Figure 33 example in which the appliance 100 rather than the matching and classification utility 900 creates the rules attribute record based on a "create rules attribute record from permissions records" method supplied by the matching and classification utility, and then sends the rules attribute record to the matching and classification utility (see Figure 34, blocks 1622, 1624).

Example Method to Construct Attribute Records from Permissions Records

Figures 35A, 35B show example steps for constructing attribute records from permissions records. The steps shown in Figure 35A, 35B may, for example, be performed as part of the method shown in block 1618 of Figure 33.

In this example method 1618, the matching and classification utility 900 may first check relevant permissions to ensure that it has the authority to perform the desired transactions (Figure 35A, block 1630). For example, the matching and classification utility 900 may examine a permissions record about the permissions records it has collected, this permissions record it is examining indicating what entities have authority to perform operations with respect to the permissions record to be analyzed. Presuming the matching and classification utility 900 has the appropriate permission, it opens a permissions to be analyzed (Figure 35A, block 1632), and performs a sequence of steps 1634-1650 to extract relevant information from the permissions record. For example, information from the permissions record header can be copied into the attribute record (Figure 35A, block 1634), and then the method may locate the rights record header (block 1636, Figure 35A). Information from the rights record header may be copied into the attribute record (block 1638, Figure 35A), along with the identifier for the corresponding right(s) (blocks 1640, 1642, Figure 35A). The process may then recursively locate and harvest data from each method header contained within the rights record (blocks 1644, 1646, 1648, Figure 35B). The process may recursively repeat steps 1638-1648 for each rights record within the permissions record (as tested for by decision block 1650, Figure 35B). Finally, the entire process of steps 1632-1652 may be performed recursively for multiple permissions records to harvest the

appropriate rules and consequences information from each of a number of permissions records (see decision block 1652, Figure 35B).

Figure 36 shows example steps to perform the "check permissions" operation shown in Figure 35A, block 1630. In this example, the process locates the permissions record from which information is desired to be harvested (Figure 36, block 1660), and then determines whether there is a permissions record for that permissions record (Figure 36, decision block 1662). If there is no permissions record that controls that permissions record (and assuming that authorization or additional permission is required to access the permissions record from which information is to be harvested) (Figure 36, "no" exit to decision block 1662), the process reports failure, writes an audit record, and ends (Figure 36, blocks 1664, 1666, 1668). On the other hand, if there is a permissions record that controls access to the permissions record from which information is to be harvested ("yes" exit to decision block 1662, Figure 36), the process determines whether that permissions record for the permissions record enables usage by the matching and classification utility 900 (Figure 36, decision block 1670). If the matching and classification utility 900 does not have permission ("no" exit to decision block 1670, Figure 36), the process reports failure, writes an audit record to that effect, and ends (blocks 1672, 1674, 1676, Figure 36)). On the other hand, if the matching and classification utility 900 is granted permission ("yes" exit to decision block 1670, Figure 36), the process accesses and uses the permissions record for the

permissions record from which information is to be harvested (Figure 36, block 1678).

Figures 37A-37C show examples of attribute records containing information harvested from permissions records. Attribute record 1680-1 shown in Figure 37A includes a user identification field 1682, an object identification field 1684, and any number of attribute fields 1686(1), ..., 1686(n). The attribute record 1680-2 shown in Figure 37B includes, as a more detailed example, a user ID number field 1682, an object ID field 1684, a right ID field 1686a, a method identifier field 1686b, another right ID field 1686c, and corresponding method type fields 1686(d), a further right ID field 1686e and two corresponding method attribute fields 1686f, 1686g, a further right ID field 1686h and corresponding method attribute fields 1686i, 1686j.

Figure 37C shows a different example in coding for the Figure 37B example attribute record.

Example Steps for Assembling Rules and Consequences

Figure 38 shows example steps for assembling rules and consequences. In this example, the matching and classification utility 900 may send one or more administrative events and a "get user rights table" method within a container 152 to an electronic appliance (Figure 38, block 1690). The electronic appliance 100 processes the one or more events using the "get URT" method, which may writes an

associated audit record (Figure 38, block 1692). The method then determines whether the associated URT records are available (Figure 38, decision block 1694). If the records are not available ("no" exit to decision block 1694, Figure 38), the method sends a failure notice to the matching and classification utility 900, and writes an associated audit record (block 1696, Figure 38). If, on the other hand, the URT records are available ("yes" exit to decision block 1694, Figure 38), the method packages the URT records and associated one or more administrative events into a container 152, and sends the container to the matching and classification utility 900 (Figure 38, block 1698). The matching and classification utility 900 may then process the administrative events using a "create attribute record from URT" method in order to extract or harvest the information from the URT(s) (Figure 38, block 1700).

Figure 39 shows another example sequence of steps for assembling rules and consequences. In this example, the matching and classification utility 900 sends one or more administrative events and a "create attribute record from URT" method to the electronic appliance 100 that stores or has access to the user rights table information (Figure 39, block 1702). The appliance then processes the events using the method sent to it, and the method writes associated audit information as it processes (Figure 39, block 1704). If the URT records are available and the step completes successfully ("yes" exit to decision block 1706, Figure 39), the method sends the resulting URT attribute record(s) and one or more administrative

events to the matching and classification utility within a container 152, and writes corresponding audit information to an audit trail (Figure 39, block 1710). On the other hand, if an error condition arises either because the URT records are not available or because the method for some other reason cannot complete successfully, the method sends a failure notice within a container 152, and writes an associated audit record ("no" exit to decision block 1706, Figure 39, block 1708).

Figures 40A, 40B show example steps performed by blocks 1700, 1704 to "create attribute record from user rights table." The method begins by checking associated permissions for the user rights table records (Figure 40A, block 1720). Assuming that appropriate user and/or group permission is available, the method next locates the user rights table (Figure 40A, block 1722), and then begins recursively analyzing the user rights table information to harvest desired attribute information from it (Figure 40A, blocks 1724 and following). In this particular example, the method locates the user rights table record (block 1724, Figure 40A, and then locates the first rights record header within the first user choice record within the URT record (blocks 1726, 1728, Figure 40A). The method copies rights record header information to the attribute record (block 1730), and then locates the right identifier and copies that to the attribute record (blocks 1732, 1734). The method then recursively locates each method header within the user rights table right record, and copies corresponding attribute information to the attribute record

(blocks 1736, 1738, 1740, Figure 40B). Steps 1728-1740 are performed recursively for each rights record within the user choice record (see Figure 40B), decision block 1742), and the above steps are performed recursively for each user choice record within the user rights table (see decision block 1744, Figure 40B). Additionally, steps 1724-1744 are performed recursively for each user rights table record within the user rights table (see Figure 40B, decision block 1746). As a last example step, the method creates a permissions record that controls access and use of the attribute record it has created (Figure 40B, block 1748).

Figure 41 shows example steps performed by the check permissions block 1720 shown in Figure 40A. For example, the sequence of steps may begin by locating a corresponding permissions record (Figure 41, block 1750) and then determining whether there is a permission record corresponding to the corresponding user rights table entry (Figure 41, decision block 1752). If there is no such entry ("no" exit to decision block 1752), the method may report failure, write an audit record, and end (blocks 1754, 1756, 1758, Figure 41). If there is a corresponding permissions record ("yes" exit to decision block 1752, Figure 41), then the permissions record may be examined whether it enables usage for the matching and classification utility 900 (decision block 1760, Figure 41). If the permissions record does not enable usage by the matching and classification utility 900 ("no" exit to decision block 1760, Figure 41), the method may report a failure, write an audit record, and end (blocks 1762, 1764, 1766,

Figure 41). On the other hand, if the matching and classification utility 900 does have the required permissions to enable usage ("yes" exit to decision block 1760, Figure 41), the method may access the permissions record (if any) for the user rights table for use in controlling access to the user rights table itself (block 1768, Figure 41).

Figures 42A-42C show example rights attributes records 1770 that may be obtained from the processes above. The Figure 42A example rights attribute record 1770-1 includes a user or group ID field 1772, an object ID field 1774, and any number of attribute fields 1776(1), ... , 1776(n). The more detailed example rights attribute record 1770-2 shown in Figure 42B includes a user ID number field 1772, an object ID field 1774, a right ID field 1776a and corresponding method attribute field 1776b, another right ID field 1776c and corresponding method attribute field 1776d, a right ID field 1776e and corresponding method attribute fields 1776f, 1776g, and another right ID field 1776h and corresponding method attribute field 1776i.

Figure 42C shows how the rights attribute record 1770 can be encoded numerically as opposed to using characters, as one example.

Example Steps for Assembling Usage Audit Records

Figure 43 shows example steps for assembling usage audit records for purposes of matching and/or classification. In this example, the matching and classification utility 900 may send one or

more administrative events and a "get audit records" method to a VDE appliance 100 within a container 152 (Figure 43, block 1780). The appliance 100 may process the one or more events using the "get audit records" method, which may write an associated audit record (block 1782, Figure 43). If the audit records are not available ("no" exit to decision block 1784, Figure 43), the method may send a failure notice within a container to the matching and classification utility 900, and may then write an associated audit record (Figure 43, block 1786). On the other hand, if the audit records are available ("yes" exit to decision block 1784), the method may send one or more administrative events and the audit records within a container 152 to the matching and classification utility 900, and write an associated audit record (block 1788, Figure 43). The matching and classification utility 900 may then process the one or more administrative events using a "create attribute record from audit record" method in order to extract or harvest the information from the audit record it will use to perform matching and/or classification (block 1790, Figure 43).

Figure 44 shows another sequence of example steps that may be used to assemble usage audit records for purposes of matching and/or classification. In the Figure 44 example, the matching and classification utility 900 sends one or more administrative events and a "create attribute record from audit record" method to an electronic appliance 100 within one or more containers 152 (Figure 44, block 1792). The appliance 100 may then process the one or more administrative events using the "create attribute record from audit

record" method, which may write an associated audit record (block 1794, Figure 44). The method may determine, in this process, whether audit records are available (Figure 44, decision block 1796). If no audit records are available ("no" exit to decision block 1796), the method may send a failure notice to the matching and classification utility 900 (Figure 44, block 1798). On the other hand, if audit records are available, the method may create the corresponding usage attribute records and associated administrative event(s), package them into a container 152, send the container to the matching and classification utility 900, and write corresponding audit records (Figure 44, block 1799).

Figures 45A, 45B show example steps for performing the method (shown in Figure 44, block 1794, for example) of creating attribute record(s) from audit records. In this example, the method first locates the audit records in a secure database or other storage facility (Figure 45(A), block 1800). The method next selects an appropriate UDE audit record to analyze (Figure 45(A), block 1802), and determines whether a permission record is available that applies to this particular audit record (Figure 45(A), decision block 1804). If a permissions record is required and is not available, the process reports failure, writes an associated audit record, and ends (Figure 45 blocks 1806, 1808, 1810). If, on the other hand, a required permissions record is available ("yes" exit to decision block 1804, Figure 45), the process determines whether the permissions record grants the device or process permission to use the audit record(s) for

this particular purpose (decision block 1812, Figure 45). If such permission is not available ("no" exit to decision block 1812, Figure 45A), the process reports failure, writes an associated audit record, and terminates (Figure 45A, blocks 1814, 1816, 1818).

If any applicable permissions record is available and grants permission to the matching and classification utility 900 ("yes" exit to decision block 1812), the process determines multiple audit records need to be analyzed together as an overall event (Figure 45A, decision block 1820). For example, an "atomic transaction" in which multiple steps are performed to achieve an overall result may have multiple audit records (e.g., from multiple appliances 100) that may need to be analyzed together in order to make sense out of the overall transaction. As another example, an object may have subparts (e.g., sub-objects) on which operations can be performed – but it may be important for matching and/or classification purposes to analyze the results of such multiple operations together in order to determine appropriate attribute(s) for matching and/or classification. If it is necessary to aggregate multiple audit records together for analysis (decision blocks 1820, 1822, Figure 45A), then the process proceeds to analyze those audit records together and create corresponding summary transaction information (Figure 45A, block 1824).

The process next determines whether it needs to produce aggregated audit statistics in order to perform the associated matching and/or classification operation (Figures 45A, 45B, decision block

1826). For example, multiple operations may be performed on a certain object. It may be important to know statistics about such operations (e.g., the number of times the object was opened on a certain day, the number of users who opened the object in a certain time period, etc.). If such aggregated statistics are required ("yes" exit to decision block 1826, Figure 45B), the process proceeds to create such aggregated statistics (block 1828, Figure 45B).

The process next copies selected audit record information to an audit attribute record (Figure 45B, block 1830). The process then determines whether it needs to process more audit records (decision block 1832, Figure 45B). If more audit records are required to be processed ("yes" exit to decision block 1832, Figure 45B), control returns to Figure 45A, block 1802 to select the next audit record. Otherwise ("no" exit to decision block 1832, Figure 45B), the process creates a permissions record associated with the newly created attribute record(s) (Figure 45B, block 1834), and completes.

Figures 46A, 46B show example usage attributes/statistic records that the Figure 45A-B process may create. The Figure 46A attribute record 1830-1 may include, for example, a user ID 1832, an object ID 1834, and any number of attribute fields 1836(1), ... , 1836(n). The more detailed Figure 46B example attribute record 1830-2 includes a user ID number 1832, an object ID 1834, a right ID 1836a and associated method characteristic 1836b, another right ID 1836c and associated method 1836d and associated statistic 1836e, a

further right ID 1836f and associated method attribute 1836g, another right ID 1836h and associated methods 1836i, 1836j, and associated additional attributes 1836k-1836o. The characteristics shown in fields 1836k-1836o could, for example, be derived from an aggregate of any number of individual audit records recording individual transactions associated with the object identified in field 1834.

EXAMPLES

The following are some non-limiting examples of how Matching and Classification Utility 900 may be useful in certain applications.

Example: Matching and Classification Utility 900 Can Support Narrowcasting or "Push" Distribution Models Based On Classes

Interactions with content, transactions, and other events on the World Wide Web are mainly driven today by following chains of hypertext links, using various search engines, and/or indexes, to say nothing of just plain luck and persistence, to find interesting and/or useful content and/or services. Time consuming and generally inefficient, these search activities share in common the feature that each consumer must intentionally "pull" desired content from a Web site to their computer after successfully identifying specific content or services of interest at that time. The present inventions also support "pull" models—a topic to be addressed shortly. However, the present

inventions also support narrowcasting or "push" models of content distribution as well.

In one example, the matching and classification utility 900 can facilitate much more automated and therefore more efficient and effective content creation, access and/or distribution services that "push" information and/or services to users. Example Figure 47 shows an example "information push" model 2000 in which an arbitrary number of users 2001(1)-2001(n) each have a VDE node (e.g., a protected processing environment 154) installed on their appliances. These example appliances may be of any kind, including computers, so-called Web television or Web-TV, DVD appliances with some form of backchannel, a settop box with a "back channel", and so on.

Perhaps with the permission of the user or other authority, such as an administrator within an organization, the VDE node collects various usage information or "info exhaust" according to the rules and usage consequences provided by one or more value chain participants. At times specified by default and/or by the associated rules and consequences, audit records are sent, in this example, in VDE containers 2006(1)-2006(n) to a usage clearinghouse 300, which in turn, may send all or a portion of these audit records in a VDE container 2008 to the matching and classification utility 900. The audit records may contain rights management information, including, but not limited to the amount of usage, the amount paid, if any, the

payment method used, if any, VDE control sets, and/or data that identify various attributes of the node, user, and/or known and/or used object(s). The audit records may also contain information about objects known to the VDE node (objects with PERC records - see Figures 35A, 35B and associated discussions) and/or objects that have been used (objects with URT entries - see Figures 40A-40B and associated discussions) on the node.

The matching and classification utility 900 may also receive from one or more providers 2010 content objects 2003 themselves, for example, information in text format and/or metadata 2005 associated with content objects. Using at least one classification method, the matching and classification utility 900 may create at least one object class hierarchy, object class, object classification scheme, object category and/or object category scheme using at least some rights management information and assign at least one object to at least one category and/or class.

The matching and classification utility 900 takes the usage information and other rights management information received from the VDE nodes and/or other information sources and may create at least one category and may assign at least one node and/or user to a cateogry and/or class. In Figure 47, the matching and classification utility 900 sends a VDE container 2002 to content provider 2010 with information showing the classes of content used by one or more nodes and/or users along with a request that the provider 2010 send similar

content back to one or more users 2001. At least one content provider 2010 then sends at least one VDE container 2004 to user A with content and/or information about available content that may be of interest to user A given the history of content usage as reflected in VDE audit records and/or other rights management information. In this "push" example, classes of content or information about available content may be pushed automatically from (a class of) content providers to one or more members of class of users and/or nodes. Consequently, users do not have to search as intensely, if at all, for content of interest to them.

In this example, user A receives content that may be most like content the user has already used, perhaps like content used most frequently in the recent past. The present inventions also support the matching and classification utility 900 and/or content provider sending content that is in a class or classes more distant from topics of prior and current interest to a particular user and/or group of users. Certain classification methods familiar to those skilled in the arts may provide quantitative indicators of distance that, in turn, may be used as at least one criterion for selection.

In another example, matching content to users and/or nodes may be based in part on class assignments that are in turn based in part on information concerning user preferences solicited by the matching and classification utility 900 or other value chain participant, such as a market research firm, advertising agency,

provider, distributor, VDE administrator 800, or other Commerce Utility System.

Although the matching and classification utility 900 and/or content provider may send "more of the same," in another example the present inventions support providers at least occasionally sending content more distantly related to the user's apparent interests to determine if the user's circle of interest might be a little larger than that indicated by past usage and other, related rights management information alone.

In another example, providers may from time to time send content unrelated to the user's apparent interests that may nevertheless reflect the interests of persons and/or groups sharing at least one attribute with the user. For instance, the matching and classification utility 900 may, by sending a VDE container with appropriate user and content class information, suggest to a provider that user A receive content similar to content used by another member or members in the same group or class as user A. In one example, the matching and classification utility 900 may suggest sending business information related to a particular vertical market segment because others in the same class as user A have paid attention to that market.

In support of various content narrowcasting or "push" models, the matching and classification utility 900 may provide content class related information to a "subject switch" or "subject mapper," which in turn, matches participants desiring information in one or more

specified classes with one or more sources of content in the requested class or classes.

The non-limiting subject switching example 2050, Figure 47A, shows a number of customers 2053(1)-2053(n) each with an appliance 2052(1) -2052(n) such as a personal computer. Other arrangements may include appliances such as a WebTV interface and/or an intelligent "settop box" connected to an interface device that uses one or more (digital) TVs for display. Still other arrangements may include an NC computer without a local hard disk logically connected to at least one server, a personal digital assistant with a network connection, and/or any other appliances with suitable processing, storage, and communications capabilities.

Referring again to Figure 47A, each customer appliance 2052 may have a VDE secure node installation 2054 incorporating a protected processing environment 154, as described in "Ginter et al", and messaging services software 2058 that manages communications with other appliances. (In an alternative example, some appliances may lack secure nodes or sufficiently secure nodes, and receive appropriate one or more protected processing environment 154 based services from one or more servers and/or peers.) These appliances may be located in the same physical and/or logical environment, such as on the same local area network, and/or may be distributed across wide area networks such as multi-location corporate Intranets and/or the Internet itself. Among other tasks, messaging services 2058

"listens" for messages destined for that particular appliance or for broadcast messages intended for at least one appliance in the set of appliances that receive the broadcast. In certain instances no appliance may actually be "listening." In other examples, the messaging services 2058 may incorporate delivery assurance capabilities that assure delivery through use of explicit or implicit acknowledgments of receipt combined with the ability to retransmit information that has not been acknowledged. Messaging services 2058 may be designed such that an operator may select from one or more delivery assurance levels, for example "no receipt acknowledgment," "retry n times, then notify operator if not received," "retry until a certain date/time, then notify operator if not received," "retry n times and/or until a certain date/time, no operator notification necessary," et cetera.

Messaging services 2058 may use the secure node 2054 to package one or more messages in a VDE secure container that may also include one or more sets of rules and usage consequences that may be associated with one or more messages in the container as described in "Ginter et al". In this example, messaging services 2058 then sends the secure container to one or more destinations using, for instance, TCP/IP and/or some other network protocol(s). Also, messaging services 2058 may broadcast a VDE container to one or more other customers 2053.

In this example, a customer 2053 uses application 2060 to persistently request or "subscribe" to one or more particular classes of content. For example, a highly detailed class might include "business information concerning the US market share of PC vendors, information in text format, costing less than a dollar per item, and for which the subscriber receives the right to excerpt at least one whole paragraph, provided that the excerpted amount constitutes less than 25% of the entire item based on word count." This same and/or another application may also be used to interact with instances of content in the desired class, for example, by displaying information on a computer screen and/or another output device in accordance with the rules and usage consequences associated with that item. If a user no longer has an interest in one or more classes, they may also use the same (or similar) application 2060 to "unsubscribe" from a particular subject, or specify further narrowing or broadening criteria to adjust the flow of content from one or more classes.

Items in the desired class or classes may be available from more than one content source 2074(1)-2074(n). To enhance the efficiency of locating content of interest to the subscriber or other participant, the matching and classification 900 may have created such a class definition and assigned one or more content items to that class. In one example, the matching and classification 900 may have sent one or more methods, and administrative events necessary to invoke the method(s), in a VDE secure container to one or more content sources 2074 where the classification methods are executed.

Such methods may, for example, assign content items to one or more classes. One or more object and/or item identifiers may have been transmitted to the matching and classification utility 900 along with class assignments for each item. If the matching and classification utility 900 has not previously created the desired class and assigned items to it, in response to a request from the subject switch 2051, the matching and classification utility 900 may do so using any appropriate combination of one or more such classification methods and procedures. The matching and classification utility 900 may may create at least one object class hierarchy, object class, object classification scheme, object category and/or object category scheme using at least some rights management information and assign at least one object, item, and/or subscriber to at least one category and/or class.

Subsequent to receipt of the request and/or "subscribe" message from the customer 2053, the subject switch 2051 may query the matching and classification 900 for content sources 2074 that have items in the desired class or classes. The matching and classification utility 900 may respond with information indicating known sources of information in the desired class(es), if any. The subject switch 2051 may then send a VDE container to the appropriate content source(s) 2074 indicating that certain customers 2053 are interested in items in the desired class and that the content source 2074 should send items in this class to this customer 2053

and/or groups of customers, and/or include such content in broadcasts which may be received by such subscribers.

The content sources 2074 may have already received class definitions and class assignment information from the matching and classification utility 900 and/or may have received from the matching and classification utility 900 or another party to the transaction one or more classification methods and associated events to invoke one or more of these methods to perform classification and/or class assignment processes.

In one arrangement, the content source 2074 may send the desired items directly to the subscribing customers 2053 by using the messaging services 2058 and subject switch 2051 to publish each item as it becomes available for distribution. In another example, the content source 2074 may broadcast the information such that subscribers' messaging services 2058 will have the opportunity to access the such items from a broadcast. The content source 2074 may call on messaging services 2058 to use the VDE secure node to package the item in a VDE container along with associated rules and usage consequences and then send that container such that one or more listening messaging services 2058 on other appliances 2052(1)-2052(n) will receive it. Based on subject information contained in the message header and/or in unencrypted (but optionally protected for integrity) areas of the VDE container, the listening messaging services 2058 may identify the message as belonging to a subject

class it is listening for, then use the VDE node to open the container and view or otherwise use the item in accordance with that item's associated rules and usage consequences.

In another arrangement, the subject switch 2051 may be located on each customer appliance 2052(1)-2052(n). Using messaging services 2058, each subject switch 2051 may communicate with the matching and classification utility 900 to locate sources of content matching the subscribed classes. In this example, the subject switch 2051 on the local appliance then uses the messaging services 2058 to communicate with one or more content sources 2074 indicating classes of content to which it wishes to subscribe. Using the messaging services 2058, one or more content sources 2074 may directly send and/or broadcast items in the desired classes to subscribing customers 2053 in VDE secure containers along with associated rules and consequences. In another arrangement, the content source 2074 may send one set of rules and usage consequences that apply to members of one or more item classes, thus potentially improving the efficiency of distribution and of rights management. In another example, the rules and content items may be sent in separate VDE containers. In this example, the messaging services 2058 and subject switch 2051 listen for messages that are addressed to those customers who subscribe to a particular content item class and makes those items available to customers using an application 2060.

In another arrangement, messaging services 2058 and/or subject switch 2051 may be installed and run on network routers, network switches, one non-limiting example being ATM switches, and other packet and/or cell switches.

Example: Digital Broadcasting Based On Matching And Classification

"Shear et al" discloses a Digital Broadcasting Network ("DBN") that may function as a cooperative of Web sites and, for example, service providers, with a central and perhaps regional and logical (e.g., market based) headquarters groups, or it may function as a for profit, shareholder corporation in a business model reminiscent of television broadcast companies (e.g., NBC), or it may function as a cooperative or virtual corporation that has some mix or combination of mixes of the above attributes and employ distributed peer to peer, hierarchical, and centralized administrative business relationships and activities.

In one example, plural corporations may join together to provide the advantages of size and coordination with individual participants providing some degree of specialty expertise and the body of entities coordinating together in some fashion in a "higher" level cooperative or corporation.

Figure 48 shows one non-limiting example 2100 of a DBN that includes one or more DBN Web servers 2104(1)-2104(n) and one or more Web users each with VDE nodes. Users are attracted to a

specific DBN server (or servers) because it provides access to specialized content and/or services 2108. Based at least in part on rights management information 2110 collected from DBN servers, for example, controls associated with the most frequently requested information, the matching and classification utility 900 creates categories of content (and/or services) and assigns DBN servers to one or more classes according to their specialization(s). The matching and classification utility 900 may may create at least one class hierarchy, class, classification scheme, category and/or category scheme using at least some rights management information and assign at least DBN server and/or at least some information to at least one category and/or class.

For example, one DBN server may specialize in consumer sports information while another may specialize in legal information. DBN servers may specialize in plural content (and/or service) areas. This class and class assignment information is provided to DBN servers, to content (and/or service) providers, or both.

The matching and classification utility 900 in one example sends VDE containers 2112 to content sources 2102 indicating specific classes of content that should be sent to one or more DBN servers 2104. Using this information, content providers 2102(1)-2012(n) then send content in these categories in VDE containers 2106 that match the categories of most frequently hit and/or consumed content on a DBN server 2104(1)-2104(n). (In another example,

other information may be used as the basis of classification, matching, and selection.) For instance, the matching and classification utility 900 sends a VDE container 2112(2) to content source 2102(1) with instructions to send content in categories 1,11, and 15 to DBN server 1 (2104(1)). This content may, in turn, be sent to one or more consumers in VDE containers 2108(1), 2108(3).

In one aspect, this example process is analogous to hard goods manufacturers and distributors keeping Wal-Mart shelves stocked with those items in greatest demand based on point of sales and inventory data. One difference, of course, is that in this example, the DBN server is stocked with intangibles in the same or similar class as the intangibles sold rather than providing replacements for hard goods that have been sold off the shelf. In another example, a DBN server may send its classification data to content providers along with a request that they send more of the same. The request may be sent independently of the class information.

In another example, the matching and classification utility 900 may receive content and/or rights management information from providers and go on to create classes of content and/or content providers in which the classes may be partly defined using rights management data. Content on one class may, among other things, be distinguished from content in another class by price, payment methods, usage opportunities (e.g., available for printing, available for viewing pay-per-use), usage consequences, and/or specific

permissions. The matching and classification utility 900 may subsequently send a communication, perhaps in a VDE container, to providers indicating that they send content in one or more specified classes to at least one DBN server.

Non-limiting example Figure 48 shows that the DBN 2100 may consist of video 2202 and/or audio 2203 content providers who send certain categories of video and/or audio content 2206 to DBN servers 2204(1)-2204(n) based on the categories of content each server may specialize in, which, in turn, may be determined at least in part on frequency of usage and/or other rights management information sent in VDE containers 2213 to the matching and classification utility 900, or to a usage clearinghouse 300 and then to a matching and classification utility 900. (In another example, other information may be used as the basis of classification, matching, and selection.) The matching and classification utility 900 sends VDE containers 2212 to content sources indicating that they should send content in specific categories 2206 to specific DBN servers 2204. In turn, each DBN server 2204(1)-2204(n) delivers video 2208 and/or audio 2209 in VDE containers to parties interested in such content. In another example, a VDE container may hold both video and audio and/or any other content type.

Example: Matching and Classification Utility 900 Can Also Support "Pull" Distribution Models Based On Classes

Notwithstanding the noted trend toward "push" content delivery models, the present inventions also enhance the efficiency, focus, specificity, and convenience of content "pull" models. In one example 2300 (Figure 49), the matching and classification utility 900 sends in VDE containers 2306(1)-2306(n) at least one administrative event and/or associated method that performs classification and/or class assignments to a VDE-aware appliance. The administrative events and method(s) are processed under the control of the VDE node. In one example, the results of processing the classification method may indicate at least one class of content and/or services of interest to a user and/or node. The classification method may also create at least one class hierarchy, class, classification scheme, category and/or category scheme using at least some rights management information and assign at least one service and/or at least some content to at least one category and/or class.

Subsequently, a VDE container 2308 may be sent to a provider 2302 with information indicating at least one class of content, services, transactions, rules and/or usage consequences, such as the ability to modify, excerpt and/or reformat, and/or events and a request that that the provider send content and/or pointers to services that meets the stated criteria and/or descriptive information about such content, services, transactions, and/or events to the requesting user

and/or node. The request may, for example, be initiated explicitly by the user and/or node or may be initiated by the node according to one or more administrative events and associated methods and/or control sets. In turn, the content provider 2302 sends a VDE container 2304 to the requesting user 2306(1) with content that matches the desired selection criteria and/or profile.

The user may elect to use, consume, purchase, and/or rent one or more content objects (or use one or more services). As this one example shows, the user pulls in content and/or interacts with services by matching at least one class indicating user preferences with at least one class of content objects and/or services and/or transaction types.

Example: The Enterprise Distributed Matching And Classification Utility

Businesses and other organizations may be concerned with privacy and confidentiality regarding information and/or services used within the company. This concern may be manifest regardless of whether the information and/or services originated inside and/or outside the organization. Thus some organizations may have strong incentives to take advantage of the present inventions by operating a distributed matching and classification utility 900 to provide matching and classification services within the enterprise while at the same time maintaining a higher degree of confidentiality and privacy by selecting and/or limiting the nature, range, and detail of information sent outside the organization.

Figure 50 shows an example 2400 of an entity 2406 that has one or more VDE enabled appliances and users 2420(1)-2420(5) on a corporate Intranet 2418. These appliances may be, for example, computers, workstations, mainframes, or more specialized devices, such as supercomputers and/or graphics workstations for animation and special effects. The company may also operate internally one or more Commerce Utility Systems, perhaps including a financial clearinghouse 200, a usage clearinghouse 300, and a matching and classification utility 900. The company may also operate at least one content server 2414. These commerce utility systems and servers are also connected to the company Intranet 2418. The company 2406 also maintains one or more connects to the Internet 2402. (In another example the company may maintain connections to at least one private network operated by themselves and/or another party in addition to, or instead of one or more connections to the public Internet.) The content server(s) may provide access to internal, proprietary company information and/or to external, often commercial information. The internal content server may act as a gateway to external providers 2404(A)-2404(C) and/or may host commercial content locally on a content server 2408.

In one example, VDE audit records and/or other rights management information are sent in VDE containers 2412 from one or more VDE nodes 2420 to the enterprise usage clearinghouse 300 which may forward at least some of this usage information in VDE containers 2410 to the enterprise matching and classification utility

900. The enterprise matching and classification utility 900 may also collect from internal information sources 2414 information in addition to audit and rights management information, such as information in a human resources, accounting, and/or budgeting database containing data about company employees. These data may indicate, in one example, titles and responsibilities within the company, budgets allocated for external information and/or services, authority to spend, and budget remaining. The budget and financial information may have come in part from the financial clearinghouse 200. The matching and classification utility 900 may also create at least one class hierarchy, class, classification scheme, category and/or category scheme using at least some rights management information and assign at least service and/or at least some content to at least one category and/or class.

In one example, using at least some VDE rights management data, for example, whether certain information can be viewed by anyone, by any employee, or only by employees in certain job classes, such as "manager," the enterprise matching and classification utility 900 creates one or more categories and assigns one or more employees and/or VDE nodes to one or more topic categories. These categories may, for example, indicate content and/or service topics, subjects, and/or content areas of potential interest to each employee and/or groups of employees sharing at least one attribute in common, for example, similar interests and/or responsibilities.

In turn, the enterprise matching and classification utility 900 sends to at least one external content and/or service provider 2404 on Internet 2402 one or more VDE containers 2424 with information that indicates categories of interest. The content providers 2404 may themselves be specialized; in one example, a content provider may specialize in general business and financial news while another may specialize in scientific, medical, and/or technical information. In another example, a single content and/or service provider may provide an extremely broad range of content and/or services.

The external provider may send at least one VDE container 2422(1) with content and/or rules and consequences and/or metadata about content and/or services to a content server internal to the enterprise. In another example, such VDE container(s) 2422(2) may be sent directly to an employee and/or one or more groups of employees. The information sent by the external provider is tailored to, or in some way responsive to the content and/or service categories requested by the enterprise matching and classification utility 900.

In another example, the enterprise matching and classification utility 900 itself may be a distributed commerce utility implemented on more than one computer and/or other appliance within the enterprise. These several matching and classification utility 900s may serve different geographic areas and/or may themselves specialize in particular content and/or service areas.

In another example, the enterprise matching and classification utility 900 send class and/or class assignment information to a matching and classification utility 900 in another organization that, in turn, may be part of a common value chain.

Example: Chain of Handling and Control Entails Class-based Rules and Usage Consequences

VDE-based value chain management or "chain of handling and control" disclosed in "Ginter et al" enables, amongst other things, plural parties to independently contribute rules and usage consequences under the authority and/or control of more senior or prior participants in the value or distribution chain. Class-based rules may play a role in the efficiency and effectiveness of creating, operating, and/or extending value chain processes.

Figure 51A shows an example 2500 of a publisher ABC 2502 using a VDE packaging application 2510 to put into a VDE secure container 2512 sets of rules and usage consequences that may vary according to class. In this non-limiting example, the class is "content type." The publisher may have rights in a wide variety of content and content types. Consequently, the publisher may create rules for text objects that may differ from rules for audio objects.

The publisher 2502 sends the class-based rules and usage consequences to a first creator 2504 who also has installed VDE on her or his appliance 2516 and who has also been given one or more certificates and/or other digital credentials by the publisher (and/or

trusted third party) indicating that he is indeed a creator authorized by the publisher 2502. The publisher has included rules that allow only authorized value chain participants to package content using publisher provided rules and/or to modify, enhance, extent, and/or change some or all of the publisher's rules.

The first creator 2504 then uses a VDE packaging application 2510 to package an image he has created in a VDE container 2514 according to the rules provided by the publisher and with the addition of the creator's own rules. In one example, the first creator contributes rules that implement a one-time 50 cent charge to the consumer for opening and viewing the creator's image. The creator may also contribute rules reflecting his wish to receive audit records with information concerning the consumer and/or context in which the image was used. These creator rules and usage consequences are contributed generally independently of the rules and usage consequences contributed by the publisher. Note that the VDE container 2514 now holds at least the publisher's 2502 rules for each object class, the first creator's image and his associated rules and usage consequences.

A second creator 2506 receives the VDE container from the first creator and using a VDE packaging application 2516 adds a text file to the container 2520 along with her rules and usage consequences. As before, she also has a certificate and/or other digital credential(s) identifying her as authorized by publisher ABC to

add and/or modify content and rules and usage consequences. As in the case of the first creator 2504, she adds her text and rules and usage consequences generally independently of controls contributed by prior participants. She may, in one example, prevent printing of the text and charge $1.00 the first time a consumer opens and views the text.

The VDE container 2508 now holds text and rules and usage consequences contributed by creator 2 (2506), an image and rules and usage consequences contributed by creator 1 (2504), and the class based rules (and perhaps other rules as well) contributed by example publisher ABC 2502.

Creator 2 (2506 sends the VDE container 2508 to publisher ABC 2502 who then sends the container 2522 directly and/or indirectly to consumers. When the consumer uses the content, the rules and usage consequences of all three value chain participants (and of other possible participants as well, distributors and repackagers, for example) are applied.

Example 2600, Figure 51B shows that the publisher 2602 may have sent a VDE container 2612 with various rules and usage consequences to a matching and classification authority 900 who may classify the rules and send the rules and their class assignments to a rights and permissions clearinghouse 400. The matching and classification utility 900 may also create at least one class hierarchy, class, classification scheme, category and/or category scheme using at

least some rights management information and assign at least one rule to at least one category and/or class.

An authorized first creator 2604 may send a VDE container 2617 to the rights and permissions clearinghouse 400 asking for rules in the class "rules for authorized creators, for image objects, from publisher ABC." The rights and permissions clearinghouse 400 returns a VDE container 2614 with rules in the requested class. The first creator 2604 uses a packaging application 2616 to package his image using these rules plus rules and usage consequences reflecting his rights and wishes and sends the VDE container 2614 to the second creator 2606.

The second creator 2606 also sends a VDE container 2619 to the rights and permissions clearinghouse 400 asking for rules and consequences in the class "rules for authorized creators, for text objects, from publisher ABC." The rights and permissions clearinghouse 400 returns a VDE container 2621 with rules and consequences in the desired class. The second creator 2606 uses a packaging application 2618 that determines that she is a creator authorized by publisher ABC 2602 and goes ahead and adds her text object and her rules and consequences to the VDE container 2608, which is then sent to the publisher ABC 2602 for further augmentation, vending, and/or distribution to other value chain participants.

Example: Secure Directory Services May Provide Class And Class Assignment Information

Whole industries have arisen to target communications to individuals, organizations, groups, and/or other classes sharing at least one common attribute, and/or to provide directories from which others can locate individuals, organizations, groups, and/or other classes. Examples of these industries include direct marketing, advertising, yellow and white pages directories, directories of directories, and various electronic and paper membership lists and professional directories.

In addition to identifying information such as names, e-mail addresses, physical mailing addresses, phone numbers, fax numbers, and/or similar attributes, the secure directory services 600 may also provide information about class membership(s) for individuals, devices, services, groups, and/or organizations. The non-limiting example 2700 shown in Figure 52 includes a secure directory service 600 that has received class and class assignment information for one or more individuals 2716(1)-2716(n). The class assignment information is shown in the bottom four rows of the directory record 2718(1) for one individual.

In this example, a content provider 2702 sends a VDE container 2704 to a secure directory services 600 asking whether the service can provide a list of individuals in class "AF." The requested class could be any class defined by one or more attributes and may be based on usage profiles that include rights management information,

non-exhaustive examples of which include price, payment methods accepted, permitted operations, meters, and privacy controls.

The secure directory services 600 returns to the content provider in a VDE container 2706 an indication that there are presently 57 individuals known to that service in class "AF." In turn, the content provider 2702 sends a VDE container 2708 with at least one piece of content and/or rules and usage consequences back to the secure directory services 600 along with instructions requesting that the secure directory services 600 forward the content and/or control sets to each of the 57 members of class "AF" who might be interested in this piece of content. The secure directory services 600, in turn, forwards the content and/or controls (in VDE containers 2714(1)-2714(n)) to members of class "AF," who may elect to interact with the content in accordance with their associated rules and consequences.

In another example, the secure directory service 600 may send identifying information 2710 directly to the content provider 2702 who may then send content 2712 in one or more classes directly to one or more members 2716(1)-2716(n) of the class. The secure directory services 600 may, for example, include permissions for the class information that have expiration dates and/or limits on the number of times the information can be used.

Example: Matching And Classification Utility 900 Supports Class-Based Micro-Merchandising And Micro-Segmented Sales Processes

The present inventions may be used in support of services as well as content distribution based business. Example 2800 (Figure 53) shows a travel company 2801 sending a VDE container 2810 to a matching and classification utility 900 requesting information on those individuals who may be interested in certain combinations of leisure-time activities. These classes might have been defined at least in part on the basis of usage and other rights management information 2816, for example, the kind of leisure-time information recently looked at, for how long, and/or its cost, and/or the kind of Web sites recently frequented, sent from consumer VDE nodes 2802(1)-2802(n) to the matching and classification utility 900, and/or to a usage clearinghouse 300 who, in turn, sends at least some of the usage information (or a summary form of such information) to the matching and classification authority 900. Classes may also be defined using information gathered directly from the consumer 2818, perhaps under the control of VDE. The matching and classification utility 900 may also create at least one class hierarchy, class, classification scheme, category and/or category scheme using at least some rights management information and assign at least one consumer, service, and/or at least some information to at least one category and/or class.

Example Figure 53 shows that a consumer 2802(1) has recently indicated a preference and/or interest in skiing, music, and flying to

Colorado. Another consumer 2802(n) has indicated a preference for and/or interest in surfing Hawaii. These preferences may be determined at least in part on the basis of rights management information. In response queries sent in one or more VDE containers 2810 from the travel company asking for interest and preference information, the matching and classification utility 900 returns one or more VDE containers 2812 with identifying and class information. The travel company may send information about already existing vacation packages and/or packages specially created to meet the specific interests of one or more individuals, for example, information about skiing in Colorado, and rock concerts 2604 to consumer 2802(1) and information 2614 about surfing Hawaii to consumer 2802(n). The recipients may send VDE containers 2806 to the travel company 2801 indicating agreement to buy the package offered or may request additional information or may negotiate terms and conditions such as price, departure date, insurance, and the like. These negotiations may be conducted using the inventions described in "Ginter et al", Figures 75A-76B using VDE negotiations.

Both services and/or hard goods may be offered to particular persons, nodes, groups, and/or entities based on the class membership of the potential purchaser and the class membership of the goods and/or services to be purchased. Thus in another example, the travel company could have included the purchase and/or rental of the skis or of the surf board.

Example: Matching And Classification Utility 900 Supports Trading in Hard Goods

Business to business trading in goods and/or services may be substantially facilitated through services provided by the matching and classification utility 900. Information on certain classes of goods and services may be delivered to certain people, groups, or entities based on the class membership of the recipient. In one example, these various class memberships may be determined using control set and audit information regarding trading preferences and/or transaction patterns. In another example class membership may be determined by actions and/or information provided by at least one value chain participant.

Example 2900 (Figure 54) shows a buyer A 2904 sending a VDE container 2908 to a trading company 2902 with a request asking if trading company will sell company A one or more desired items. Trading company 2902 may then send a VDE container 2910 to a matching and classification utility 900 with a query asking who can supply the desired items under terms and conditions that are also included in the container. Since these terms and conditions may be the subject of negotiations, they may be in a format conducive to VDE-based negotiations as described in "Ginter et al" Figures 75A-76B.

The matching and classification utility 900 may send inquiries 2910 to one or more suppliers 2906(A)-2906(N) and/or may have already received information and/or associated control sets from

suppliers in VDE containers 2912. Based on the request from trading company 2902 and supplier 2906 information obtained 2912, the matching and classification authority 900 returns a VDE container 2916 indicating that in this one example, suppliers A 2906(A) and Z 2906 (N) can provide goods in the class(es) defined by trading company's 2902 request(s) 2910. In turn, trading company 2902 sends at least one VDE container 2918 to buyer A 2904 indicating that they will sell buyer A the previously requested items under the enclosed terms and conditions. In another example, there may be some VDE-based (see "Ginter et al", Figures 75A-76B) negotiations between the various parties in this value chain, including between trading company 2902 and buyer A 2904.

In another example, buyer A 2904 may consult the matching and classification authority 900 directly and may then purchase directly from one or more suppliers 2906.

Example: Matching And Classification Utility 900 Supports Securities Trading/Brokering

In addition to hard goods, the matching and classification authority 900 may also support securities trading. Example 3000, Figure 55, shows the matching and classification authority 900 sending to a VDE-aware appliance with one or more stock trading related applications 3004 a VDE container 3010 with an administrative event and method (as described in "Ginter et al") for classifying equities related information, including, as non-limiting examples, current and historical price, volume, and index

information, financial performance data for publicly held companies, forecasts, risk management information, options and futures, and the like. The classification method may also utilize rights and permissions, including access control information, permitted operations, and/or expiration times and/or dates for rights management information. The classification method may also create at least one class hierarchy, class, classification scheme, category and/or category scheme using at least some rights management information and assign at least one element to at least one category and/or class.

In turn, using the VDE aware appliance 3004, the stock trader 3006 sends a smart object 3012 to at least one information source 3002 asking for information in at least one class identified by the classification method. In one example, the class may be information concerning "publicly traded companies with annual revenue greater than $500M in the healthcare sector in which the CEO has been in place less than 5 and greater than 1 year and with access restricted to customers (rather than available to anyone) with access and use expiring in 90 days." The information provider(s) 3002 returns a VDE container 3014 with information meeting and/or more closely meeting the stated class criteria. Based upon this and other information, the trader 3006 may go ahead and enter an order for at least one trade in at least one stock 3008. In another example, the trader may create or obtain methods that trade automatically in certain classes of securities.

Example: Matching And Classification Utility 900 Supports Trading in Currency and Debt Instruments

Among the classes of great value to traders are the classes of items whose trading maximize profits and/or minimize losses. Example 3100, Figure 56, shows a trader in currency and/or debt instruments 3102 sending a VDE container with market and other financial and economic information and VDE control set information 3108 to a matching and classification authority 900 with a query 3114 asking the matching and classification authority 900 to identify the class of currency trades and/or debt instrument trades that maximizes profit and/or minimizes losses. The matching and classification authority 900 applies one or more methods to the data and returns at least one class definition 3112, the assignment of possible trades to that class 3110, and relevant control set information, such as controls indicating who may see the information, and those that prevent unauthorized modification of the information. The matching and classification authority 900 may also return methods for executing the trade. The matching and classification utility 900 may also create at least one class hierarchy, class, classification scheme, category and/or category scheme using at least some rights management information and assign at least some trading information to at least one category and/or class.

The example trader 3102 examines the recommendation and sends VDE containers 3118 (A, B) with trade methods and control sets to a foreign exchange market 3104 and/or to a debt instrument

market 3106 where the trades are consummated. The markets send back VDE containers 3116(A, B) with audit information indicating the results of the trading order. In another example, the matching and classification authority 900 may be instructed to send trading orders directly to the market(s) for execution. In another example the trader may send a VDE container to at least one source of relevant information asking that source to send certain information to the matching and classification authority 900. In another example, having established the desired trade(s) using the matching and classification authority 900, the trader may place the trade by phone and/or computer and/or other communications device without using VDE.

Example: Matching And Classification Utility 900 Supports Consumers Locating Services That Are Members Of A Specified Class

The services of the matching and classification authority 900 may also benefit consumers by locating certain classes of services. Example 3200, Figure 57, shows a consumer sending a VDE container 3206 to a matching and classification authority 900 asking, "which banks are in class A?," where class A are "those banks that offer the highest savings interest, no ATM fees, online/Web banking using VDE, insured accounts, free checking with balances larger than $2,500, "image" statements (where check images rather than the actual checks are returned), and complete privacy protection (except

where legally required to disclose) for VDE based banking transactions.

The example matching and classification authority 900 sends a query in a VDE container 3208 to one (or more) information sources 3202 and receives one or more VDE containers 3210 with the requested information. The matching and classification authority 900 then determines which bank or banks meet the stated criteria of the consumer 3204 and then sends a VDE container 3212 with the answer to the consumer, in this example, banks A, B, and C. The consumer 3204 may then go ahead and execute a financial transaction, for example, transferring funds from one bank to a bank identified by the matching and classification utility 900 as offering higher interest rates, while being assured of maximal privacy for this (and perhaps other) transactions.

In another example, after determining which banks are in the desired class, the matching and classification authority 900 may send a VDE container to one or more banks saying that the consumer wishes to know about their services and requesting the bank to contact the consumer directly. The bank may send controls ensuring the privacy of future interactions with the customer. For example, controls that apply to audit records such that only the bank and the consumer will have permission to access these records.

Example: Matching And Classification Authority 900 Supports Class-Based Software Distribution

VDE and the inventions disclosed in "Ginter et al" at last provide a way of ensuring that the efforts expended on creating software will be rewarded since the software can now be persistently protected, usage information can be collected, and payment ensured. These inventions also support micropayments and microtransactions, thus creating a world in which the price of software objects—any kind of objects actually—may become very small. Pay per use, rental, rent to own, and other pay as you go pricing models together with VDE may create a new explosion of creativity in software design and creation, since use prices will be low and providers can be assured of receiving payment.

The present inventions provide opportunities for software providers to more efficiently market their wares. Example 3300, Figure 58, shows a number of users with VDE installed on their appliances 3304(A-F). These people are using software (and other content). VDE meters usage of various objects and sends audit records in VDE containers 3306 (A-F) to a usage clearinghouse 300, which then sends audit records 3308 to the matching and classification authority 900. A software distributor 3302 sends a VDE container 3310 to the matching and classification authority 900 with a query asking who is in the class, "buys Java applets, with pay per use pricing, and for which the cost per use is between $.0001 and $.001?"

The matching and classification authority 900 returns a VDE container 3312 with a list of names and (network) addresses of those matching, or most nearly matching the desired characteristic(s). The software distributor 3302 then sends at least one VDE container 3314 with at least one software object, and/or a pointer to a software object, in this case a Java applet, and perhaps other relevant information, such as VDE control sets and/or various metadata describing some aspect of the object, for example, what it does, what it costs, etc. The user may then elect to use the object or not. In another example, instead of individuals or VDE nodes, the users might be groups of nodes, users, organizations, parts of an organization, and others that can be identified as belonging to at least one class. In this case, the software may be offered to some or all members of class, group and/or organization.

Example: Matching & Classification Utilities Provide Services To Authenticated Classes of Nodes, Users, Content Services and/or Transaction Services

Among the ways in VDE nodes, users, content services, and/or transaction services can be authenticated is through the use of certificates and/or other digital credentials issued by an appropriate trusted third party, a certifying authority 500, for instance, that warrants and/or attests to some fact or facts, which may include membership in one or more classes, including the identity class. Figure 59 shows a non-limiting example 3400 in which a number of matching and classification authority 900(1-N)s, each of which may

provide its services to different classes, where class membership is authenticated using certificates and/or other digital credentials. In other examples, additional authentication mechanisms may be used in combination with, or instead of certificates, such as information known only to the user, VDE node, and/or appliance, including passwords, cryptographic keys, information stored in hardware, and/or software.

In example 3400, Figure 59, commerce participants including, the matching and classification authority 900, may make rules and consequences conditional on class definitions and/or the assignment of members to a class. Class membership may be authenticated by a certificate and/or other digital credential issued by one or more commerce participants in addition to, and/or instead of a trusted third party such as a certifying authority 500. For example, a certificate and/or other digital credential may attest to user identity, that is, that a user is the user he or she claims to be. Nodes, devices, networks, servers, clients, and services, are other non-limiting examples of other commerce elements that may be authenticated with certificates and/or other digital credentials. Any commerce participant may issue a certificate, but other participants are not required to accept a given certificate as an authenticator.

Figure 59 shows multiple matching and classification authorities 900(1)-900(N), each of which may provide services to members of a particular class, in these non-limiting examples, to

nodes in a particular deployment (matching and classification authority 900(1)), in a particular vertical segment and/or institution of society, such as Higher Education (matching and classification authority 900(2)), one or more value chains, such as business information content providers (matching and classification authority 900(3)), and/or a particular transaction and/or service arena, such as hard goods trading (matching and classification authority 900(n)). Other commerce utility systems, a certifying authority 500 shown in Figure 59, for instance, may also provide services to a class. In each of these instances, the services of the matching and classification authority 900 may depend upon finding certain authenticating certificate(s) and/or other digital credentials on the appropriate VDE nodes.

For example, matching and classification utility 900(1) provides services to nodes 3410(1-n) in the deployment 3402 administered by VDE administrator 800. Each node may have a certificate 3412 issued by certifying authority 500(1) that provides services to this deployment.

In another example, certifying authority 500(2) provides certificates and/or other digital credentials to participants in a higher education value chain 3404 consisting of an arbitrary number of colleges and universities 3416(1)-3416(n), providers 3418(1) and students 3418(n), and a matching and classification utility 900(2) that provides classification, matching, and selection services to higher

education 3404. In one example, the matching and classification utility 900(2) only provides services to value chain participants who have a certificate 3420 issued by certifying authority 500(2).

Matching and classification utility 900(3) services can be provided only to members of one or more classes based on certificates issued by a certifying authority 500(3). In one example, the class is participants in a business information value chain 3406, comprising an arbitrary number of content providers 3424(1)-3424(n), an arbitrary number of users and/or consumers of business information 3422(1)-3422(n), and a certifying authority 500(3) that issues certificates and/or other digital credentials to members of the value chain 3406.

In addition to membership in certain deployment, institutional, and/or content usage classes, the matching and classification authority 900(4) may provide services to members of a certain transactional value chain, in one example, traditional transactions 3408. In this example, a certifying authority 500(4) issues certificates 3432 to one or more companies 3428(1)-3428(n) and one or more trading companies 3430(1)-3430(n). In another example, other participants may receive certificates and/or other digital credentials, including banks and financial institutions, government authorities, for example, tax and/or customs authorities, consumers, suppliers, and/or transportation companies. The matching and classification utility 900(4) provides services only to those entities and/or individuals in

possession of the appropriate certificate 3432 indicating that the holder of the certificate is an authenticated participant in one or another trading value chains.

In other examples, a commerce utility system may provide services to more than one class where class membership is indicated by at least one certificate and/or other digital credential issued by a certifying authority 500 and/or value chain participant. In one example, matching and classification authority 900 might provide services to the class "Higher Education" and to the class "K-12 Education."

Possession of a certificate and/or other digital credential may be among the information used to classify a node, user, appliance, device, entity, and/or other commerce participant, and rules and consequences can be made conditional on membership in one or more authenticated classes and/or on the degree of confidence the rule provider has in the trustedness of the certificate and/or other digital credential issuer. In one example, a discount to higher education may be larger if the root for chain of trust for a given certificate is a well-known, highly respected and trusted third party, such as an authoritative accrediting organization, and smaller if the root belongs to the MIS department of a small college. In this example, the provider is willing to grant a higher discount when there is higher certainty that the recipient is in fact a member of a specific class or classes.

Example: Matching And Classification Authority 900 Supports Control Sets Based In Part On Employee Classes, Content Classes, And/Or Certificates And/or Other Digital Credentials

Chain of handling and control enables, amongst other things, multiple organizations to work together in secure, trusted, efficient, cooperative commerce processes. One way in which the present inventions extend these ideas is through control sets with rules and usage consequences that may be based in part on classes and the assignment of persons, entities, devices, content, services, or other process elements to classes of one kind or another by the matching and classification authority 900.

One example technique to classify employees is at least in part according to their roles and responsibilities within an organization. The matching and classification utility 900 supports classification, matching, creation and/or modification of VDE control set(s) based at least in part the class assignment of individual and/or groups of employees. In part by virtue of their employee classification, at least one employee may receive certain rights management information, for example, permission to access certain classes of information or permission to perform one or more permitted operations, transactions and/or events.

Example 3500, Figures 60A-60C shows a nurse 3504(1), physician 3504(2) , and billing clerk 3504(3) all work directly for an example hospital. The present inventions are in no way limited to

hospitals, but apply to any organization, group, entity, and/or institution with at least some defined roles and responsibilities and/or other class definitions that apply to employees, members, and/or others associated, affiliated, and/or employed by the organization, group, entity and/or institution. Rights management information may be part of the claim definition, for example, permissions to view, modify, excerpt, and so on.

Control sets may provide permissions conditional on employee class, for example, certain classes of employees may modify certain information and/or classes of information in a database while others may not. Class membership may be indicated by digital credentials, non-limiting examples of which include digital certificates and digital membership cards. Controls may be conditional on other information as well, for example, some computers and/or display devices may not show certain classes of data or updates to certain data elements may not be performed from certain computers or display devices.

Another example role is a representative 3504(4) of an insurance company 3508, who may have access to certain classes of hospital information by virtue of her or his class membership(s), some of which may derive from her or his role in the insurance company 3508 and/or from the insurance company's relationship with the hospital and/or with some of the hospital's patients and/or staff. The present inventions are not limited in application to an insurance company, but may be applied to any individual, group, organization,

entity, and/or institution with whom the example hospital and/or other entity has some form of relationship.

An example insurance company 3508 have received a certificate in a VDE container 3534 issued by certifying authority 500(1) attesting to the identity of the insurance company. In another example, this certificate and/or one or more additional certificates may attest to the fact that the insurance company has the appropriate charter, licenses, and other grants of authority to be in the health insurance business. The certifying authority 500(1) may also send a certificate in a VDE container 3532 attesting to hospital's identity. In another example, this certificate and/or one or more additional certificates may attest to the fact that the hospital has the appropriate charter, licenses, and other grants of authority to provide hospital and related services.

The insurance company 3508 may have sent one or more control sets to the hospital in a VDE container 3542. These controls may be based in part on one or more certificates 3530 and/or on the classification output of an example matching and classification utility 900(2) operating within and/or on behalf of the insurance company 3508. The controls in container 3542 may indicate which individuals are actually employees of the insurance company, employee membership in one or more classes, permissions associated with that individual and/or class, and/or permissions associated with specific devices, communications channels (devices, ports, etc.), and/or

processes. In this one example, the hospital matching and classification utility 900(1) may create controls using the same and/or additional classes and controls received from the insurance company 3508.

The insurance company 3508 may also provide one or more certificates to the hospital attesting to the fact that one or more information sources within the insurance company are to be take by the hospital as trusted sources. Lastly, in this regard, the insurance company may issue one or more certificates on behalf of each employee attesting that each is in fact an employee of the company and may have certain authorizations.

In example 3500, Figures 60A-60C, a matching and classification utility 900(1) has identified various classes of hospital employees using information from at least one hospital information system 3502 and/or VDE node. The matching and classification utility 900(1) may also make use of certificates issued by a certifying authority 500(1)outside (a trusted third party) and/or a certifying authority 500(2) inside the hospital. Using data dictionaries 3522, patient records 3520, various employee information 3524, automated procedures, and/or other means, the matching and classification utility 900(1) creates classes 3526 of patient record information and associates one or more control sets 3528 with each class of information and/or with a patient record as a whole. These control sets may specify who has permission to use and/or modify the record

and/or an element(s) of the record that has been assigned to one or more classes on which the control set(s) may in part depend. In one example, the class based controls 3528 may be combined with other hospital and/or other party controls, controls from the insurance company 3508, to create new controls 3510(1)-3510(n) associated with patient records 3512(1)-3512(n).

The example nurse 3504(1) and physician 3504(2), for example, may be able to view, modify, print, and/or copy patient's name, address, and other similar descriptive information, next of kin, insurance, and medical information in accordance with controls 3510(1) and 3510(2), respectively . In another example, some members of the class "nurse" and/or the class "physician" may have different permissions by virtue of membership in one or more additional classes. A physician who is in the class "hospital administration" may have different permissions, for example, to billing records.

A billing clerk 3504(3) in the hospital may not have permission in control set 3510(3) to view medical information and/or next of kin, and in this example may be restricted to name and other patient descriptive information, insurance information, and billing information from the patient record. A representative 3504(n)of the insurance company may have permission by virtue of control set 3510(n) to view, but no permission to modify, print, or copy patient record 3512(n). In each of these examples, the VDE control sets are

at least partially conditional on the presence and/or absence of certain certificates indicating membership in one or more classes.

The present inventions may be applied to any information, person, group, device, network, service, database that pertains to any commerce activity whatsoever, and regardless of whether the parties to the commerce activity are individuals, groups, entities, organizations, institutions, nations, and/or societies.

Example: Matching And Classification Authority 900 Supports Classes And Matching Based In Part On Workflow And Work Process Automation

Not only do the present inventions enhance commerce processes that principally entail information, but the present inventions enhance workflow and work process automation as well. Example 3600, Figure 61, shows PCs 3608(a-c) functioning as station controllers connected to various manufacturing devices 3610 (a-c). These station controllers that exchange data and instructions with the equipment they control and/or manage. The station controllers are VDE-enabled. In another example, the manufacturing equipment may also have VDE nodes installed.

An example work in progress (WIP) and/or manufacturing control application 3606 keeps track of the overall manufacturing processes and exchanges information with other applications not shown, such as materials management, materials ordering, order

databases, logistics, inventory, accounts payable, accounts receivable, general ledger, human resources, time cards, and the like.

An example employee 3602 of the company sends a query 3612 in a VDE container 3604 to an enterprise matching and classification utility 900 within the company asking, "which VDE-controlled equipment will be available 3rd shift today, for 2 hours, capable of performing operations xyz with a nominal error rate of less than .0001 per cent?" The enterprise matching and classification utility 900 may request data 3616 from the WIP/manufacturing process control application 3606 and/or may already have access to the required data, indicating equipment availability, security level, capabilities, and statistical error rates. The WIP/manufacturing process control application 3606 may return a VDE container 3618 with the requested information. Based upon the query and available information, the matching and classification utility 900 responds by sending a VDE container 3620 to the employee 3602 with the answer, "equipment B and equipment C." In turn, the employee 3602 sends another VDE container 3622 to the WIP/manufacturing process control application 3606 with VDE a control set(s) indicating B and C should be scheduled for 2 hours on 3rd shift to do xyz operations. As part of this particular chain of handling and control, the WIP/manufacturing process control application 3606 sends VDE container 3624 to the VDE-enabled station controllers for equipment B or C with control sets that schedule work and specify the manufacturing processes and/or "recipes" for those specific

equipment 3610(b) or 3610(c). In turn, the respective station controllers carry out their instructions and report progress and completion in VDE containers 3626 sent back to the WIP/manufacturing process control application 3606, which may in one example, provide results to other applications and/or to the employee who originally requested the work to be scheduled and performed.

Example: Matching And Classification Authority 900 Supports Classes And Matching Based In Part On Government/Societal Commerce Administration

Among the rightsholders in commerce processes of all kinds are societies and governments. Governments may foster rules indicating that certain classes of individuals may have not have access to certain classes of content. Some classes of information may be treated as members of classes that define permissions, such as "confidential," "secret," "top secret," and so on. Other non-limiting example governmental rights may address permissions for import, use, and/or export of certain classes of hard goods, services, currency and financial instruments, and content. Travelers entering the United States, for example, are usually asked about currency (and currency equivalents) being brought into the country by the traveler. Children, for example, may be prohibited as a matter of law by governments from viewing content in the class "sexually explicit."

Another example of government rights is that different tax rules may be applied to different classes of electronic commerce

transactions using VDE. Example 3700, Figure 62A-62B, shows a certifying authority 500 operated by and/or on behalf of a government issuing a certificate and/or other digital credential indicating jurisdiction, namely, country. The certificate is sent in a VDE container 3710(a) to a VDE administrator 800. The government certifying authority 500 also sends certificates in VDE containers 3710(b)-3710(n) to the government matching and classification authority 900 attesting to the "country," in one example, the United States, and another certificate 3716 attesting to the fact that the matching and classification authority 900 is indeed an authorized service of the United States government.

In one example, the government matching and classification authority 900 has created tax class definitions 3712 and tax control sets 3714 that apply those definitions in various classes of circumstances, including the presence of certain control-related information, such as an appropriate country certificate from an authorized issuer of such jurisdictional certificates. The tax class definitions 3712, tax control sets 3714, and government authority certificates 3716' are sent in at least one VDE container to a rights and permissions clearinghouse 400, who, in one example, redistributes the tax class definitions 3712(1), tax class control sets 3714(1), and/or government authorization certificate 3716(1) to content providers 3702, service providers 3704, and other value chain participants. The certifying authority 500 also sends country certificates to one or more VDE administrators 800 who, in turn, send country certificates 3710'

to VDE nodes 3706(A)-3706(n) in their deployment. When content provider 3702 distributes content of any kind, the appropriate tax control sets 3714(A) are also included in the VDE container. A tax control set is applied whenever content is used in accordance with a tax class and provided that the appropriate jurisdictional certificate 3710' is present on the VDE node 3706(a). For instance, a VDE node may have a tax control set to be applied to sales of a class of content, specifically, to the class of "software." Whenever a software vend occurs, the appropriate tax is applied according to these rules.

In another example, the various country and government authority certificates may be sent directly from the certifying authority 500 to one or more VDE nodes 3706. The VDE controls that implement tax policy for one or more classes may also be sent directly to VDE nodes 3706 and/or to VDE administrators 800.

Example: Classification May Be Used In Automatically Selecting The Proper Display Context Based On Classes Of Information

Content objects may be displayed using one or another formats according to class membership of that object. In example 3800, shown in Figure 63A, a matching and classification utility 900 provides content class information 3810 to information providers 3802. A consumer 3807(1) previously has sent a VDE container to a provider of sports information 3802(1) indicating interest in "class b" stories, and perhaps other classes as well. The sports information provider 3802(1) sends back a VDE container 3808(1) with one or

more stories in "class b," perhaps "all stories about baseball, New York, Yankees, history, heroes with permission to print" an example of which is 3814(1), along with, in this example, one or more VDE control sets. The VDE container 3808(1) is received by a customer 3807(1) who then displays the content 3814(1) using one or another page formatting technologies based on macros, scripts, administrative events, methods, and/or other techniques. Also included in the VDE container is an image 3812(1) that was selected by the information provider as especially appropriate to the class of story being sent. In this example, perhaps the image 3812(1) is a faint image of Joe DiMaggio. This image also meets the criteria of "permission to print."

Example 3800, Figure 63A, also shows another instance in which a different consumer 3807(n) previously has informed a nature information provider 3802(n) of interest in class A stories. Here the information provider sends a VDE container 3808(n) that holds a class of stories different from the class of interest in the previous example. This VDE container 3808 holds a "class A" story, an example of which is 3814(n), that is displayed with a different image 3812(n), one that is appropriate to the story class, in this case, an image of a dog.

The class assigned to each story may be carried in the container as metadata for one or more story objects in another example. An example Web browser may request of the information provider an

image appropriate to that class, which if available, would be sent in another VDE container.

Class may affect display rules in other example ways as well. For instance, several team sports news stories may be displayed in a Web browser window in which a scene from a football or basketball game is faintly discernible in the background. Which image is displayed may be determined by the user's preferences given the classes of stories being presented on the page. The user, may have looked most at stories about the New England Patriots and a Patriots-related image may be displayed as background even stories about teams in addition to (or even instead of) the Patriots were being displayed.

In (another) example 3850, shown in Figure 63B, a matching and classification utility 900 provides class information to a provider 3852(1). Previously, one user 3857(1) has indicated to the provider 3852(1) that she prefers information in topic class A more than information in topic class C and information that costs less than $.50 per article while the other user 3857(n) has the opposite preferences and is not price sensitive. A matching and classification utility 900 may provide classification information, class assignments for objects, administrative events, and/or methods for these and related purposes. Regardless, the information provider 3852(1) sends the identical VDE container 3858 to each of the users 3857. However, their browser and page formatting software 3856 produces different pages in

accordance with each user's topic class preferences. In the example first case, the user 3857(1) sees three columns of topic A and one column of topic C while the second example user 3857(n) sees three columns of topic C and one column of topic A. As this example illustrates, the class preferences of users may affect the way in which the user interacts with content in various classes.

In another example, the matching and classification utility 900 may have sent one or more administrative events and/or methods 3859 to at least one user 3857 where the method performs the topic classification on documents and/or establishes topic classes and/or topic classes of greatest interest to the user.

Example: Information May Be Classified With Respect To Difficulty -- And This May Pre-Determine An Appropriate Interface

The class of content and/or the class of user may determine at least one display characteristic. One interesting example way of classifying content is with respect to its difficulty. One example measure of difficulty is reading level, which may reflect such aspects as vocabulary and/or complexity. It is well known that children (and adults) of the same approximate age read at different levels. In the example 3900, shown in Figure 64, a provider sends a VDE container 3902(1) with text at a 4th grade reading level and controls indicating that when used by a person reading at that level, the charge is 50 cents. However, if a person reads at less than the 4th grade level, the

charge is only 40 cents. "Reading level" may be indicated by a certificate and/or other digital credential.

A matching and classification utility 900 may send administrative events and/or classification methods 3910 to information providers, one or more other value chain participants, or to the students appliances directly. These methods may, for example, classify documents according to the degree of difficulty and create or modify controls for the whole document and/or subparts of the document, controls that may indicate the different prices for users at different reading levels. The matching and classification utility 900 may also send administrative events and methods to users that know how to make the document appear in the example browser at a lower reading level.

The example VDE container 3902(1) is sent from the provider to a child 3906(1) in the 4th grade who is reading that at that level. When the child opens the container to view (or otherwise use) the text, she or he is charged 40 cents (which might be paid by a third party such as a school and/or parent. The child sees the text as written 3904(1)

Example 3900, Figure 64, also shows the exact same document being read by a student 3906(3) in the class of 2nd grade readers. Now the browser displays the document 3904(3) modified by methods that may make the syntax less complex and may substitute simpler words and/or phrases for harder ones. A similar example

document and controls in a VDE container 3902(n) involving a 12th 3906(2)and 9th grader 3906(n) is also shown.

In other examples, the prices may be higher when users are reading text below their capabilities, they may be offered discounts for reading at a higher level, and/or they may be charged more for reading on different levels since modifying the text is a value added process, and providers of that value may wish to be compensated for their efforts.

Example: Classification May Describe Degree Of Focus Of The Content Unit Or Portion On A Topic, Or Characteristics Related To Conventional Formatting, Such As File Type

Sometimes the most interesting and/or useful content is at the intersection of various topics. Also, user often want content in a form or format that will be most useful, and most practical, to them. In the example 4000, shown in Figure 65, a matching and classification utility 900 receives from user 4002 a VDE container 4004 holding a request for documents in the class, "on economics and politics, costing less than $5.00, and in MS Word format." The matching and classification utility 900 responds in this example by providing in a VDE container 4006 at least one Uniform Resource Locator (URL) that points to the location of the document(s) on the World Wide Web.

The user 4002 in this example sends a message in a VDE container 4008 asking for the document identified in the URL. A

provider sends back a VDE container 4012 with the desired document 4010 that has been classified by the matching and classification utility 900. In this example, parameter data is provided in the form of scores indicating the relative emphasis on various topic classes, including Economics (score=15), Politics (score=7), and Religion (score=2). Also indicated is the format of the content, which in this example is the desired MS Word. Also conveyed in the VDE container 4012 are a control set indicating, among other things, that the price is $2.98 and no modifications are allowed.

In other examples, the classes might have been much more narrow, for example, "Clinton," "Greenspan", Federal Reserve Policy, Interest Rates. Also, the customer might have requested only those documents for which controls could be obtained that permitted modifications and/or excerpting and/or derivative works. In another example, the matching and classification utility 900 may send one or more administrative events and/or classification and/or matching methods to the customer so that these methods could be applied by the customer. Alternatively, the customer may have send one or more methods as part of a smart object to one or more information providers in search of information meeting the desired criteria.

Example: The Atomic Aspects Can Support Automated Extraction Of Portions Of A Content Unit For Aggregation With Topically Consistent Portions And/Or Units From Other Sources

Not only may people desire specific information, but that information may come from different parts of the same object or parts of two or more objects. The matching and classification utility 900 can support the use of smart, classification based extraction and aggregation methods. as shown in example 4100, Figure 66, where two documents 4102(1,2) have been classified by the matching and classification utility 900 into "chunks" or subobjects reflecting topic classes and VDE controls have been provided for each chunk. The "chunking", classification, and control set creation may be performed and stored in a database and/or may be performed "on the fly" or as needed.

To satisfy a request for information concerning travel to and in the United Kingdom plus background information, an information provider extracts parts of each document in the desired classes and creates a new, recombinant document comprised of the subobjects and packages the new document with appropriate controls in a VDE container 4102(n). VDE controls for the subobjects may also be carried along and may be modified by the provider and/or other participants in a chain of handling and control.

The request for information may have been generated using any query and/or search method, including semantic, Boolean, heuristic,

concept-based, and other approaches, and may have been generated explicitly and intentionally by a user and/or other value chain participant, or may have resulted more automatically from the analysis by a matching and classification utility 900 of usage, audit, and/or other rights management information and/or of "info exhaust," and/or of preference, demographic, and/or psychographic data and/or classes of data.

In another example, the matching and classification utility 900 may have sent administrative events and/or classification, search, and/or subobject combining methods 4106 to a provider and/or to a user for execution under the control of a local VDE node.

Example: Matching And Classification Utility 900 Supports Classification For Subsets Of Content Within A Content Unit (Nested Virtual Classifications)

Not only may the matching and classification utility 900 assist in locating whole objects, it may also assist in identifying and/or classifying any number of subobjects for a given whole. New control sets may be associated with each of these subobjects. These new control sets may differ from the control set that applies to the object as a whole. This capability allows matching and classification utility 900 and others value chain participants to locate desired classes of content that may be part of a larger object and possibly to retrieve, pay for, manage, use, or combine these parts in addition to, and/or instead of the whole object.

In example 4200, Figure 67, a VDE container 4202 created by the matching and classification utility 900 holds a text document that in this non-limiting example is the US "State of the Union Address." The matching and classification utility 900 has first classified the entire document in the class "politics." The matching and classification utility 900 has also identified various subparts or subobjects and has classified each them into different classes or categories. In this example, the different classes represent different topic categories.

A user and/or other value chain participant may request only subobjects that have been categorized in one or more desired class(es). The desired subobjects may be packaged in a VDE container 4204 along with appropriate VDE controls for both the overall, new composite object and/or for each of the desired subobjects. (The VDE controls can also be sent separately from the content subobjects.) These controls may pertain to the new whole object created from subparts selected on the basis of their membership in one or more specified class(es) and/or to the whole, new object comprised of these selected subobjects. In another example, the subobjects may be drawn from different documents sharing the same overall topic, for example, from State of the Union addresses given in different years.

In one example, any value chain participant may send distribute one or more subparts of the original object.

In another example, the matching and classification utility 900 may send one or more administrative events and/or methods 4206 to value chain participants who may execute the methods to perform the operations to identify subobjects and/or to subset the whole object in to parts based on class assignments.

Search engines can also use the subobject classifications to provide more precise results. For example, a search engine may have retrieved the State of the Union Address because the search criteria were "US politics speeches," but the whole or part of the object may also have been retrieved searching for "US politics speeches welfare" or "speeches US president defense."

Example: Matching And Classification Utility 900 Supports Classes Of Classes Based On Object Identifier Standards And/Or Other Object Metadata

Among the numerous advantages of the present inventions is the ability to create classes of classes based in part on rights management information. The feature may enhance search efficiency by enabling search engines to locate members of classes provided by any of numerous schemes for object naming and object metadata that have been proposed. For example, the IETF Uniform Resource Locator (URL), the International Standard Book Number (ISBN), International Standard Serial Number (ISSN), MARC library catalog records, and the recent proposed "Dublin Core"(Weibel, Stuart, Jean Godby, Eric Miller, and Ron Daniel, "OCLC/NCSA Metadata Workshop Report", URL http://www.oclc.org:5047

/oclc/research/conferences/metadata/ dublin_core_report.html) are non-limiting examples of prior classifications that can themselves be classified using the present inventions.

Example 4300, Figure 68A-68B, shows several objects 4304(1)-4304(n) each of which may have associated with it various metadata 4302(1)-4302(n) that locates the object in one or more classes, non-limiting examples of which may include network address (URL), price, control set information, permission strings, subject category, title, and publisher.

In example step "1," object metadata 4302 is sent to a matching and classification utility 900 which (example step "2") may create new "classes of classes" 4306. These new classes 4306 are then made available on a Web page 4308 (example step "3") to interested parties who may then search for objects according to their membership in one (or more) of these new classes of classes. In example step "4" an interested party 4320 sends a VDE container with a request to retrieve the Web page 4308 with the classes of metadata information. The Web server (in example step "5") returns a copy of the page 4312 to the interested user 4320, who (in example step "6") sends a VDE container with a query to the matching and classification utility 900 asking, in this example, for objects in new class 3 that cost less than $1.98, and that grant a "modify" permission. In example step "7," the matching and classification utility 900 returns a VDE container 4316 with list of objects that match the criteria. The matching and

classification utility 900 may, in turn, provide URLs or other location information for at least one member of the desired class(es) in the list in container 4316.

Example: Matching and Classification Utility 900 Supports Electronic Gambling

Electronic gambling may be among the services that will drive Internet growth in coming years. Such services raise many questions for both providers and for users or players of the service. For example, providers want to be able create attractive, compelling entertainment experiences and in doing so, capture an important share of their intended markets. Users of these services will of course want to locate the most stimulating, entertaining, and perhaps most of all, rewarding gambling experiences.

Gambling providers may, in one example, differing classes of games, rules, payoffs, odds, and/or interfaces. The present inventions can assist players in identifying the nature of various classes and locating specific instances of one or more classes. Within a particular class of games, for example, players may be particularly interested in the odds at the game of blackjack. In one example, a player may prefer playing with a single digital deck of 52 cards and a particular number of (emulated) shuffles rather than with say four decks and more shuffles, the affect of the latter being to create a more random distribution. Smaller decks and fewer shuffles may make it easier to count cards and/or to otherwise increase the odds in favor of the player, or at least in favor of the experienced, knowledgeable player.

In example 4400, shown in Figure 69, an arbitrary number of gamblers 4402(1)-4402(n) whose usage information flows in VDE containers 4404(1)-4404(n) to a usage clearinghouse 300. The usage clearinghouse 300 sends in VDE containers 4406 at least some of this usage information to a matching and classification utility 900. In another example, the usage information may be sent directly from at least one user to the matching and classification utility 900. In this example, an arbitrary number of gambling providers 4406(1)-4406(n) may also send in VDE containers 4408(1)-4408(n) descriptive and/or usage information to the matching and classification utility 900. Based on available information from relevant sources, the matching and classification utility 900 may create one or more classes and assign one or more providers, services, and/or users to a class. These class definitions may at least in part be based on privacy-related control information.

In this one example, a gambler 4402(1) sends a VDE container 4410 with a query concerning best odds for blackjack to a matching and classification utility 900, who, in turn, sends back a VDE container 4412 with content indicating that gambling provider 2 gives the best odds in blackjack, "best" here meaning those most favorable to the player. In another example, the gambler may then contact gambling provider 2 to play, and the play may consist of a series of communications in VDE containers between the gambling provider and the gambler.

Example: Matching and classification utility 900 Supports Electronic Ticket Sales and Distribution

The performing arts, exhibitions, theaters, and conferences are some non-limiting examples of events that may require tickets for admission. Electronic ticket agencies on the Internet and other electronic arenas provide a connection between the consumer and producers of the event. Consumers may want to know such information as the nature of the event, what classes of tickets exist for a given event and/or class of events, the price for different classes of tickets to an event, the availability of different classes of tickets to different classes of events, and similar information.

In the example 4500, shown in Figure 70, an arbitrary number of users 4504(1)-4504(n) whose usage information is sent in VDE containers 4508 to a usage clearinghouse 300 who, in turn, may send at least some of this usage information in at least one VDE container 4526 to a matching and classification utility 900. The usage information may reflect past ticket purchases, prices, seating preferences, preferred payment methods, preferred theaters and other venues, and other user preference and historical information.

Various ticket agencies 4506(1)-4506(n) may send information about specific events 4512 (1)-4512(n) and/or information about agency services 4514(1)-4514(n) to the matching and classification utility 900. In another example, an event promoter may send event information directly to the matching and classification utility 900.

In one example, a user wishes to find four seats for a particular concert or class of concerts and/or other events whose cost is not more than $25.00. The user sends a VDE container with a request for information on who can supply the desired tickets to the desired events at the requested price. In turn, the matching and classification utility 900 returns a VDE container indicating that tick agency 2 can provide the tickets.

In this example, user 2 sends a VDE container with a purchase request to ticket agency 2. The purchase request may specify not only the specific event, desired pricing, and class of tickets, seat location, for example, but payment method as well, MasterCard for example. The ticket agency, in turn, may return a VDE container with confirmation of the ticket purchase at a given price, location, date, event, and/or using a particular payment method.

In another example, the tickets may be digital and may have associated with them one or more "seals", digital signatures, and/or certificates indicating the authenticity and/or integrity of the digital tickets.

* * * *

While the inventions have been described in connection with what is presently considered to be the most practical and preferred embodiments, the inventions are not to be limited to the disclosed embodiments but, on the contrary, is intended to cover various

modifications and equivalent arrangements included within the spirit and scope of the appended claims.

WE CLAIM:

1. A method including:

(a) determining at least one class, class hierarchy, classification scheme, category or category scheme;

(b) assigning cases, persons, and/or things to said determined class, class hierarchy, classification scheme, category or category scheme; and

(c) selecting and/or matching cases, persons, and/or things based at least in part on said class, class hierarchy, classification scheme, category or category scheme and/or said assignment,

wherein at least one of said steps (a)-(c) includes the step of using at least some rights management information.

2. A method as in claim 1 wherein said using step includes using at least one control set.

3. A method as in claim 1 wherein said using step includes using at least some information for controlling use of digital information.

4. A method as in claim 1 wherein said using step includes using at least some information for controlling at least one transaction.

5. A method as in claim 1 wherein said using step includes using at least some information for controlling at least one event.

6. A method as in claim 1 wherein said using step includes using at least some information for controlling at least one consequence of digital information use.

7. A method as in claim 1 wherein said using step includes using at least some information for controlling at least one consequence of at least one event.

8. A method as in claim 1 wherein said using step includes the step of using at least some information for controlling at least one consequence of at least one transaction.

9. A method as in claim 1 wherein said using step includes using at least some information outputted by a rights management process.

10. A method as in claim 1 further including the step of outputting at least some rights management information.

11. A method as in claim 1 wherein at least one of steps (a)-(c) includes using at least one secure container.

12. A method as in claim 1 wherein at least one of steps (a)-(c) includes using at least one protected processing environment.

13. A method as in claim 1 further including the step of using at least one of the techniques set forth at pages 60-82 of this specification.

14. A method as in claim 1 wherein said using step includes using at least one or more rules and/or their consequences.

15. A method as in claim 1 wherein at least one of steps (a) and (b) includes at least one of the following steps:

(a) using at least one statistical technique identifying at least one cluster of cases sharing similar profiles and/or features;

(b) using numerical taxonomy;

(c) using at least one of cluster analysis, factor analysis, components analysis, and other similar data reduction/classification technique;

(d) using at least one pattern classification technique, including components analysis and neural approaches;

(e) using at least one statistical technique that identifies at least one underlying dimension of qualities, traits, features, and/or characteristics, and assigning parameter data indicating the extent to which a given case has, possesses, and/or may be characterized by the underlying dimension, factor, class, and/or result in the definition of at least one class and/or the assignment of at least one case to at least one class;

(f) using at least one statistical method employing fuzzy logic and/or fuzzy measurement and/or whose assignment to at least one class entails probabilities different from 1 or zero;

(g) using a Baysian statistical classification techniques that uses an estimate of prior probabilities in determining class definitions and/or the assignment of at least one case to at least one class;

(h) using at least one statistical and/or graphical classification and/or data reduction method that uses rotation of reference axes, regardless of whether orthogonal or oblique rotations are used;

(i) using at least one statistical method for two and three way multidimensional scaling; and

(j) using at least one knowledge based approach to classification.

16. A system including:

an automatic class generator that generates at least one class, class hierarchy, classification scheme, category or category scheme;

an automatic class assigner that assigns cases, persons and/or things to said determined class, class hierarchy, classification scheme, category or category scheme; and

at least one further component for automatically searching, selecting and/or matching cases, persons, and/or things based at least in part on said class, class hierarchy, classification scheme, category or category scheme and/or said assignment,

wherein said system uses at least some rights management information.

17. A system including:

first means for determining at least one class, class hierarchy, classification scheme, category or category scheme;

second means for assigning cases, persons, and/or things to said determined class, class hierarchy, classification scheme, category or category scheme; and

third means for selecting and/or matching cases, persons, and/or things based at least in part on said class, class hierarchy, classification scheme, category or category scheme and/or said assignment,

wherein at least one of said first, second and third means uses at least some rights management information.

18. A Commerce Utility System providing a secure execution space, the Commerce Utility System performing at least one component based service function including at least one secure component for execution within the secure execution space, the Commerce Utility System including a communications facility permitting communication of secure control information with at least one electronic community participant,

wherein said component based service function uses at least one class based at least in part on rights management information.

19. A Commerce Utility System as in claim 18 wherein the component based service function assigns at least one member to at

least one class based at least in part on some rights management information.

20. A Commerce Utility System as in claim 18 wherein the component based service function matches persons and/or things based at least in part on at least some rights management information.

21. A Commerce Utility System as in claim 18 wherein the component based service function selects persons and/or things based at least in part on at least some rights management information.

22. A Commerce Utility System as in claim 18 wherein the component based service function narrowcasts information to recipients based at least in part on at least some rights management information.

23. A system or method including:

a computer network and

a control arrangement within the network that determines and/or uses at least one of the following through use of rights management information:

(a) class hierarchy,

(b) class structure,

(c) classification scheme,

(d) category, and

(e) category scheme.

24. A class-based system including at least one computer that processes digital information, said system including at least one element that uses at least some rights management information.

25. A method of operating a class-based system including at least one computer that processes digital information, said method including the step of using at least some rights management information.

26. A system for assigning at least one thing or person to at least one class including at least one computer that processes digital information, said system including at least one element that uses at least some rights management data in making said assignment.

27. A system for making and/or using at least one class-based assignment including at least one computer that processes digital information, said system including at least one element that uses at least some rights management information.

28. A system for clearing at least one transaction including at least one computer that processes digital information, said system including at least one element that uses at least one class defined, assigned, selected, and/or matched based at least in part on rights management information.

29. A method for authorizing at least one computer and/or computer user including the step of using at least one class defined,

assigned, selected, and/or matched based at least in part on rights management information.

30. A method for authorizing at least one electronic transaction including the step of using at least one class defined, assigned, selected, and/or matched based at least in part on rights management information.

31. A method for initiating and/or performing at least one at least in part secure electronic transaction including the step of using class related information defined, assigned, selected, and/or matched based at least in part on rights management information.

32. An information processing method including the steps of:

securely charging a fee; and

conditioning said charging step at least in part on at least one class defined, assigned, selected, and/or matched based at least in part on rights management information.

33. A method for securely exchanging digital information including the step of at least in part defining, assigning, selecting, and/or matching at least one class based at least in part on rights management information.

34. A method for performing at least one rights operating system based transaction including the step of defining, assigning,

selecting, and/or matching at least one class based at least in part on rights management information.

35. A method for performing at least one protected processing environment operation including the step of defining, assigning, selecting, and/or matching at least one class based at least in part on rights management information.

36. A method of pushing information including the steps of classifying recipients and/or information to be sent to said recipients based at least in part on rights management information, and selecting said information to distribute to said recipients based at least in part on said classifying.

37. A method of pushing information including the steps of classifying recipients and/or information to be sent to said recipients based at least in part on rights management information, and matching at least a portion of said information with at least one class of said recipients based at least in part on said classifying.

38. A method of pushing information as in claim 37 further including the step of creating a classification scheme and/or hierarchy using at least some rights information.

39. A method of pushing information as in claim 37 further including the step of assigning at least some information and/or at least one recipient to a class or category, said assignment based at least in part on rights management information.

40. A subject switch for matching subscribers and/or recipients desiring information in one or more classes with one or more sources of information, wherein the subject switch matches at least one subscriber and/or participant with at least one information source on a mapping based at least in part on rights management information.

41. A subject switch as in claim 40 wherein said information source:

selects at least some information, said selection based on at least one class, and wherein said assignment of said at least some information to said at least one class is based at least in part on rights management information; and

sends at least some said selected information to said subscriber in accordance with said subscriber's subscribing to said class of information.

42. A subject switch as in claim 40 wherein at least one of said subject switch, said subscriber and/or participant and said information source includes at least one computer providing a protected processing environment.

43. A subject switch as in claim 40 wherein at least one subscriber and/or participant uses rights management information at least in part to persistently subscribe to at least some information provided by at least one information source.

44. A subject switch as in claim 40 wherein the subject switch includes means for using at least one class definition for said mapping.

45. A subject switch as in claim 40 wherein the subject switch includes means for responding to a subscriber and/or participant request by providing information indicating information sources in at least one specified or desired class.

46. A subject switch as in claim 40 further including a messaging service for use by at least two of said subject switch, said subscriber and/or participant and said information source and/or participant to communicate electronically.

47. A subject switch as in claim 46 wherein said electronic communications uses at least one secure container.

48. A subject switch as in claim 40 wherein at least one of said subject switch, subscriber, or information source uses at least one control set associated with at least some information received by at least one subscriber.

49. A digital narrowcasting arrangement comprising:

a computer; and

at least one classifying element used to select content to narrowcast to recipients based at least in part on rights management information.

50. A digital narrowcasting arrangement as in claim 49 wherein the classifying element classifies at least one of (a) a recipient, and (b) content, based at least in part on rights management information.

51. A digital narrowcasting arrangement as in claim 49 wherein said classifying element defines at least one class using at least some rights management information.

52. A digital narrowcasting arrangement as in claim 49 wherein the classifying element assigns at least some content to at least one class, said assignment based on at least some rights management information.

53. A digital narrowcasting arrangement as in claim 49 wherein the classifying element defines at least one class based at least in part on content selections previously made by the recipients and/or profiles generated based at least in part on recipient input.

54. A digital narrowcasting arrangement as in claim 49 wherein the classifying element sends a content request including classification data and destination information to at least one provider.

55. An information distribution system including: a computer network; and a selection arrangement that selects information for use by individual recipients using classes based at least in part on rights management information.

56. An information distribution system as in claim 55 wherein the system further includes a classifying element that determines at least one class of content and/or service of interest to at least one recipient.

57. An information distribution system as in claim 56 wherein said classifying element defines at least one class using at least some rights management information.

58. An information distribution system as in claim 56 wherein said classifying element assigns at least some content to at least one class, said assignment based on at least some rights management information.

59. An information distribution system as in claim 55 wherein the system includes means for allowing the user to choose to receive the selected information.

60. An enterprise information system including a computer system for classifying employees, said system including at least one rights management component that distributes information to the employees based at least in part on employee classification.

61. An enterprise information system as in claim 60 wherein the computer matches the information to employees based at least in part on the employee classification.

62. An enterprise information system as in claim 60 wherein the employee classification is used to gather information for

employees without revealing substantial information concerning individual employees.

63. A method for conducting a chain of handling and/or control including the steps of allowing plural parties to contribute rules and/or consequences, and performing at least one classification based at least in part on said rules and/or consequences.

64. A method as in claim 63 wherein at least some of said contributed rules and/or consequences are class based.

65. A method as in claim 63 wherein at least one of said parties modifies at least one of said rules and/or consequences based at least in part on class.

66. A method as in claim 63 including the step of generating class assignments based at least in part on said rules and/or consequences, and sending said class assignments to at least one clearinghouse.

67. A method as in claim 63 including the step of classifying said rules and/or consequences to provide at least one class, and fulfilling at least one request by selecting based on said class.

68. A directory services system for classifying confidential information, the system including:

a communications component that receives directory requests; and

a response component that uses said classification to respond to

directory requests while preserving confidentiality of said confidential information.

69. A directory services system as in claim 68 wherein said response component uses at least one classification process to classify items in a directory, and uses results of the classification process, at least in part, to respond to directory requests.

70. A directory services system as in claim 68 wherein said response component sends information to destinations revealed by the results of the classification process without revealing at least some information concerning said destinations to the information source.

71. A microsegmented merchandising technique including the steps of performing classification based at least in part on usage data and/or lifestyle profiles, and distributing offers for products and/or services based at least in part on the classification.

72. A microsegmented merchandising technique as in claim 71 wherein the performing step includes defining at least one class hierarchy based at least in part on rights management information.

73. A microsegmented merchandising technique as in claim 71 further including the step of combining plural offers for different products and/or services based at least in part on said classification.

74. A trading network including:

a communications element for communicating digital signals; and

means for matching value chain participants through a classification based at least in part on rights management information.

75. A trading network as in claim 74 further including means for defining at least one class hierarchy based at least in part on rights management information.

76. A trading network as in claim 74 further including means for determining class membership based at least in part on action and/or information provided by at least one value chain participant.

77. A trading network as in claim 74 wherein said matching means includes means for at least in part performing at least one electronic negotiation.

78. A securities trading method including the step of performing a classification process at least in part using at least one rights management element, and using the classification process to select securities for trade.

79. A securities trading method as in claim 78 wherein said classification process includes defining at least one class hierarchy based at least in part on rights management information.

80. A currency/debt trading system including:

a currency or debt trading computer; and

an arrangement coupled to said computer that performs at least

one classification process based at least in part on rights management information.

81. A currency/debt trading system as in claim 80 wherein said arrangement includes means for defining at least one class hierarchy based at least in part on rights management information.

82. A currency/debt trading system as in claim 80 wherein the arrangement uses classification to maximize return or minimize loss.

83. A financial institution selection system including a computer that classifies financial institutions based at least in part on rights management information.

84. A software distribution method including the steps of generating class information based at least in part on rights management information, and selecting software to be distributed and/or recipients who are to receive distributed software based at least in part on class information.

85. A software distribution method as in claim 84 wherein said generating step includes defining a class hierarchy using at least some rights management information.

86. A software distribution method as in claim 84 wherein the selecting step includes selecting software to be distributed by classifying the software based at least in part on rights management information associated with the software.

87. A software distribution mehtod as in claim 80 wherein the selecting step includes selecting recipients to receive software based at least in part on usage information provided by a rights management process.

88. A classification technique including the step of authenticating class membership based at least in part on digital credentials and/or certificates.

89. A classification technique as in claim 88 wherein said digital credentials are digital certificates.

90. A classification technique as in claim 88 wherein said digital credentials are digital membership cards.

91. A classification technique as in claim 88 further including the step of deciding class membership based at least in part on rights management information.

92. A classification technique as in claim 88 further including the step of classifying at least one of users, nodes, devices, networks, servers, clients and services based at least in part on rights management information.

93. A classification technique as in claim 88 further including the step of conditioning at least one rights management process at least in part on authenticated class membership.

94. A computer system including:

a first arrangement that generates class-based controls to participants based at least in part on class and/or class-based assignments; and

a second arrangement that allows participants to interact with information and/or one another at least in part using said class-based controls.

95. A computer system as in claim 94 further including means for using said class-based controls to limit participants' access to information and/or services based on participants' classes.

96. A health care computer system including an arrangement for issuing health care workers, administrators and insurers class-based digital credentials and/or certificates, wherein the digital information sent to said health care workers and administrators includes class-based controls that condition use and/or access to information based at least in part on said class-based digital credentials and/or certificates.

97. A health care computer system as in claim 96 further including means for allowing said health care workers, administrators and insurers sharing a common object subject to class-based controls to have access to different portions of the object based at least in part on said class-based controls.

98. A work process automation system including a matching and/or classification computer that matches tasks to resources based at least in part on assigning classifying the tasks and/or the resources to at least one class.

99. A work process automation system as in claim 98 wherein said matching and/or classification computer includes means for defining at least one class hierarchy based at least in part on rights management information.

100. A work process automation system as in claim 98 wherein said matching and/or classification computer includes means for matching based at least in part on rights management information.

101. An automatic governmental and/or societal rights supporting system including a matching and/or classification computing element that assigns and/or classifies entities to at least one class based at least in part on rights management information.

102. An automatic governmental and/or societal rights supporting system as in claim 101 wherein the matching and/or classification computing element includes means for defining a class hierarchy based at least in part on rights management information.

103. An automatic governmental and/or societal rights supporting system as in claim 101 wherein the matching and/or classification computing element includes means for classifying entities based on at least one of the following:

tax status;

right to receive certain information;

right to engage in certain transactions; and

jurisdiction.

104. An automatic taxing authority computer including means for issuing tax class control sets based at least in part on tax-based class definitions, and means for using said tax control sets at least in part to collect and/or enforce taxation.

105. A method for adaptively presenting information differently to different participants, including associating said participants with classes, and controlling presentation based at least in part on class-based control sets included within the information.

106. A method as in claim 105 further including using said class-based control sets to match participants with different portions of said information.

107. A method as in claim 105 further including using said class-based control sets to change the form in which information is presented based at least in part on said classes.

108. A method as in claim 105 further including the step of operating said class-based control sets based at least in part on metadata associated with different portions of said information.

109. A method as in claim 105 further including selecting said class-based control sets between different images for

presentation based at least in part on one or more classes associated with a participant.

110. A method as in claim 105 further including using said class-based control sets to emphasize certain portions of said information over other portions in said presentation based at least in part on one or more classes associated with a participant.

111. A method as in claim 105 further including using at least one computer having a protected processing environment.

112. A method for adaptively presenting information differently to different participants including:

classifying the different participants based on capability; and

using class-based control sets associated with said information to change the difficulty of the presentation based at least in part on said classification.

113. A method as in claim 112 wherein the different recipients are classified based on grade level.

114. A method as in claim 112 including the step of changing the vocabulary and/or syntactical complexity of the presentation based at least in part on said classification.

115. A method as in claim 112 further including the step of using said class-based control sets to ensure that in at least some cases, recipients in different classes pay different levels of compensation for said presentation.

116. A method for adaptively presenting information differently to different participants including:

classifying different participants based on capability, and

using class-based control sets associated with said information to change the language of the presentation based at least in part on said classification.

117. An information searching mechanism including a matching computer element that classifies information based at least in part on rights management information, said computing element including means responsive to user requests to search for information based at least in part on said classification.

118. An information searching mechanism as in claim 117 wherein said matching computer element further includes means for assigning information to classes based at least in part on rights management information.

119. An information searching mechanism as in claim 117 wherein said matching computer element includes means for scoring information based at least in part on user indicated parameters.

120. An information searching mechanism as in claim 117 wherein said matching computer element includes means for responding to at least some user requests by providing Universal Resource Locator designations of where information can be found.

121. An information handling method including the step of using class-based controls to control support extraction and/or aggregation of information.

122. An information handling method as in claim 121 further including using a computing element to extract information from plural objects based at least in part on class-based criteria.

123. An information handling method as in claim 121 further including using a computing element to aggregate information based at least in part on class-based criteria.

124. An information handling method as in claim 121 further including using said class-based controls to represent nested or multi-level classifications.

125. An information classification method including the step of generating at least one class hierarchy from other plural classification hierarchies based at least in part on rights management information and/or class-based rights management information based at least in part on classification metadata.

126. An information classification method as in claim 125 further including basing said other plural classification hierarchies at least in part on object metadata.

127. An information classification method as in claim 125 further including specifying said classification object metadata

specified classifications based on at least one of location, name, prices, permissions, ISSN, title, author, publisher and/or date.

128. An information classification method as in claim 125 further including generating said class-based rights management information by classifying classes.

129. An electronic gambling system including a computer that matches gamblers with plural gambling providers based at least in part through classifying the gambling providers using rights management information.

130. An electronic gambling system as in claim 129 wherein the computer includes means for classifying the gamblers based at least in part on rights management information.

131. An electronic gambling system as in claim 129 wherein the computer includes at least one protected processing environment.

132. An electronic gambling system as in claim 129 wherein the computer uses at least one control set to classify, select and/or match at least one of said gambling providers, and/or gamblers.

133. An electronic ticketing system including a computer that matches recipients with tickets to events through classifying said recipients, said system including a computer that matches tickets and/or said events based at least in part on rights management information.

134. An electronic ticketing system as in claim 133 wherein a recipient provides a request containing event and rights management criteria, and the computer matches the recipient with a provider based at least in part on said classifying process.

135. An electronic ticketing system as in claim 133 wherein the rights management information includes method of payment information.

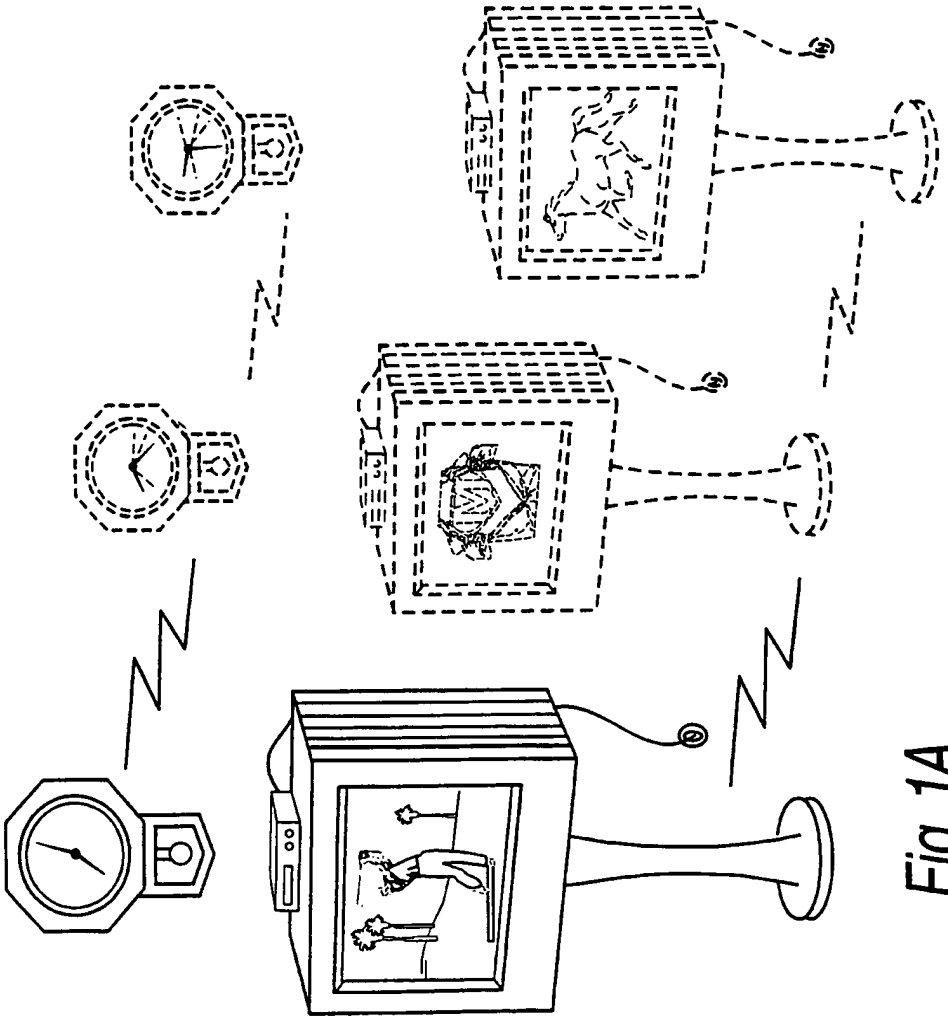
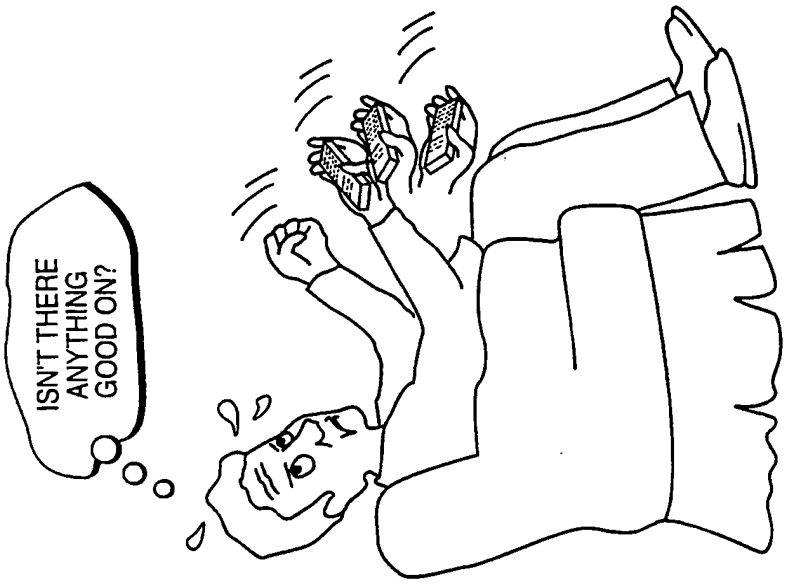


Fig. 1A
(Prior Art)



Petitioner Apple Inc. - Exhibit 1004, p. 5664

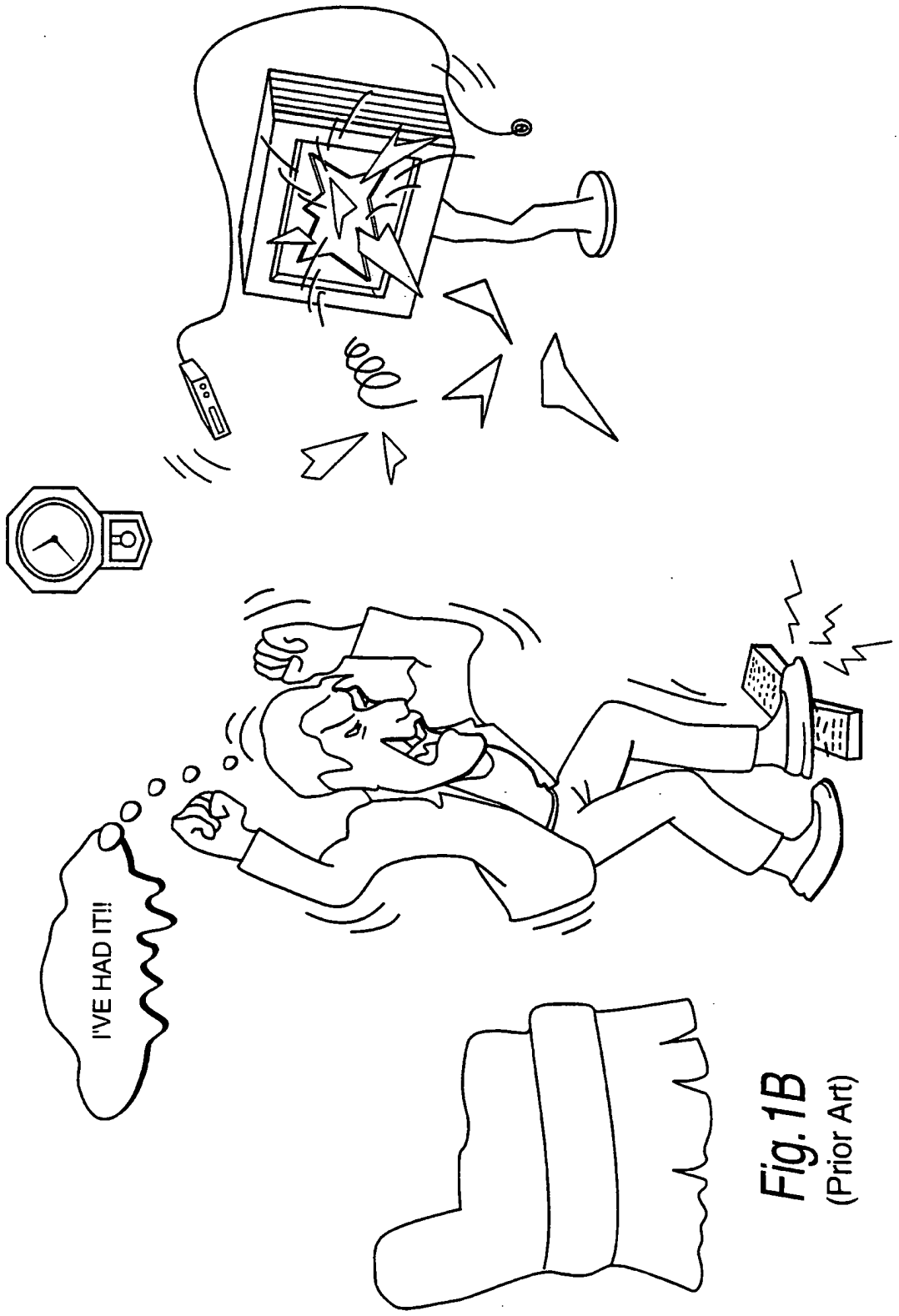


Fig. 1B
(Prior Art)

Petitioner Apple Inc. - Exhibit 1004, p. 5665



Fig. 2
(Prior Art)

Petitioner Apple Inc. - Exhibit 1004, p. 5666

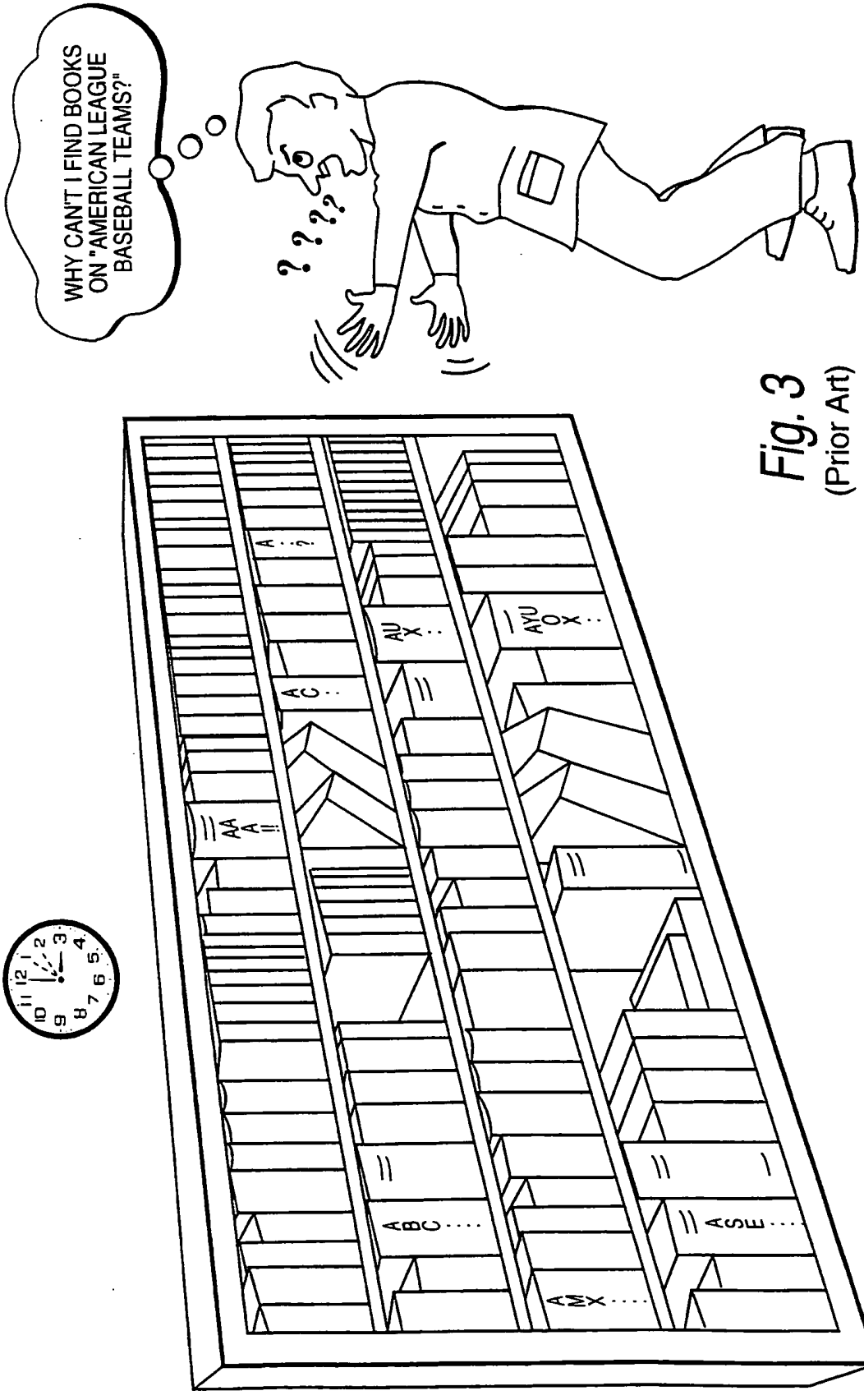


Fig. 3
(Prior Art)

Petitioner Apple Inc. - Exhibit 1004, p. 5667

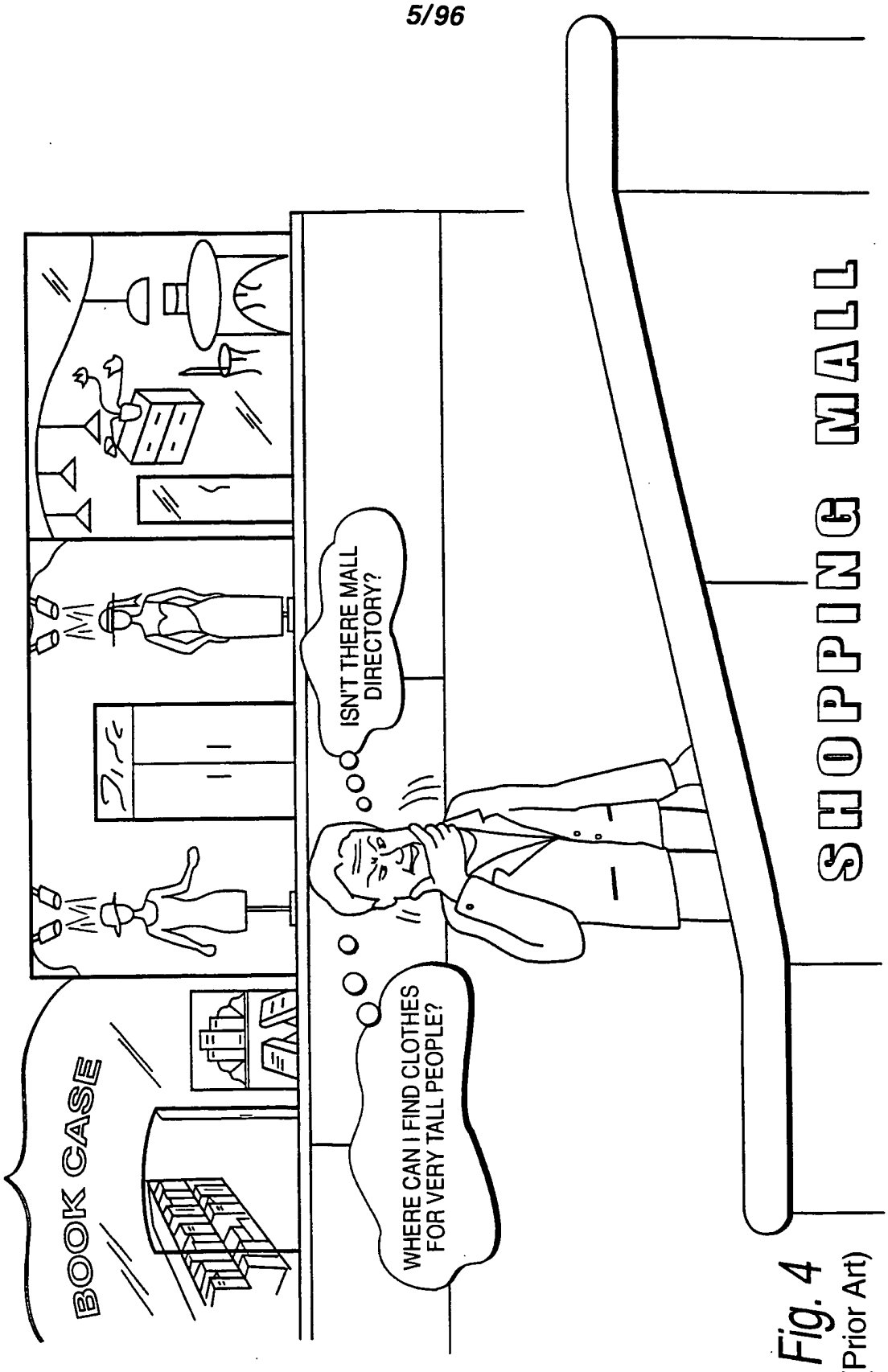


Fig. 4
(Prior Art)

Petitioner Apple Inc. - Exhibit 1004, p. 5668

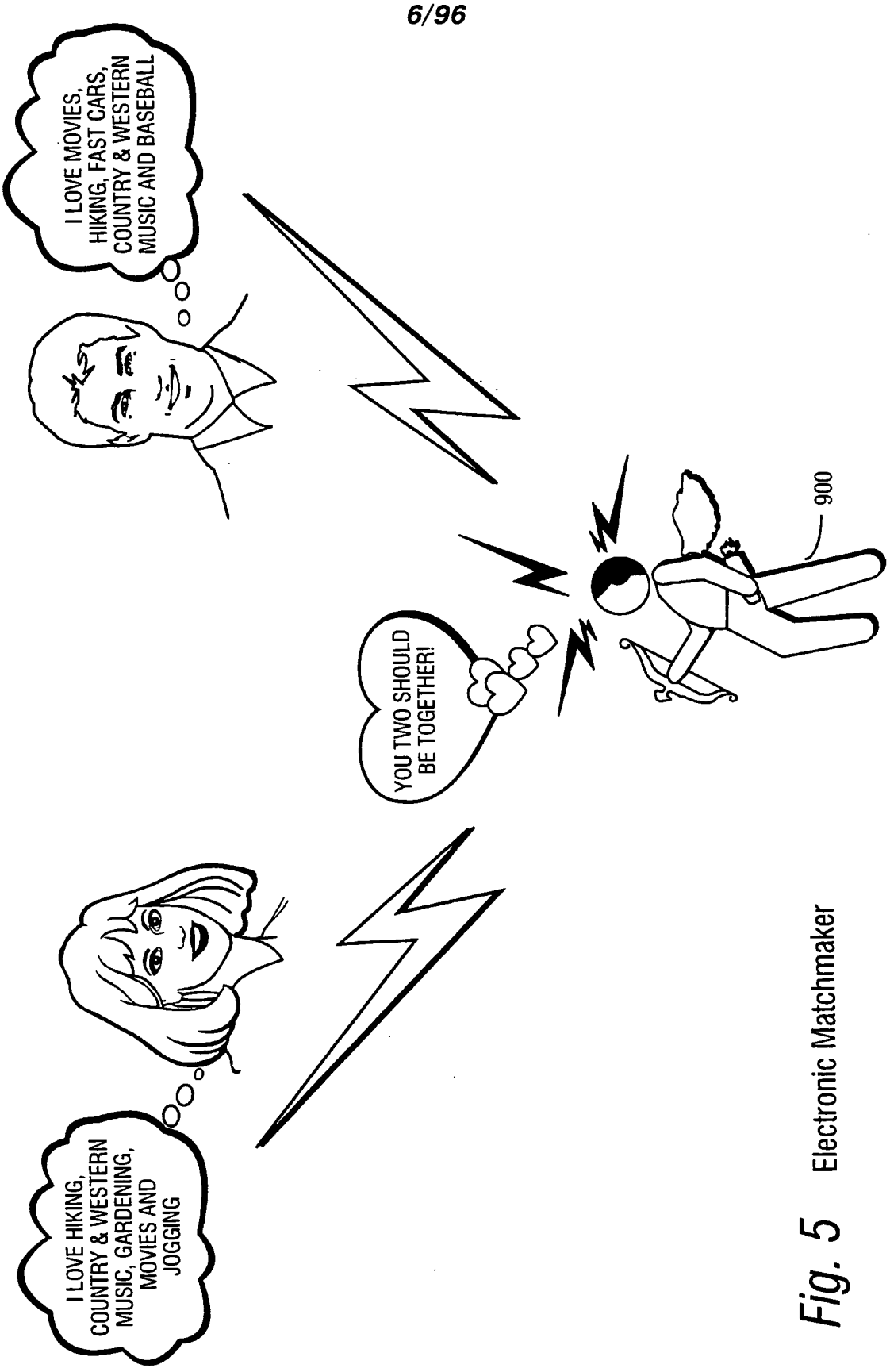


Fig. 5 Electronic Matchmaker

Petitioner Apple Inc. - Exhibit 1004, p. 5669

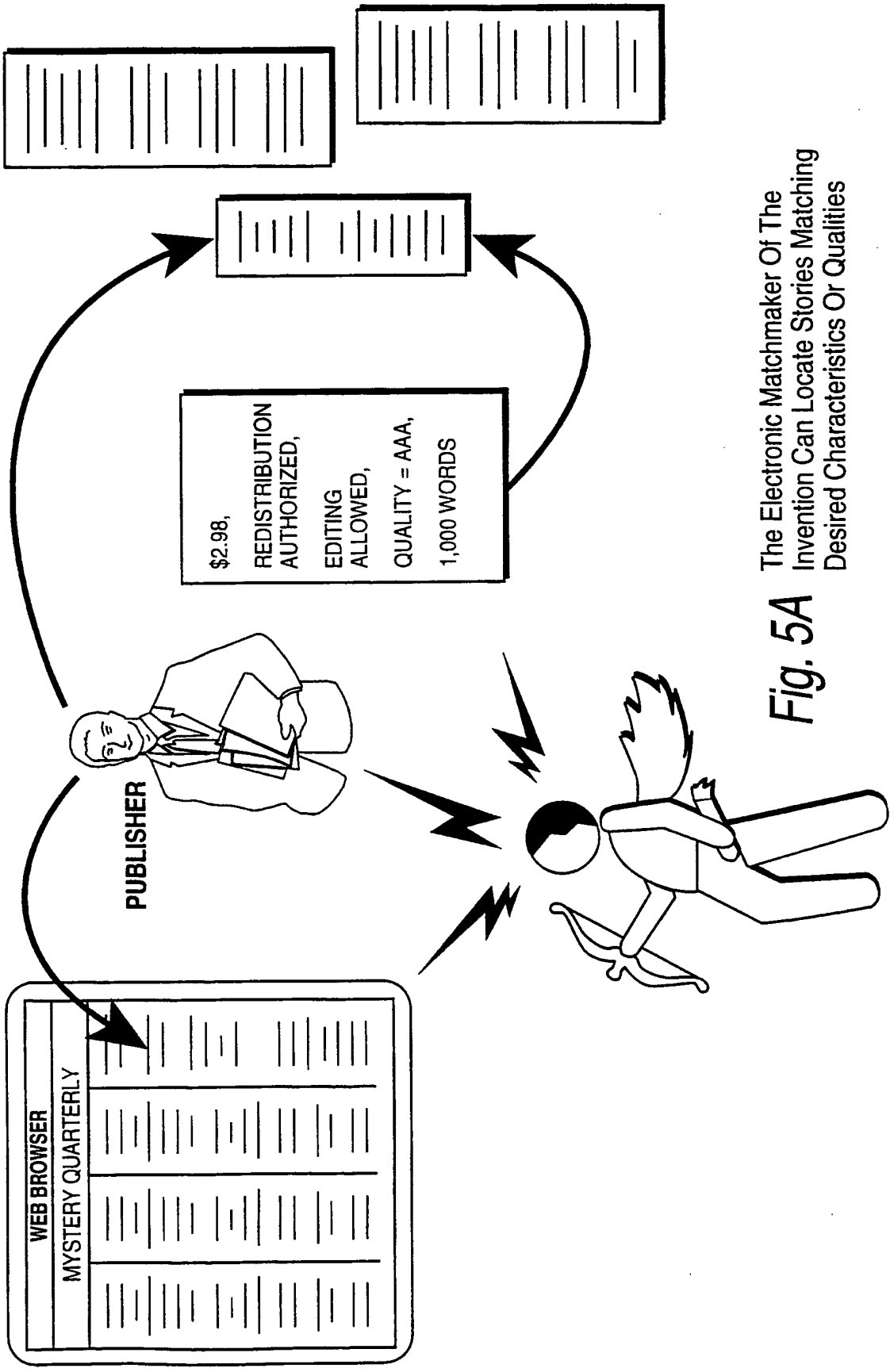


Fig. 5A The Electronic Matchmaker Of The Invention Can Locate Stories Matching Desired Characteristics Or Qualities

Petitioner Apple Inc. - Exhibit 1004, p. 5670

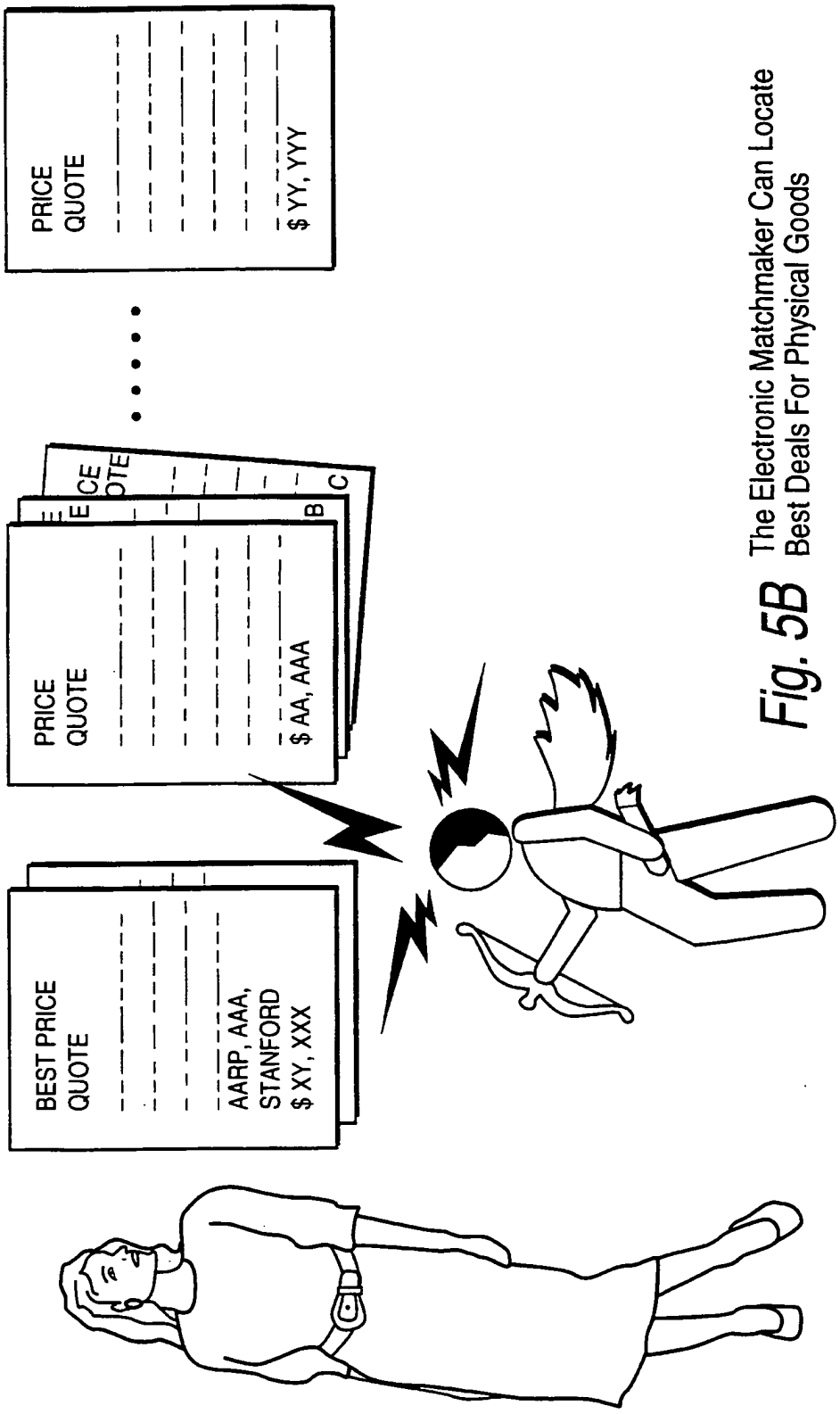


Fig. 5B The Electronic Matchmaker Can Locate Best Deals For Physical Goods

Petitioner Apple Inc. - Exhibit 1004, p. 5671

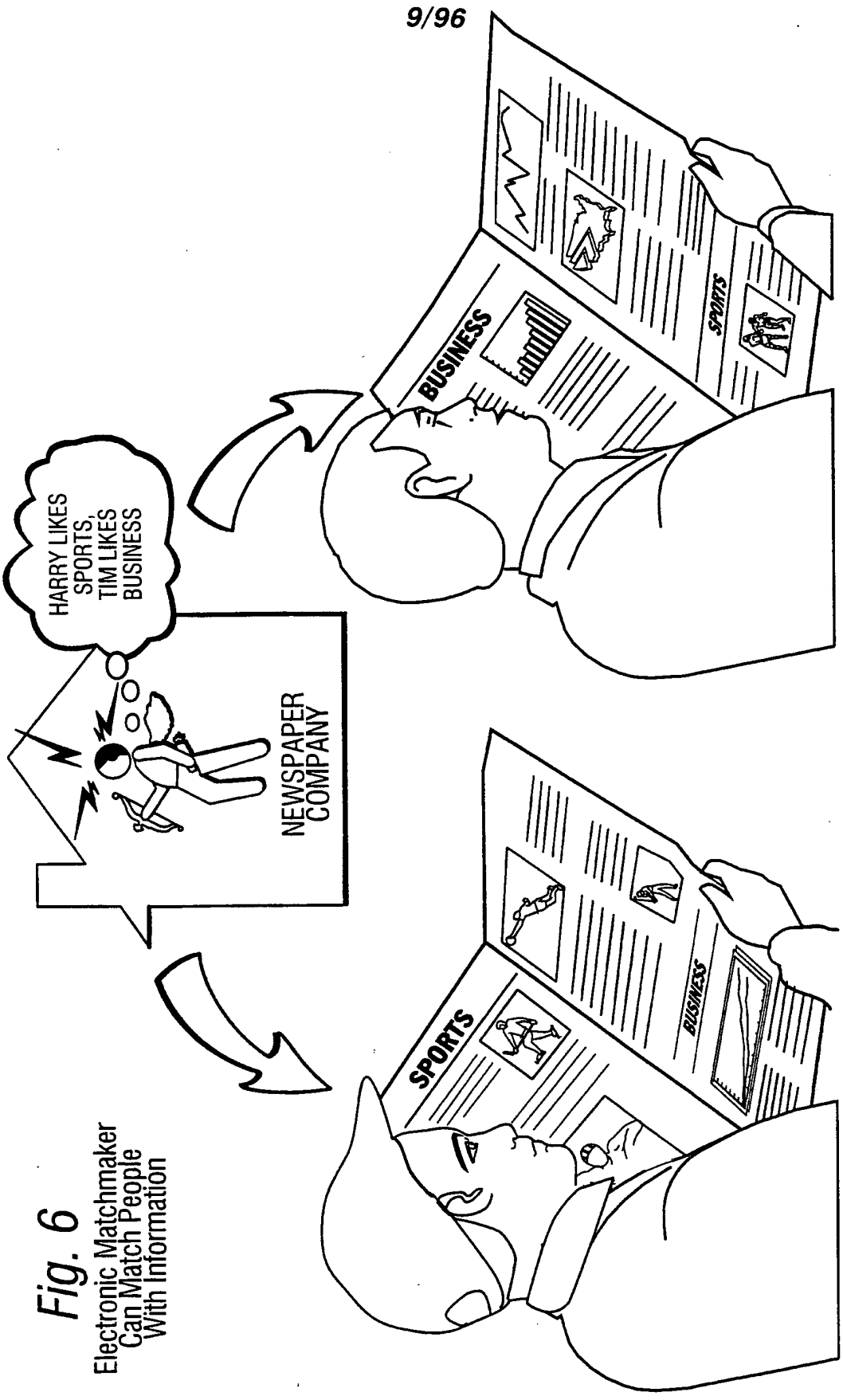


Fig. 6
Electronic Matchmaker Can Match People With Information

Petitioner Apple Inc. - Exhibit 1004, p. 5672

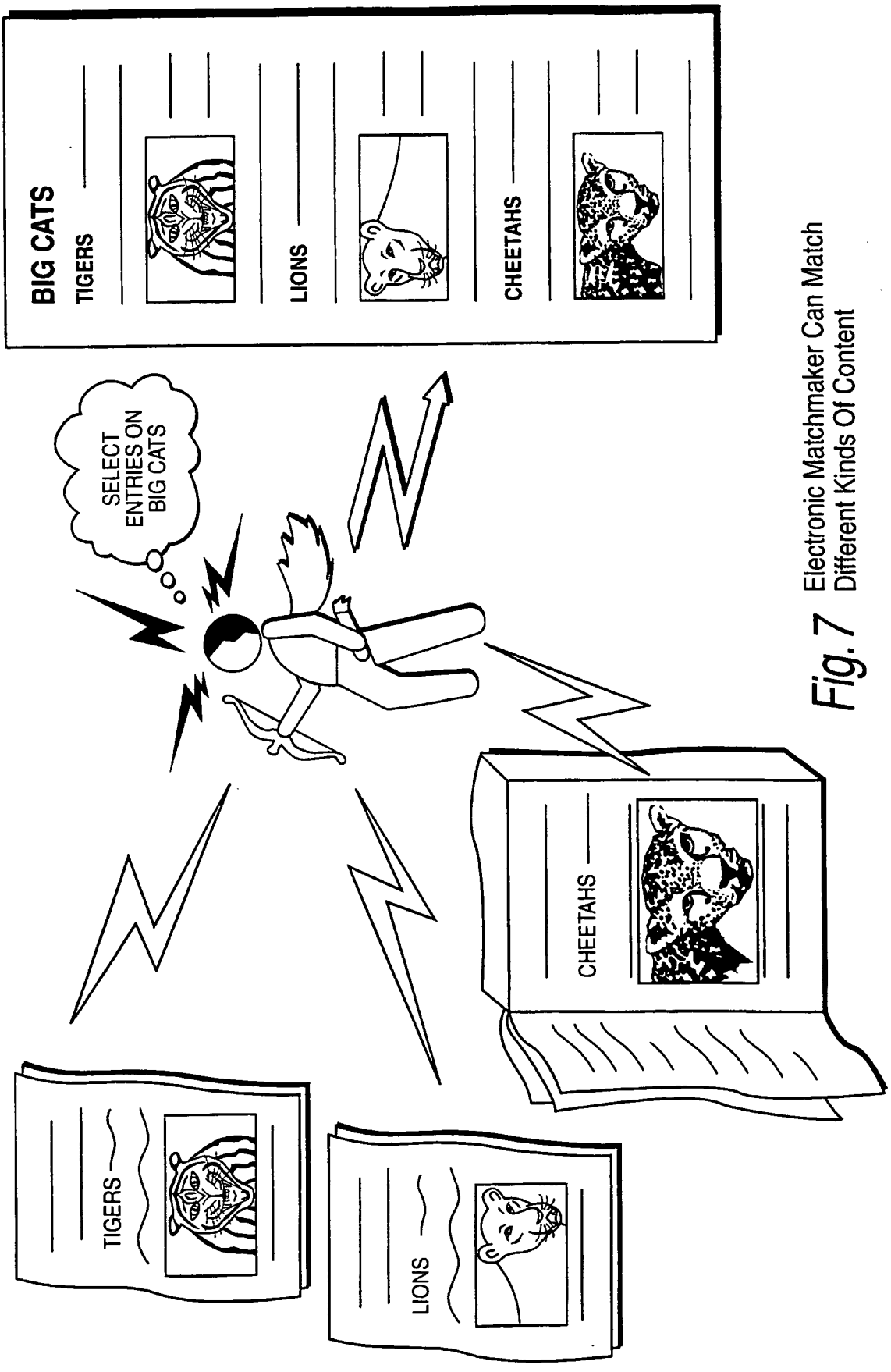


Fig. 7 Electronic Matchmaker Can Match Different Kinds Of Content

Petitioner Apple Inc. - Exhibit 1004, p. 5673

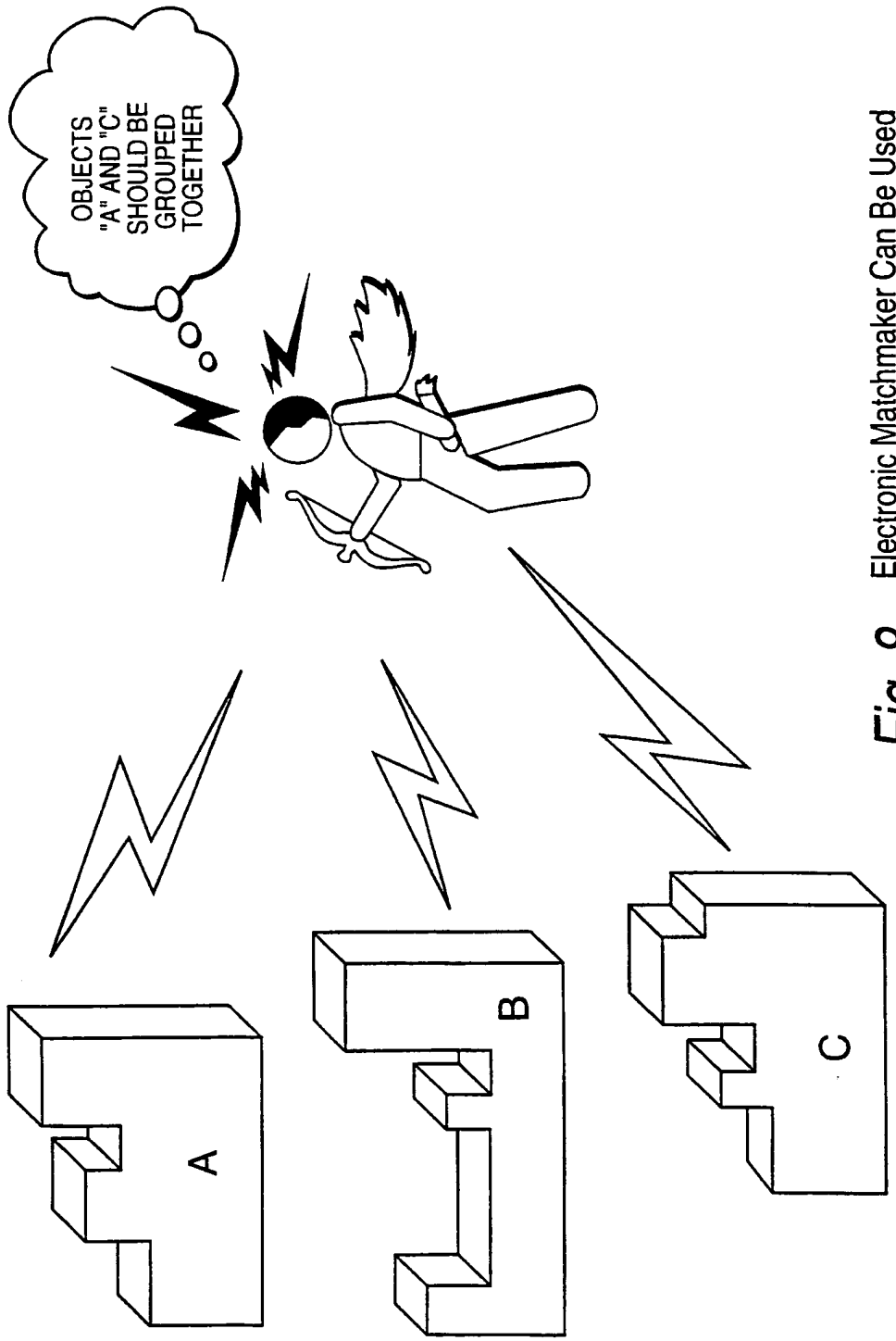


Fig. 8 Electronic Matchmaker Can Be Used For Matching Any Kinds of Things

Petitioner Apple Inc. - Exhibit 1004, p. 5674

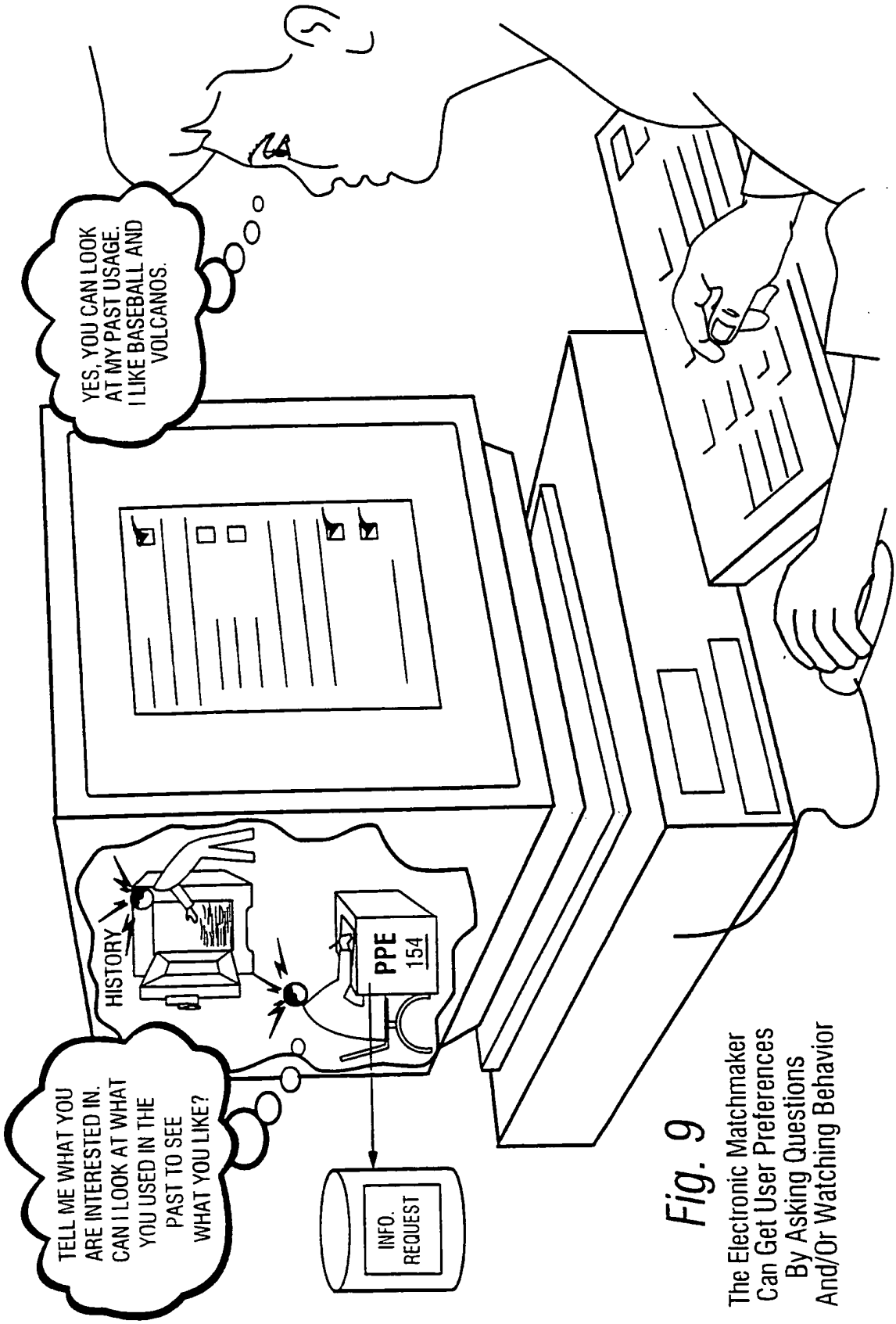


Fig. 9

The Electronic Matchmaker Can Get User Preferences By Asking Questions And/Or Watching Behavior

Petitioner Apple Inc. - Exhibit 1004, p. 5675

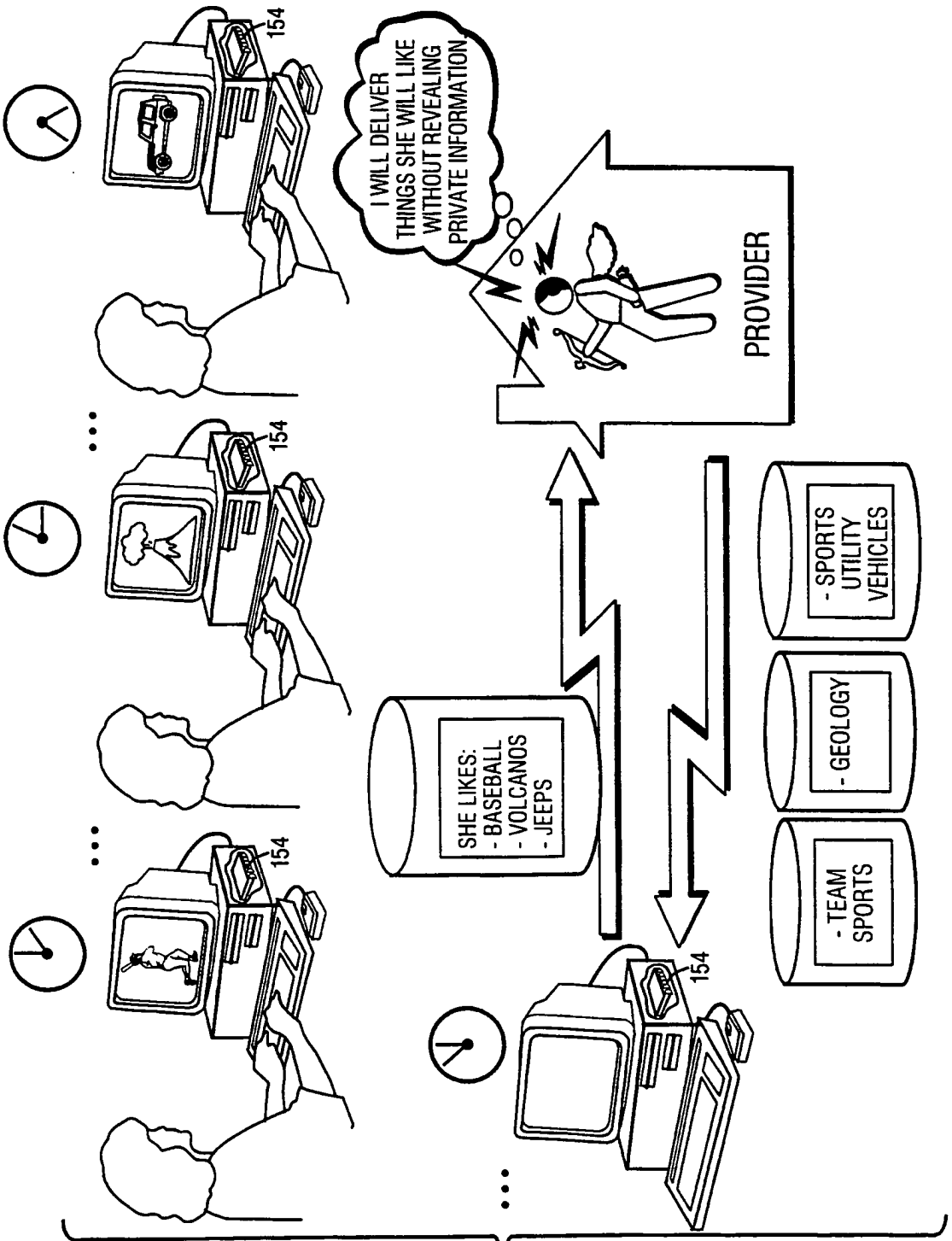


Fig. 10
Example Electronic Matchmaking Process

Petitioner Apple Inc. - Exhibit 1004, p. 5676

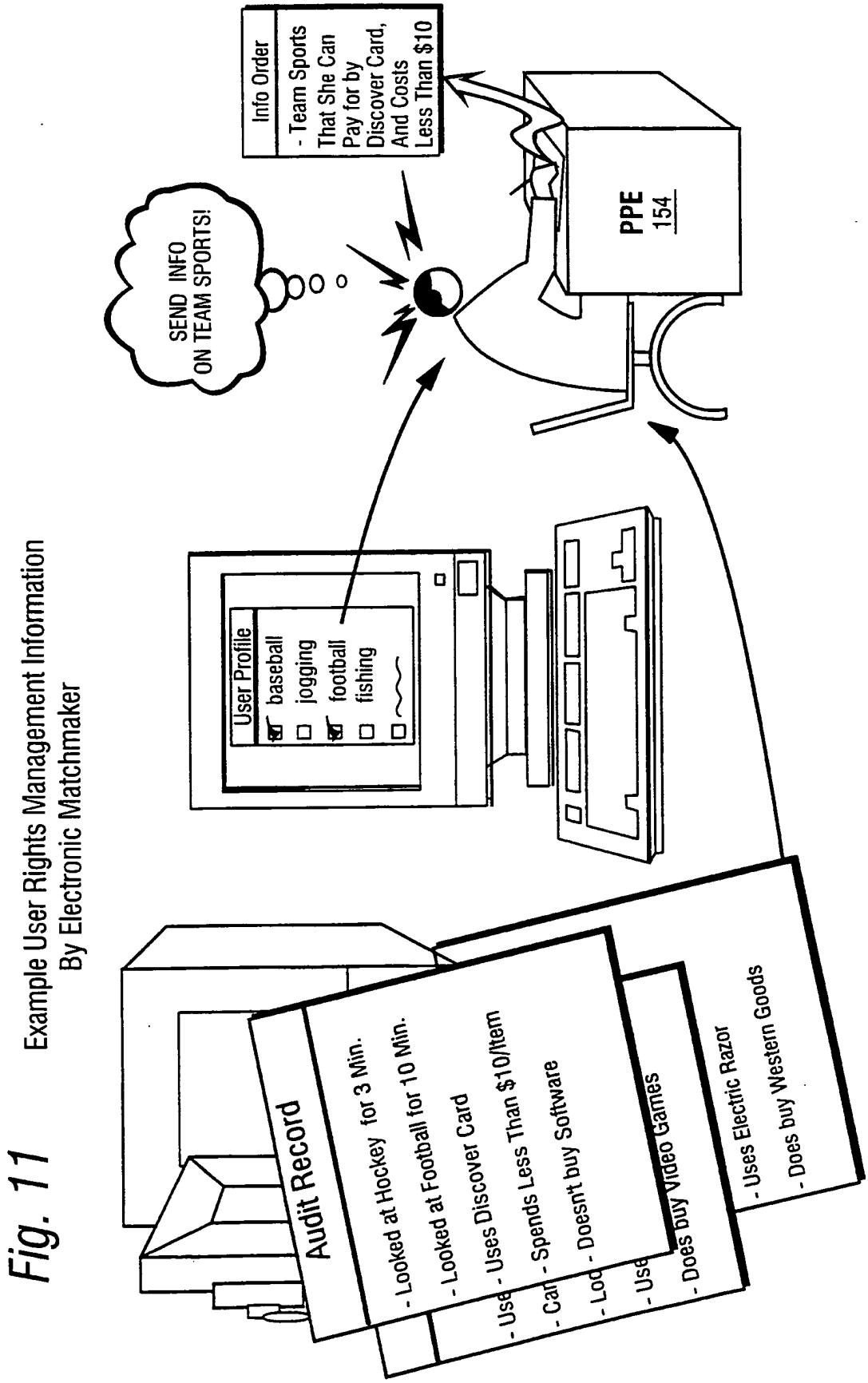


Fig. 11 Example User Rights Management Information By Electronic Matchmaker

Petitioner Apple Inc. - Exhibit 1004, p. 5677

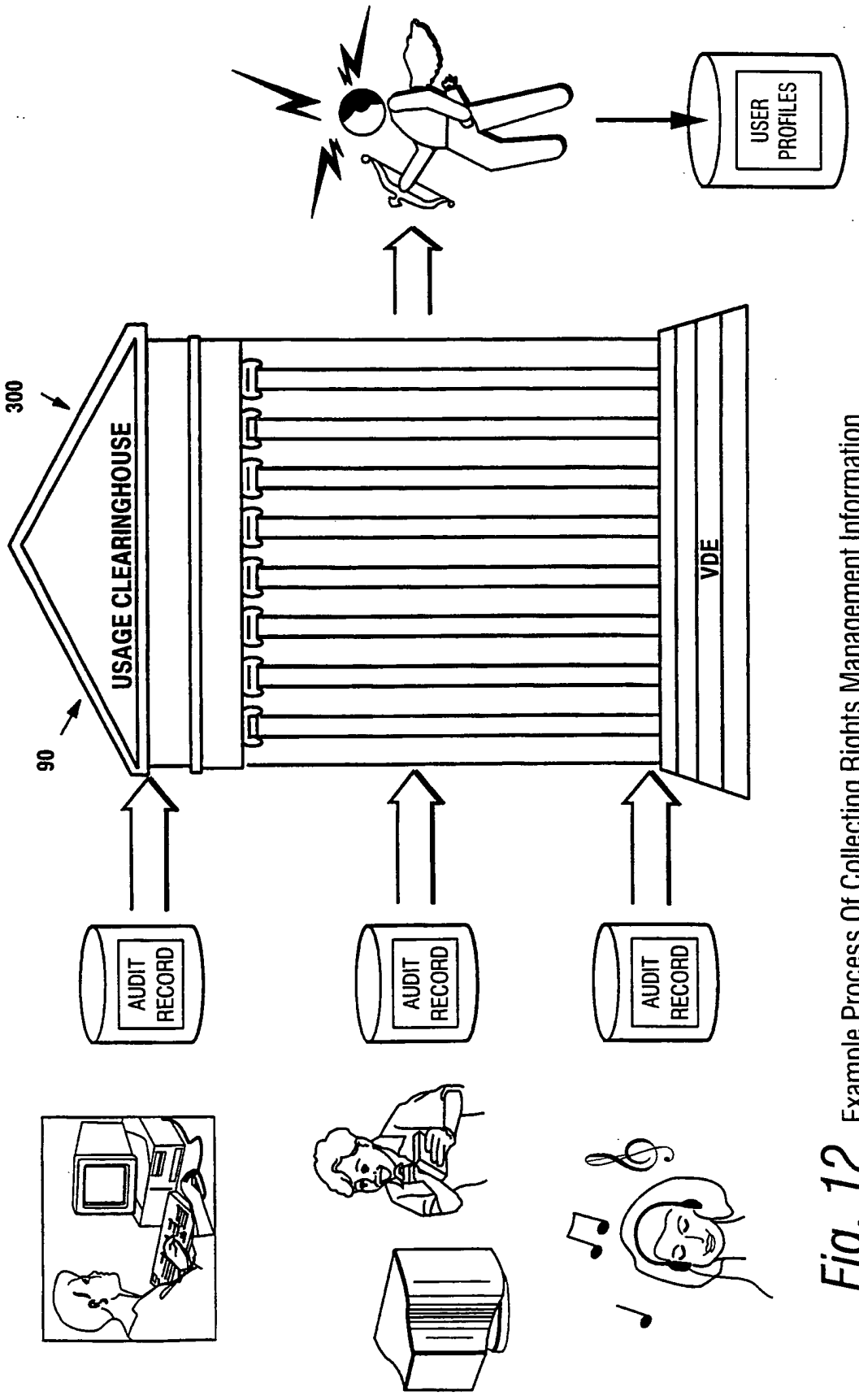


Fig. 12 Example Process Of Collecting Rights Management Information

Petitioner Apple Inc. - Exhibit 1004, p. 5678

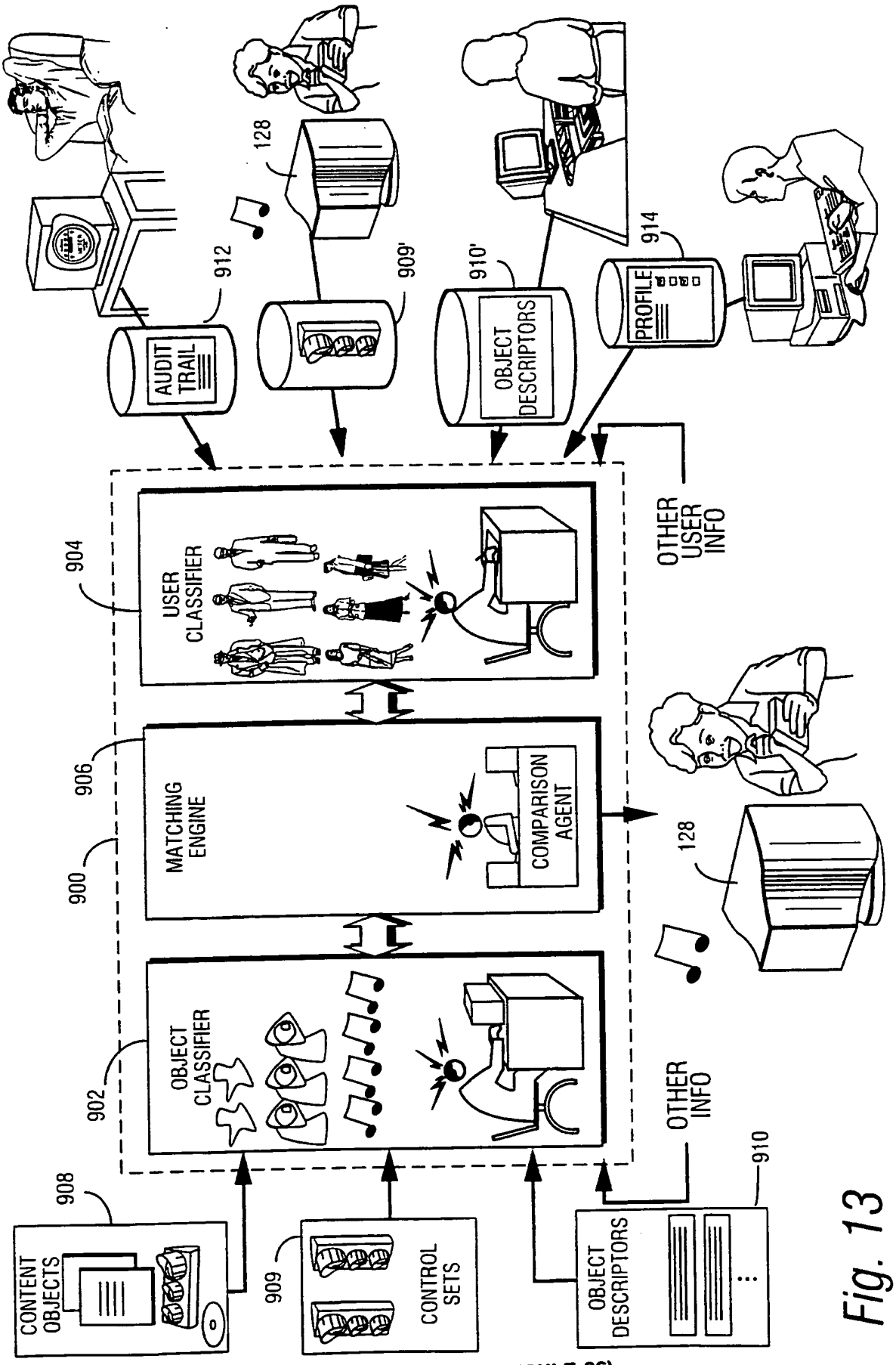


Fig. 13

Petitioner Apple Inc. - Exhibit 1004, p. 5679

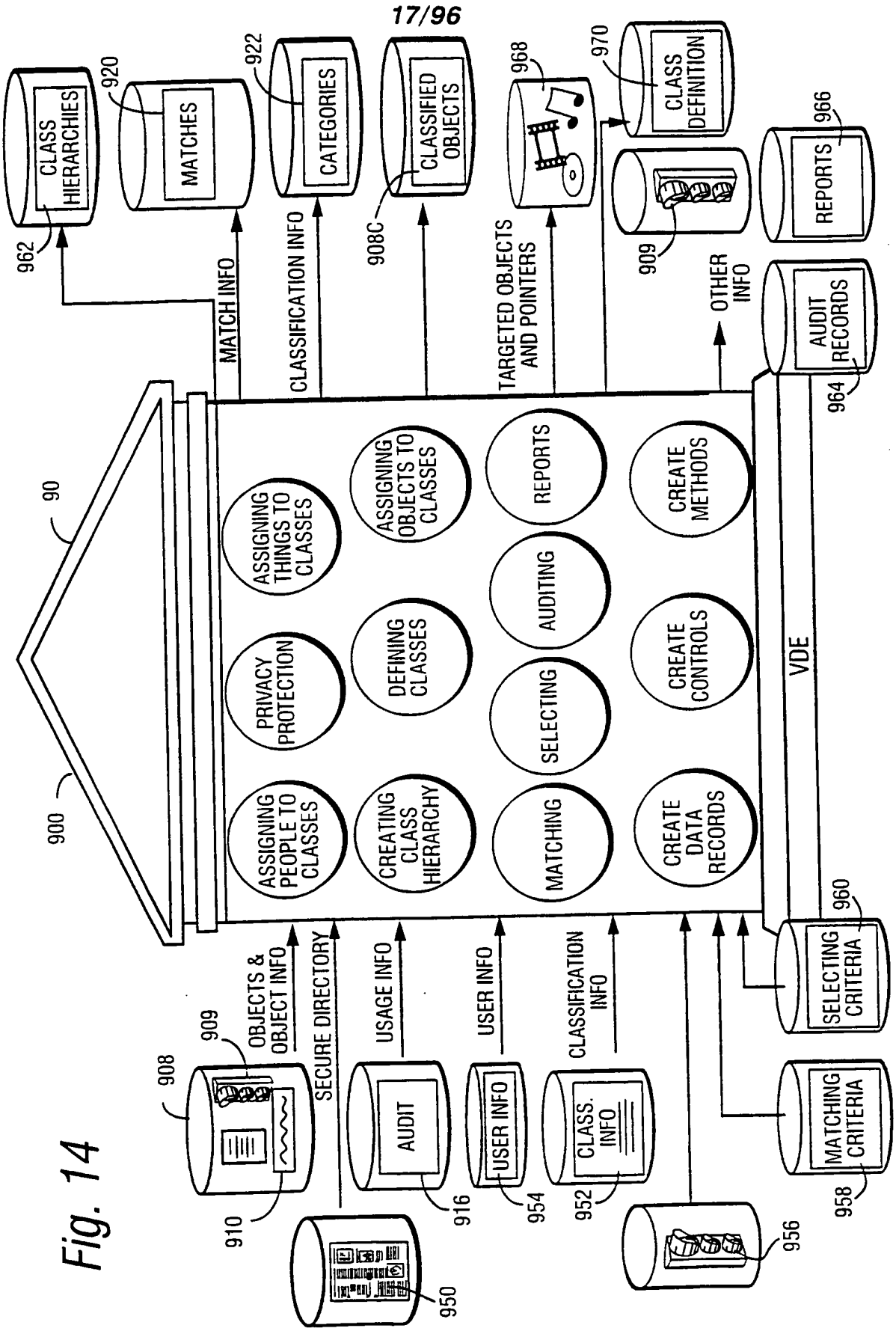
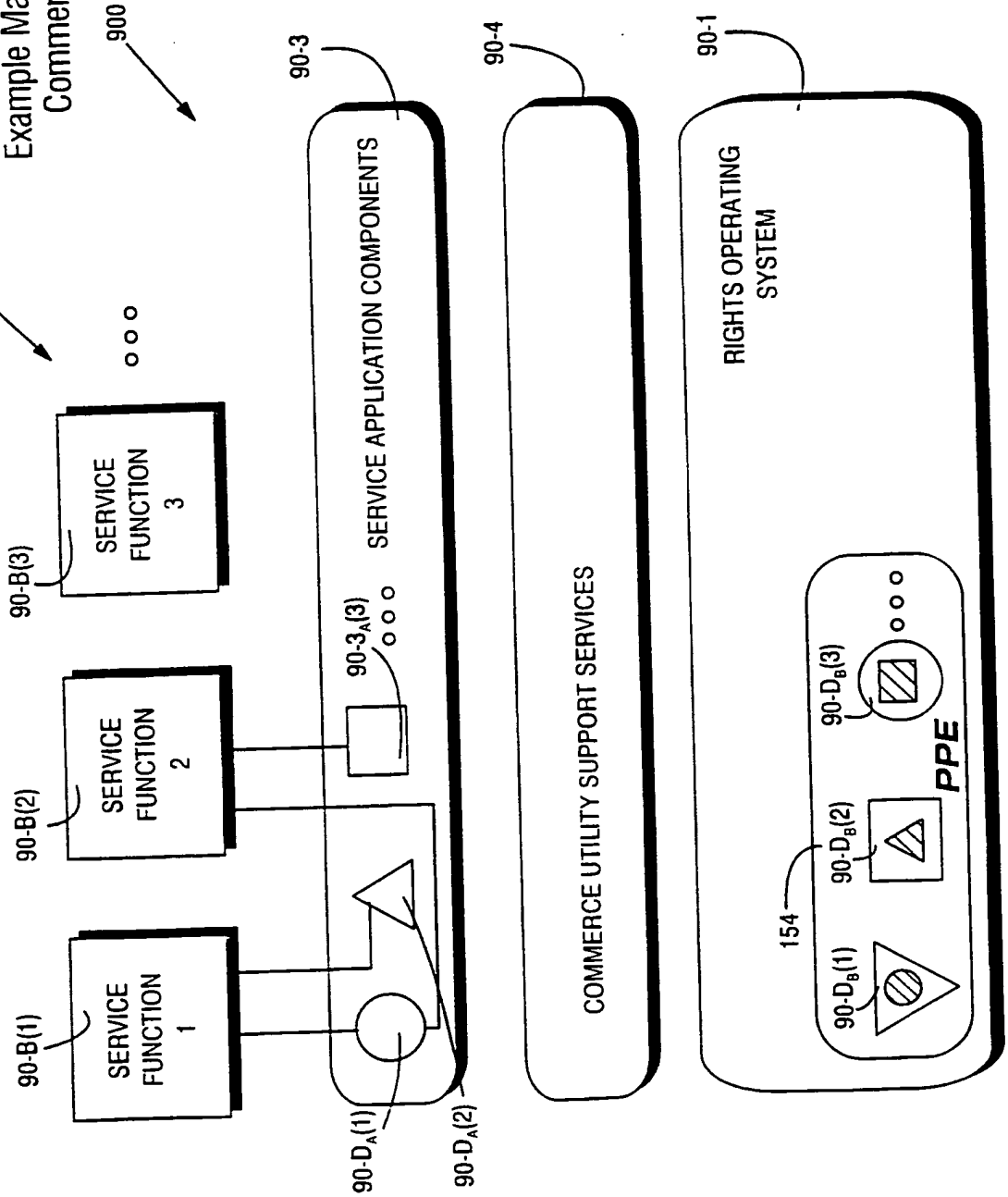


Fig. 14

Petitioner Apple Inc. - Exhibit 1004, p. 5680

Fig. 14(A)

Example Matching and Classification Commerce Utility System 900



Petitioner Apple Inc. - Exhibit 1004, p. 5681

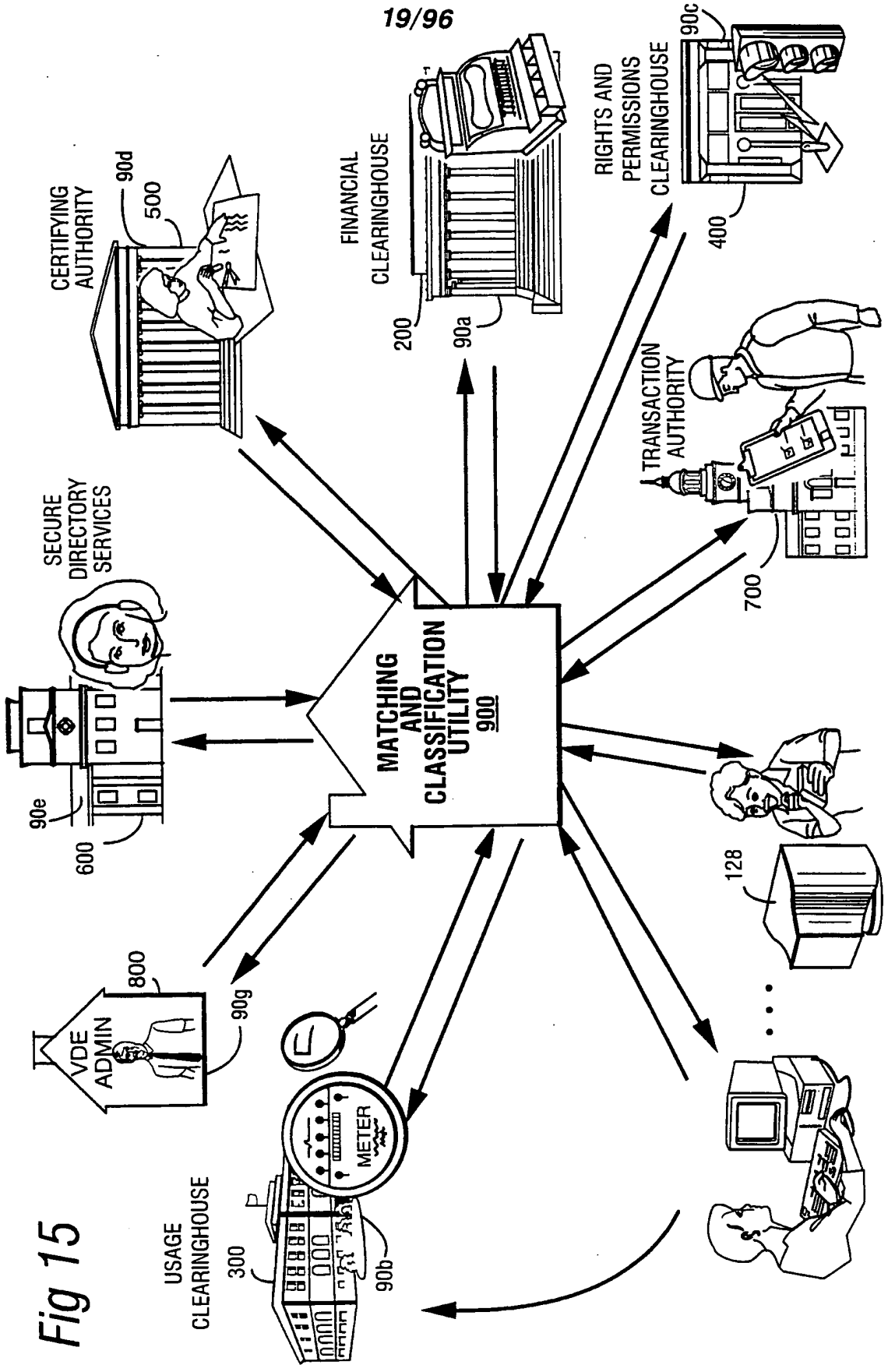


Fig 15

Petitioner Apple Inc. - Exhibit 1004, p. 5682

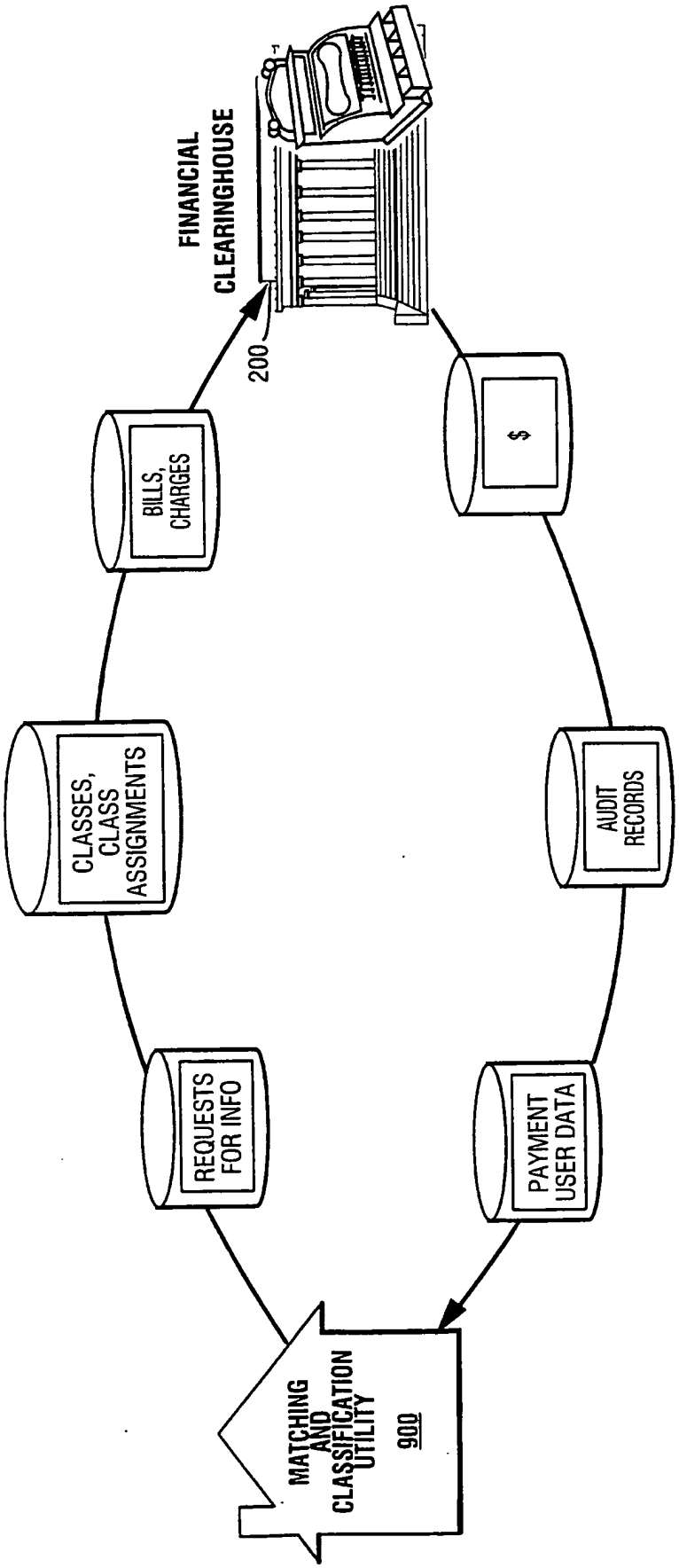


Fig. 15A

Petitioner Apple Inc. - Exhibit 1004, p. 5683

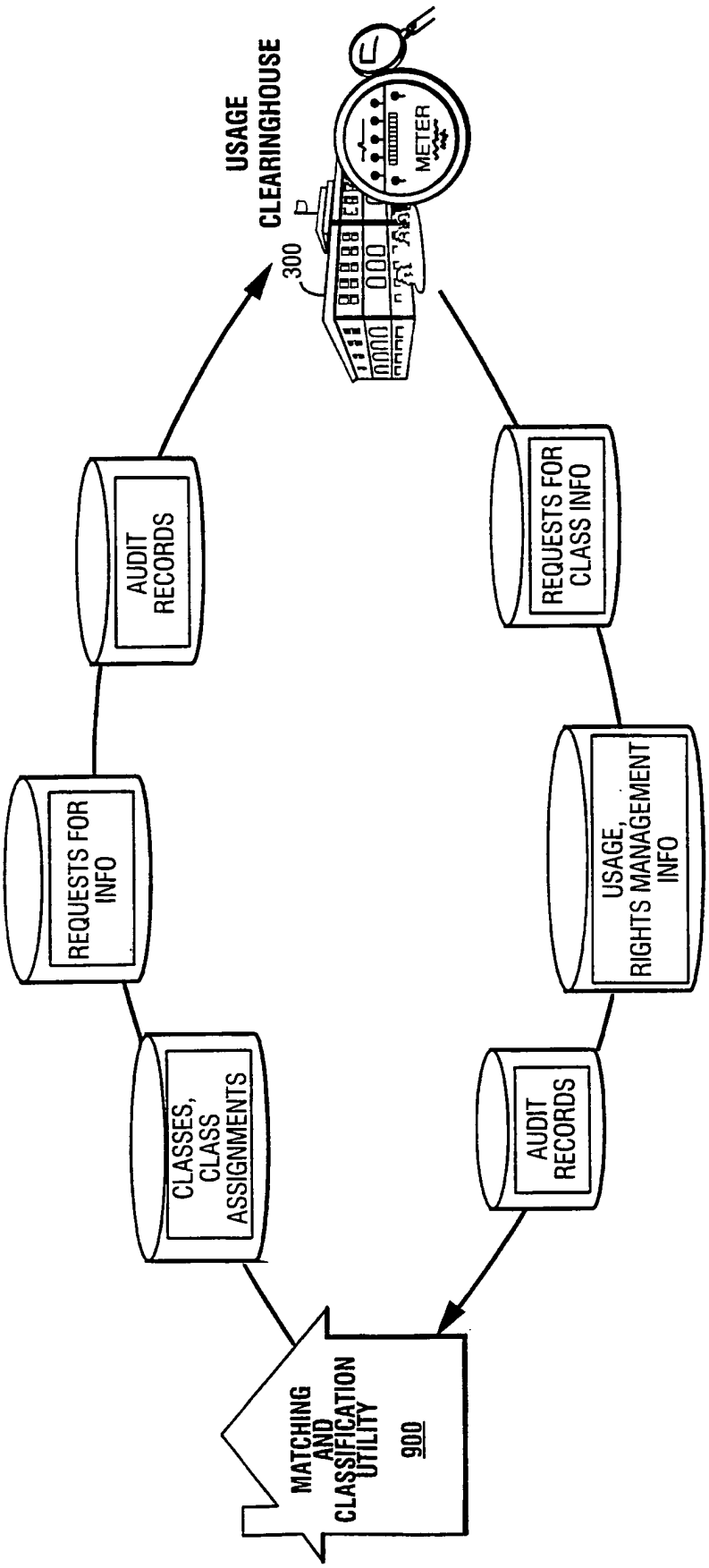


Fig. 15B

Petitioner Apple Inc. - Exhibit 1004, p. 5684

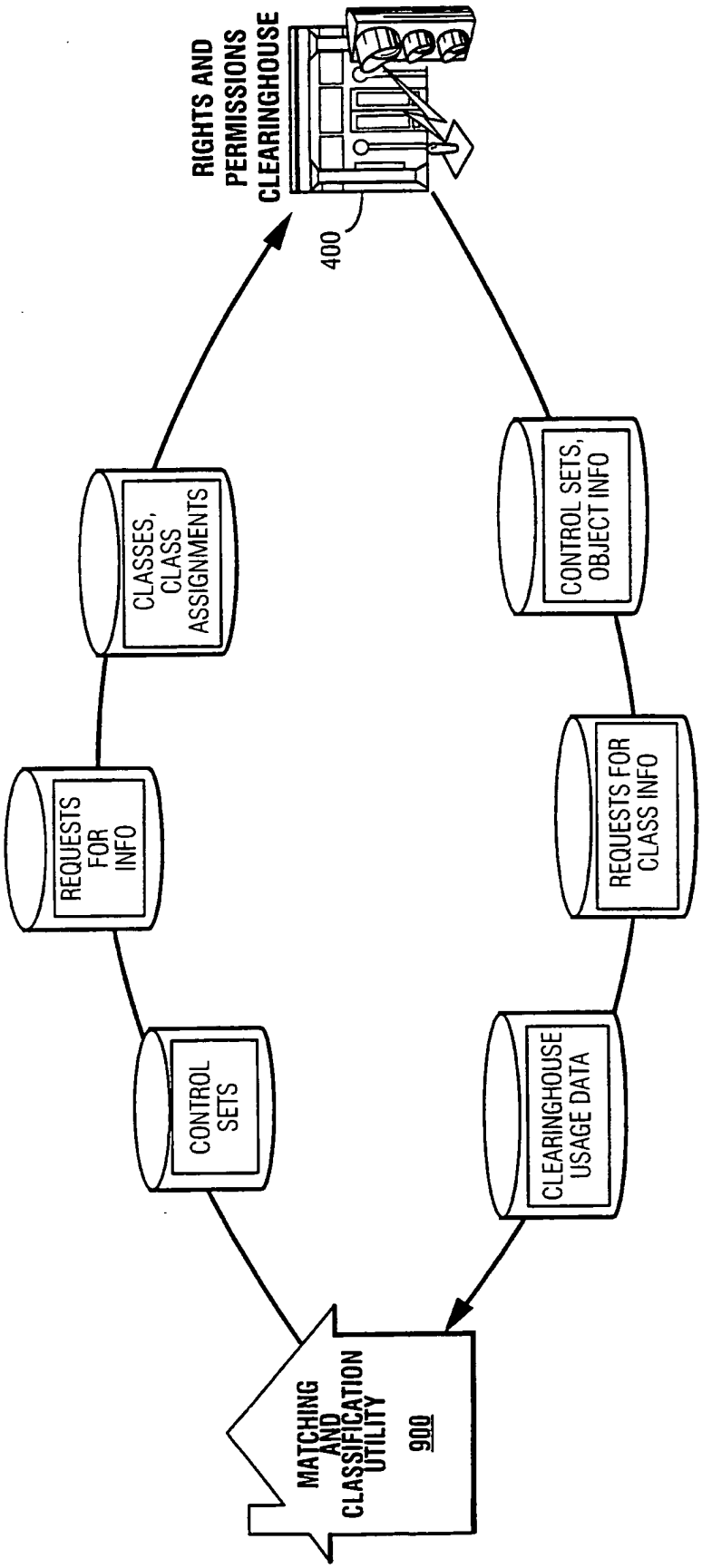


Fig. 15C

Petitioner Apple Inc. - Exhibit 1004, p. 5685

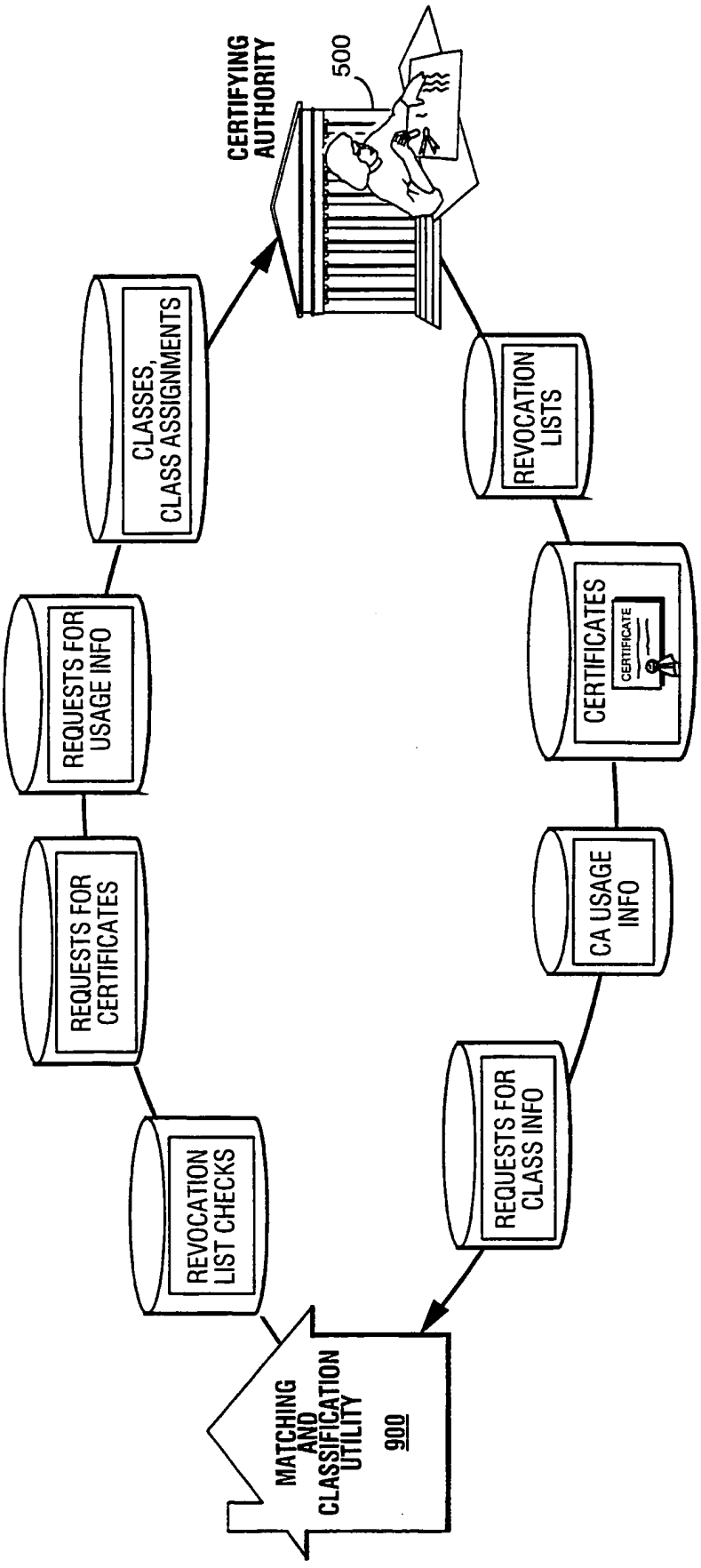


Fig. 15D

Petitioner Apple Inc. - Exhibit 1004, p. 5686

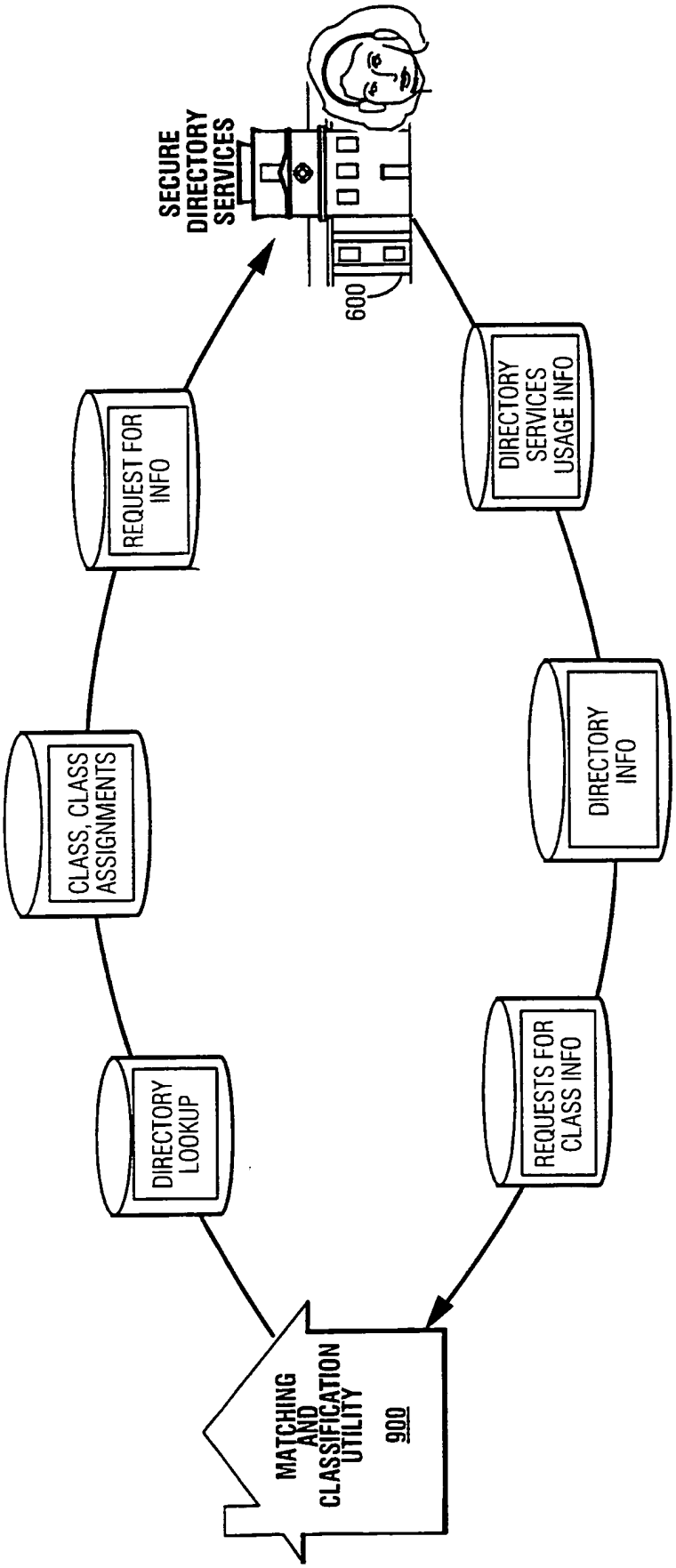


Fig. 15E

Petitioner Apple Inc. - Exhibit 1004, p. 5687

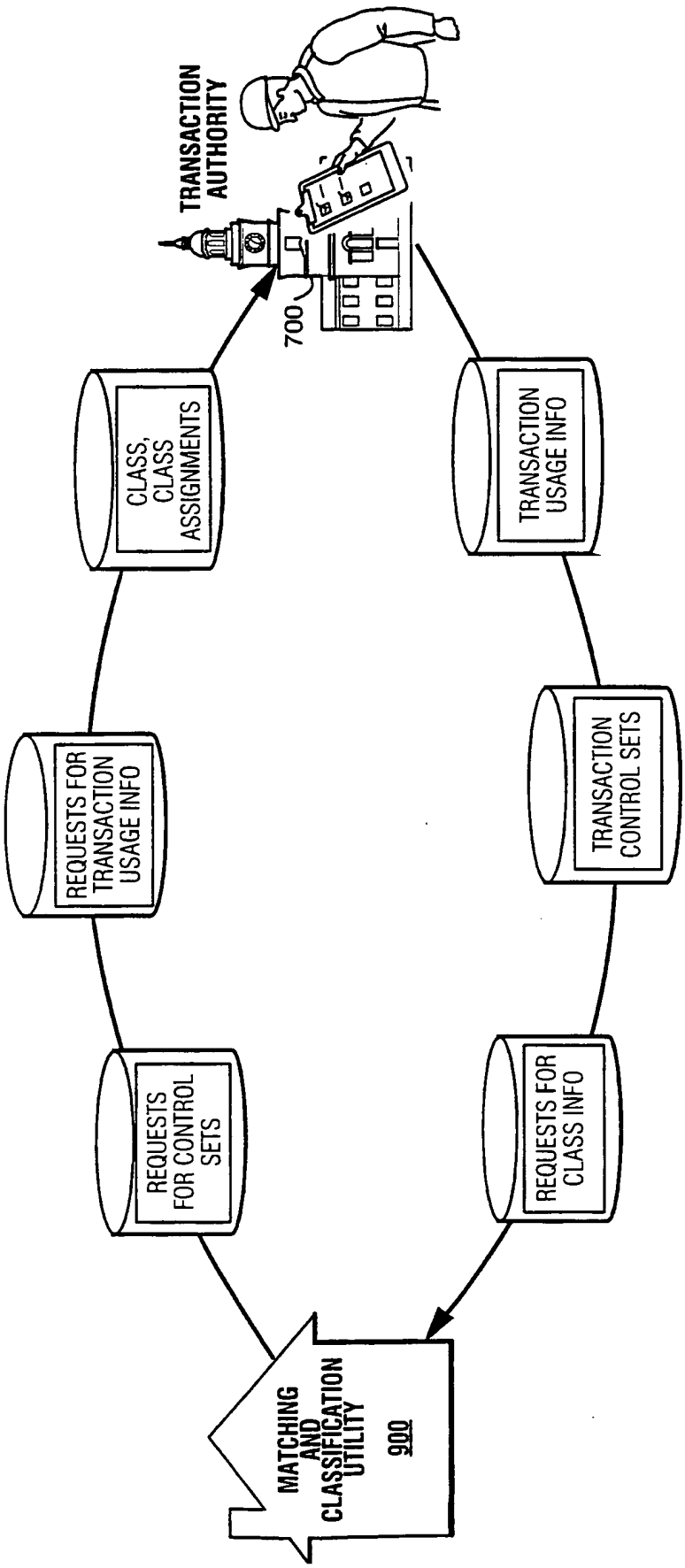


Fig. 15F

Petitioner Apple Inc. - Exhibit 1004, p. 5688

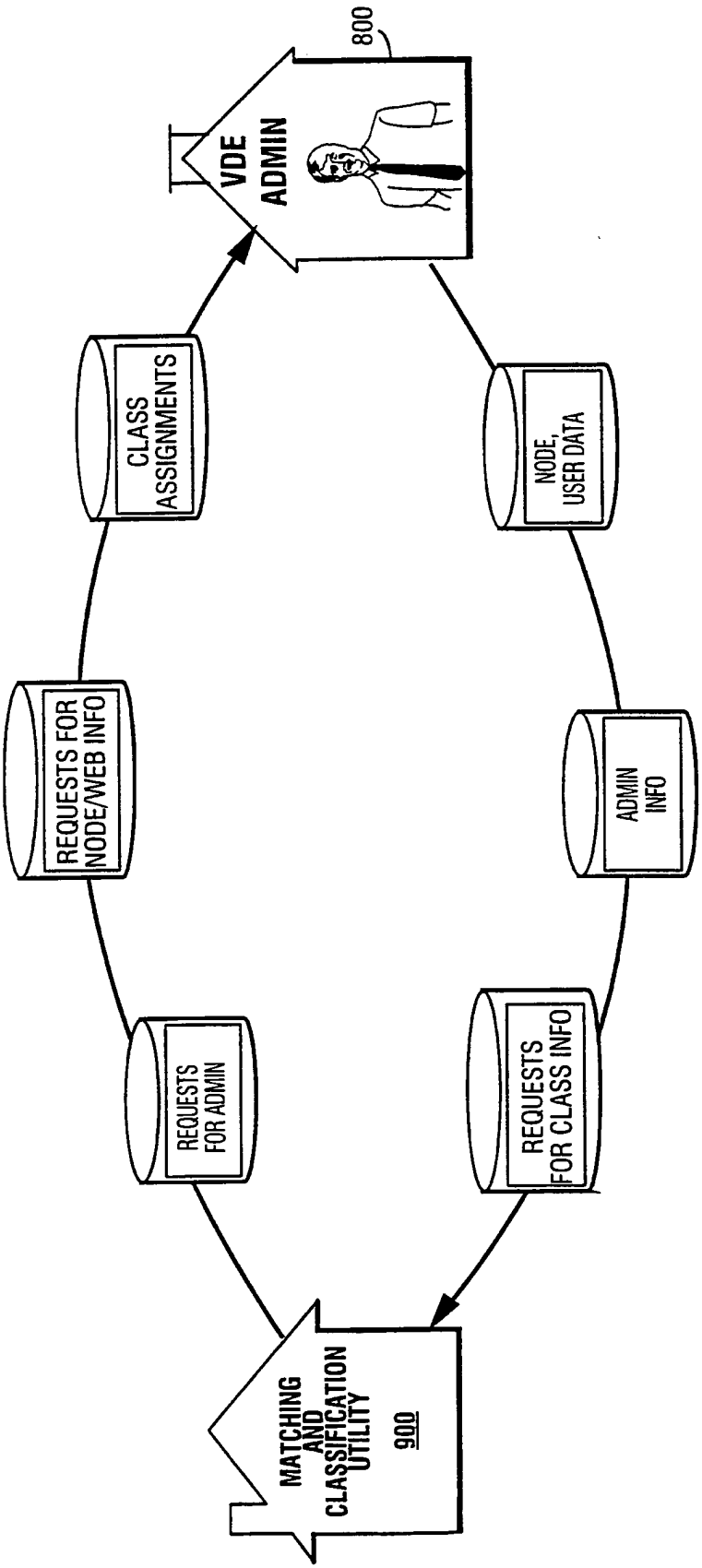


Fig. 15G

Petitioner Apple Inc. - Exhibit 1004, p. 5689

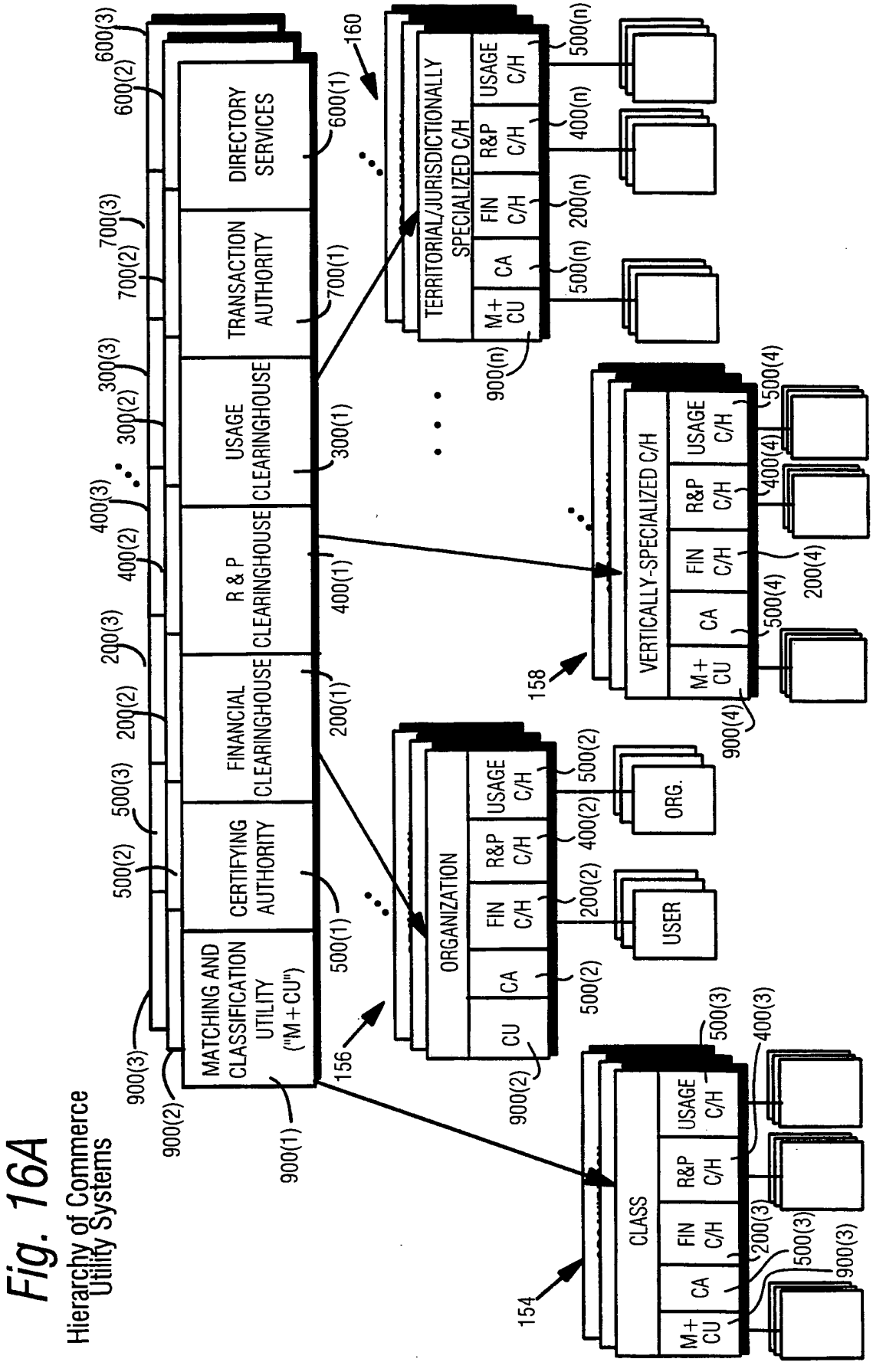


Fig. 16A
Hierarchy of Commerce Utility Systems

Petitioner Apple Inc. - Exhibit 1004, p. 5690

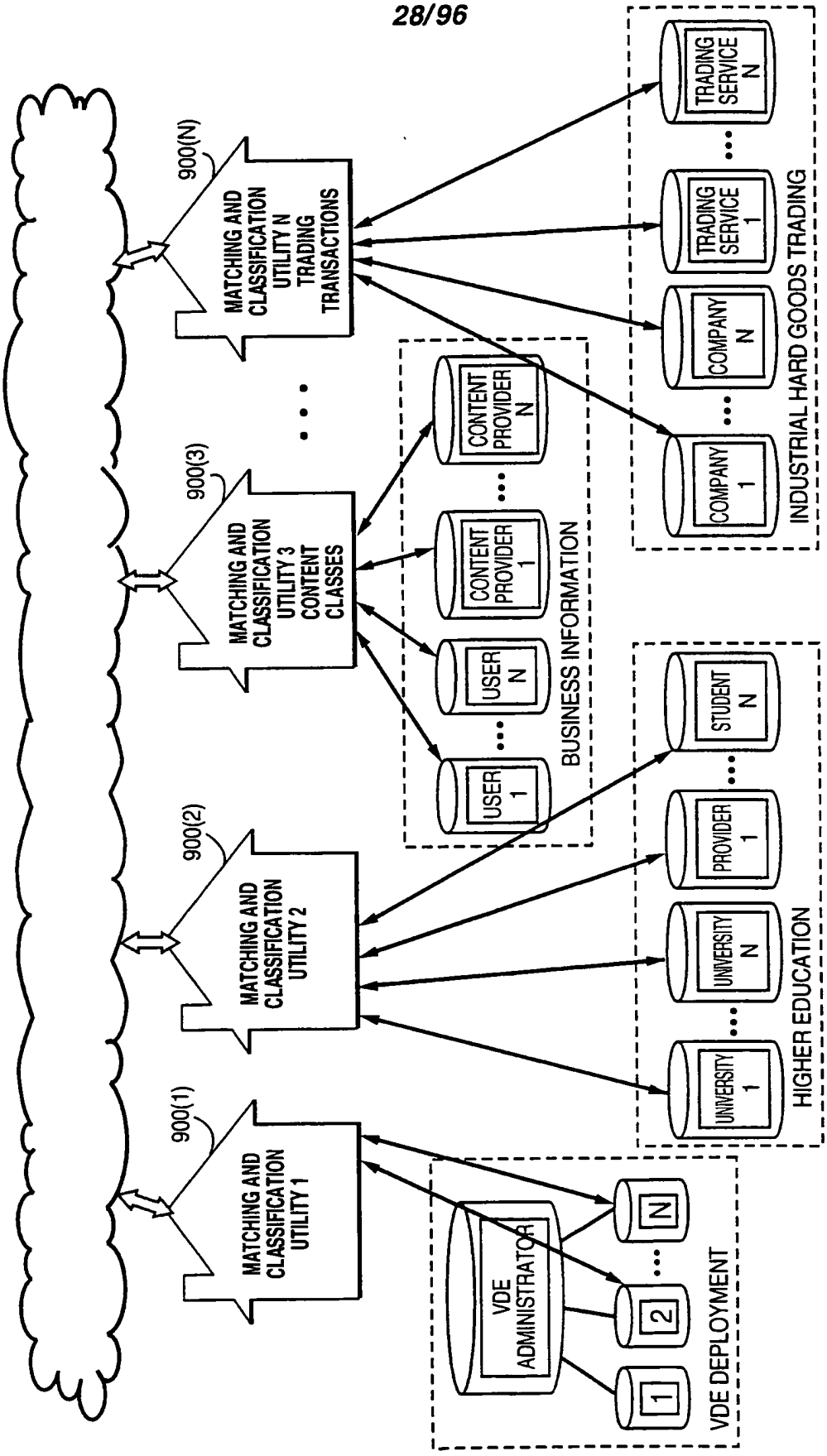


Fig.16B Matching & Classification Utilities Provide Services To Classes Of Nodes, Users, Content Services, Transaction Services.

Petitioner Apple Inc. - Exhibit 1004, p. 5691

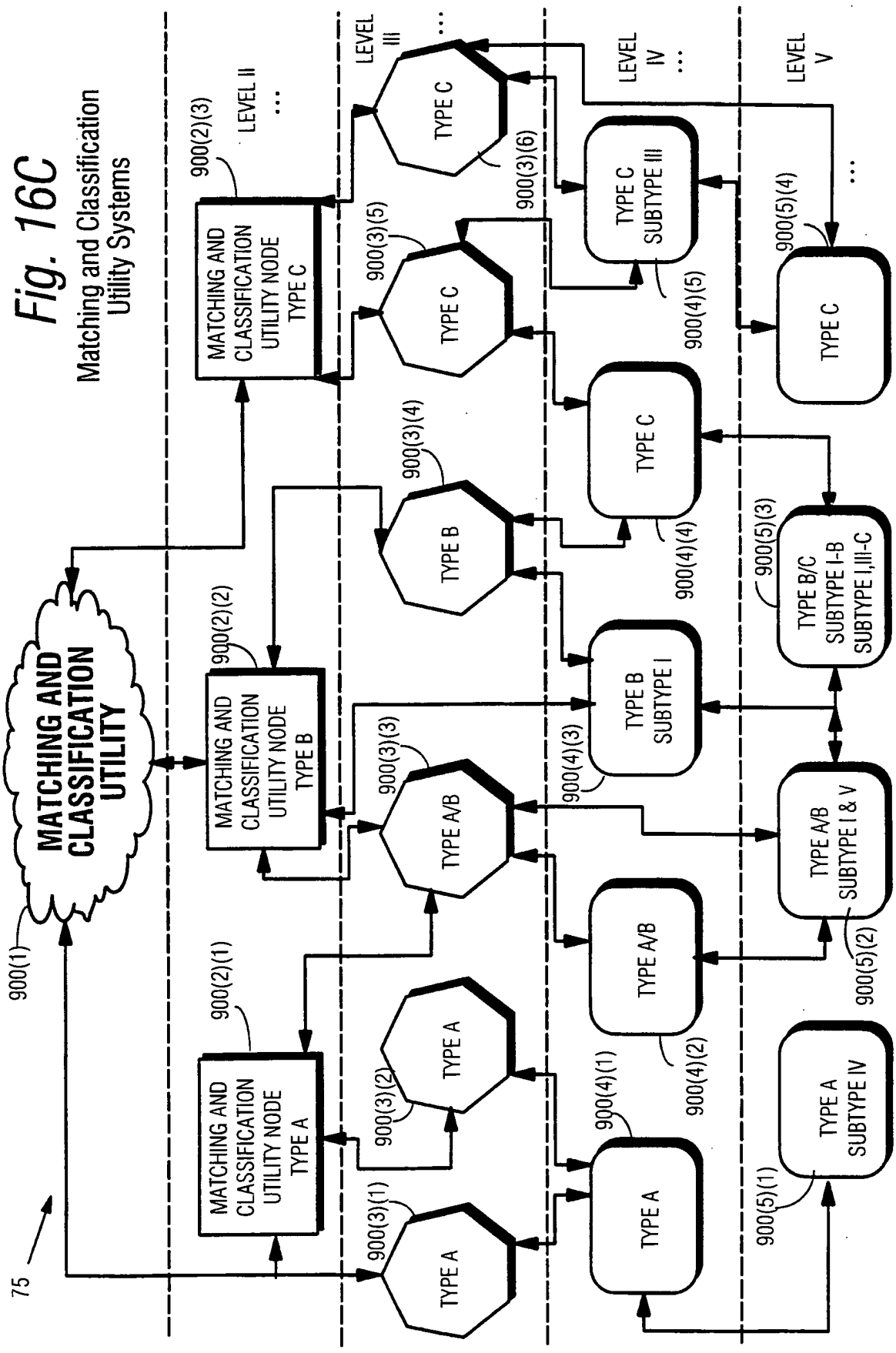
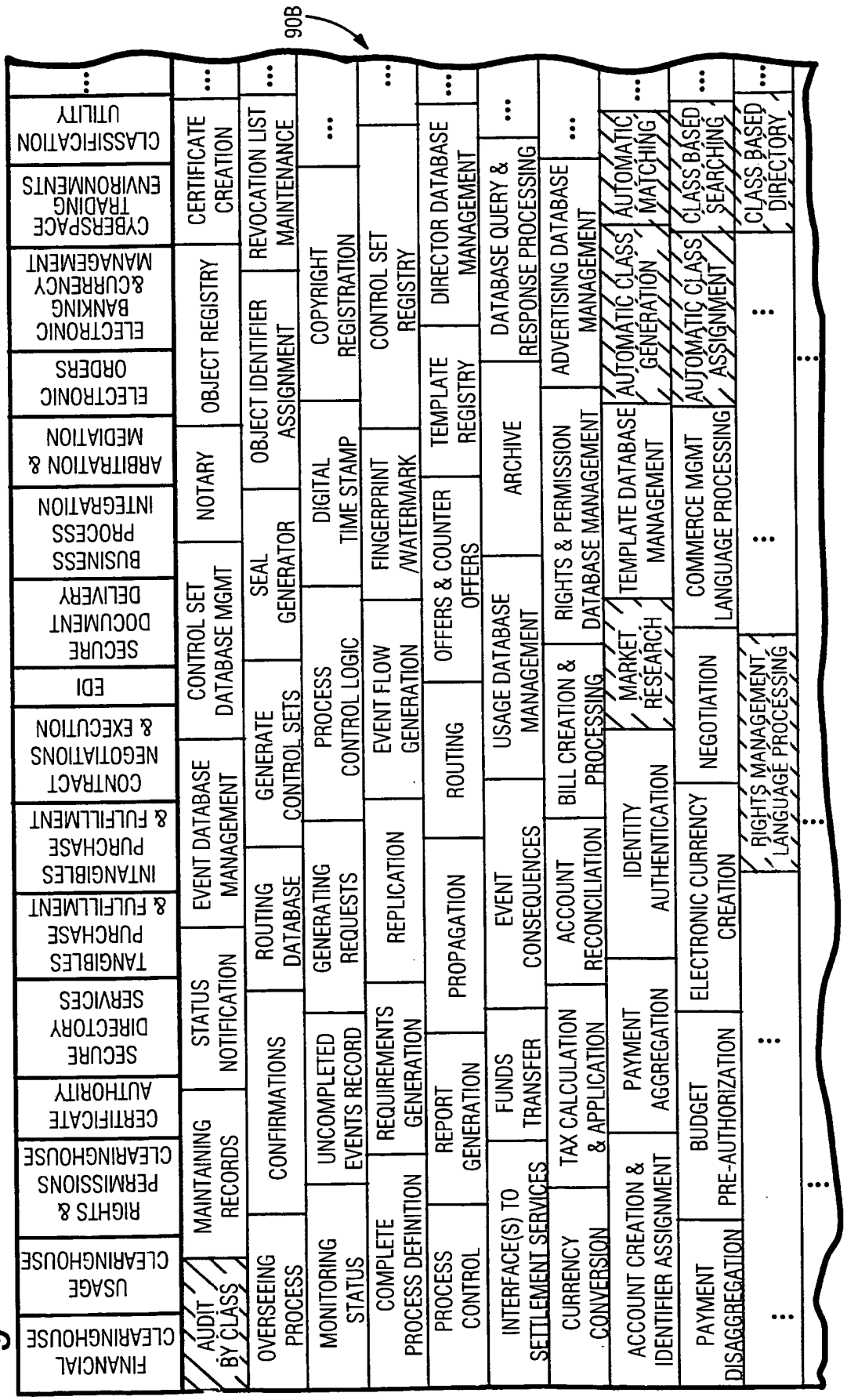


Fig. 16C
Matching and Classification Utility Systems

Petitioner Apple Inc. - Exhibit 1004, p. 5692

Fig. 17



Petitioner Apple Inc. - Exhibit 1004, p. 5693

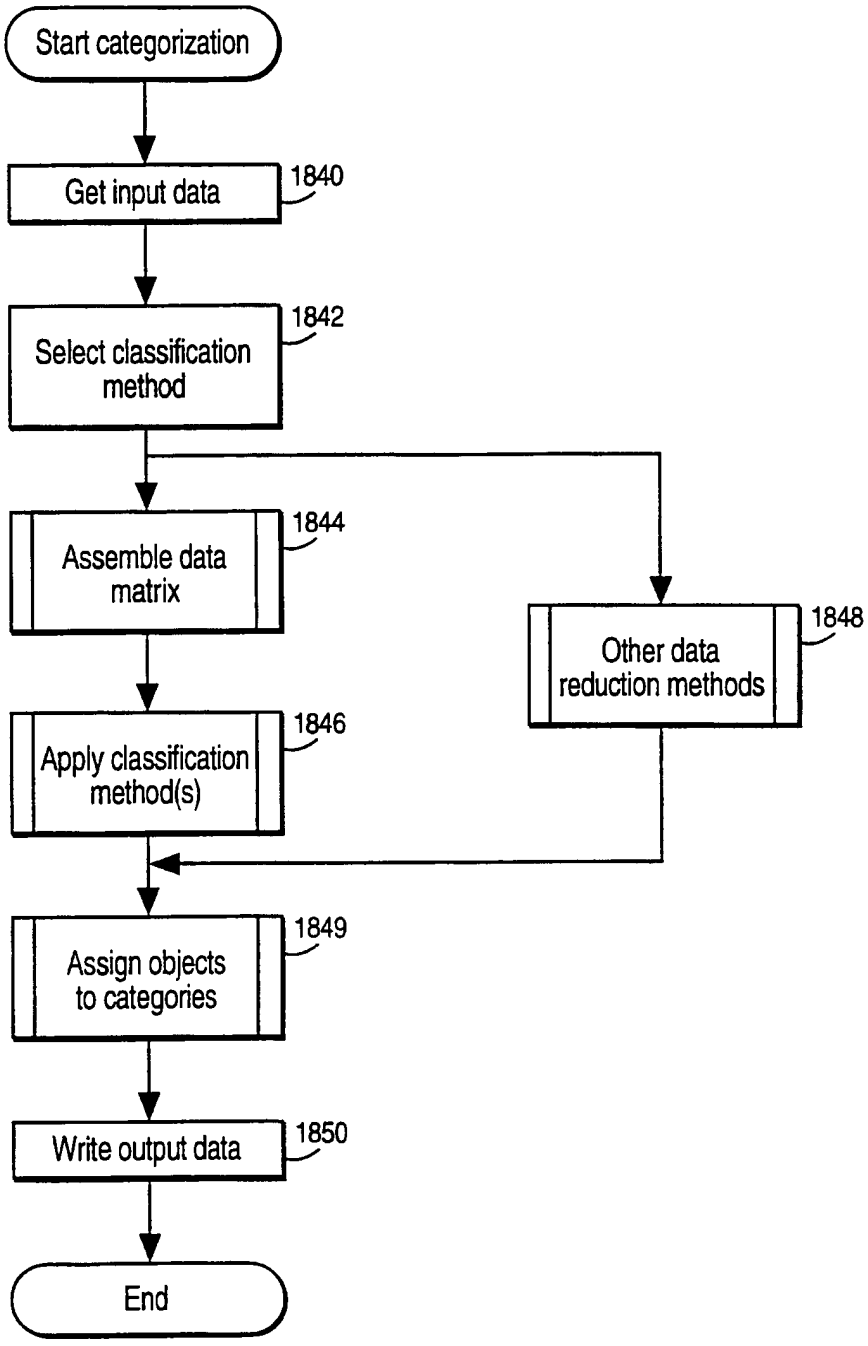


Fig. 18

Example Steps to Categorize Objects

Petitioner Apple Inc. - Exhibit 1004, p. 5694

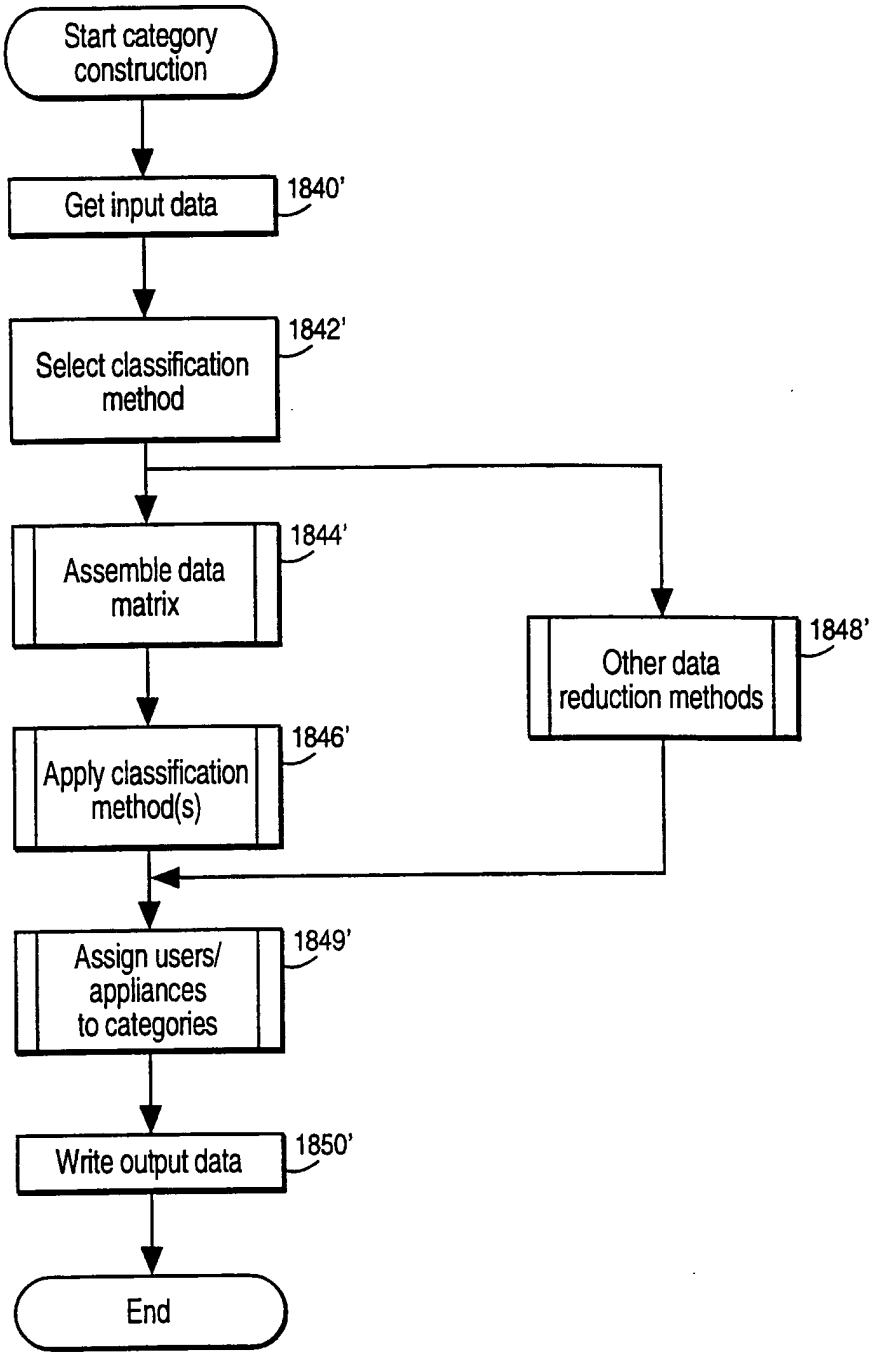


Fig. 19

Example Steps to Categorize Users/Appliances

Petitioner Apple Inc. - Exhibit 1004, p. 5695

Fig. 20
Example Composite Record-Input To Classification Process

Petitioner Apple Inc. - Exhibit 1004, p. 5696

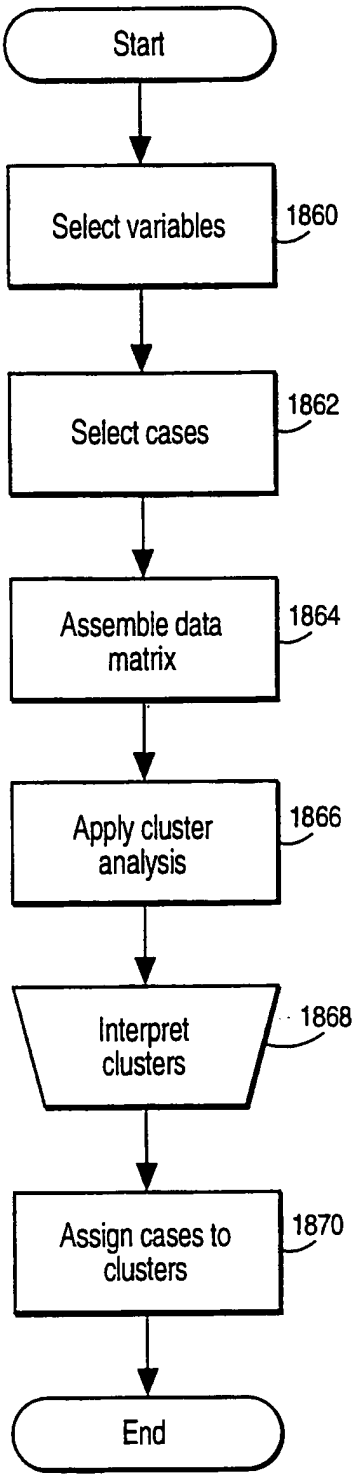


Fig. 21
Example Cluster Analysis Process

Petitioner Apple Inc. - Exhibit 1004, p. 5697

Variables	Typical Class 1-Profile	Typical Class 2-Profile
City	Washington, DC	Knoxville, TN
Av. price of content purchased last 30 days	$8.79	$1.95
Number of trips abroad in last 2 years	3	0
Type of content most frequently purchased	National and international news	Sports
2nd most frequently purchased	Business information	Religious
Third most frequently purchased	Travel information	Movies
Pay per view	No	Yes
Add new controls to content	Yes	No
Stated religious affiliation	None	Methodist
SRI internet lifestyle category	Surfer	Worker
Modification rights purchased	20% of text items	5% of text items

Fig. 22 Example Classification Output Illustrating Different Classes Based Upon Differing Profiles

Petitioner Apple Inc. - Exhibit 1004, p. 5698

Variables	Factor 1 Loadings	Factor 2 Loadings
Region of US	.82	.11
Family income	.90	-.09
Av. price of content purchased last 30 days	.72	.15
Number of trips abroad in last 2 years	.91	.09
Percent news, business	.79	-.12
Percent entertainment	-.69	.21
Add new controls to content	.88	.19
Religiosity	-.60	-.22
Participates in sports	-.21	.87
Watches team/individual sports on TV	-.11	.62
Owns a sports utility vehicle	.12	.72
Consumes beer/wine	-.18	.83
Male/female	.21	.92
Education beyond college	.45	-.45
Buys pay per view sports events	-.25	.77
Number of TVs in house	-.11	.66

Fig. 23 Example Classification Output Illustrating Principal Components Analysis On Parameter Data And Categories Data

Petitioner Apple Inc. - Exhibit 1004, p. 5699

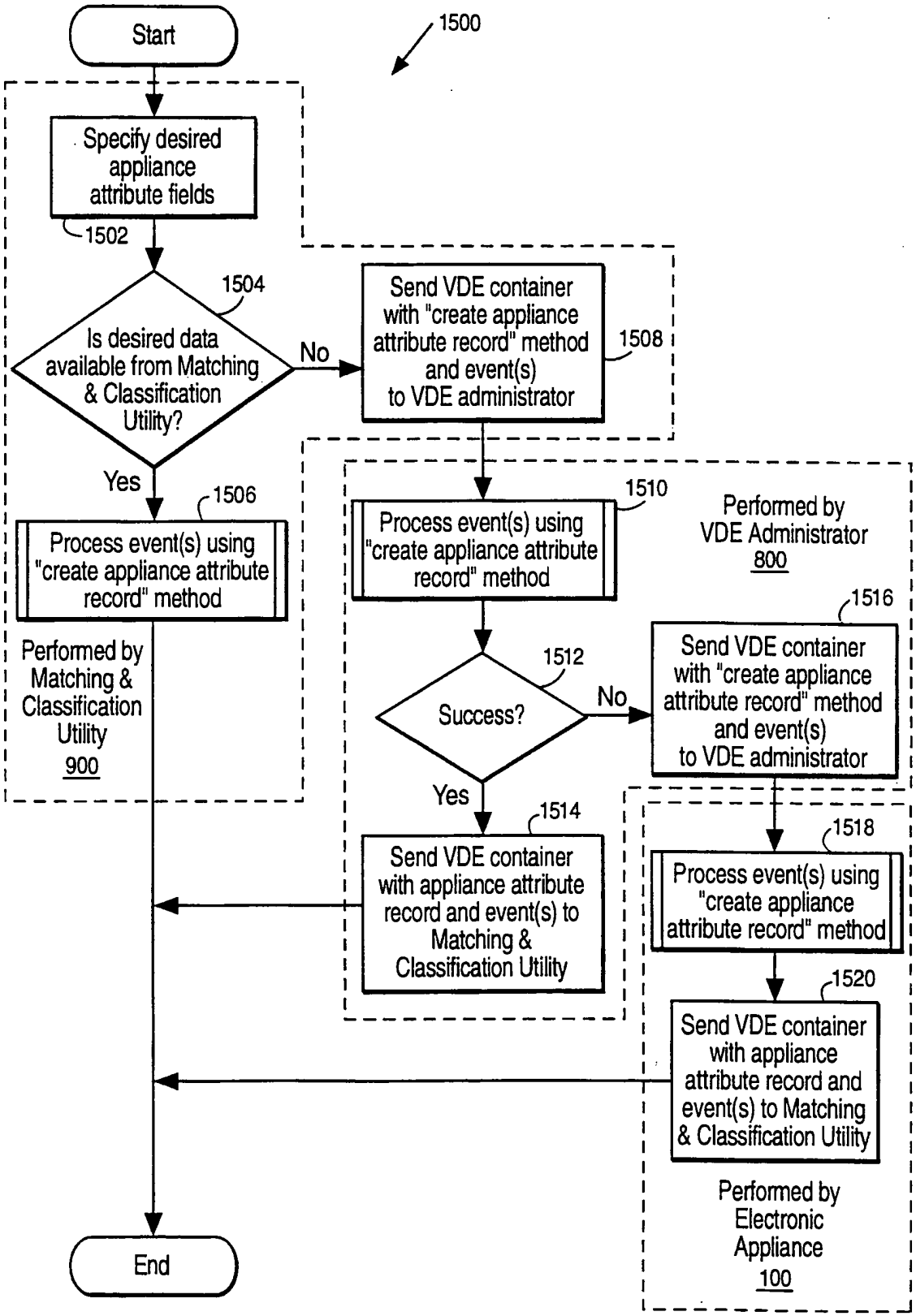


Fig. 24

Example Steps for Collecting Appliance Attribute Data

Petitioner Apple Inc. - Exhibit 1004, p. 5700

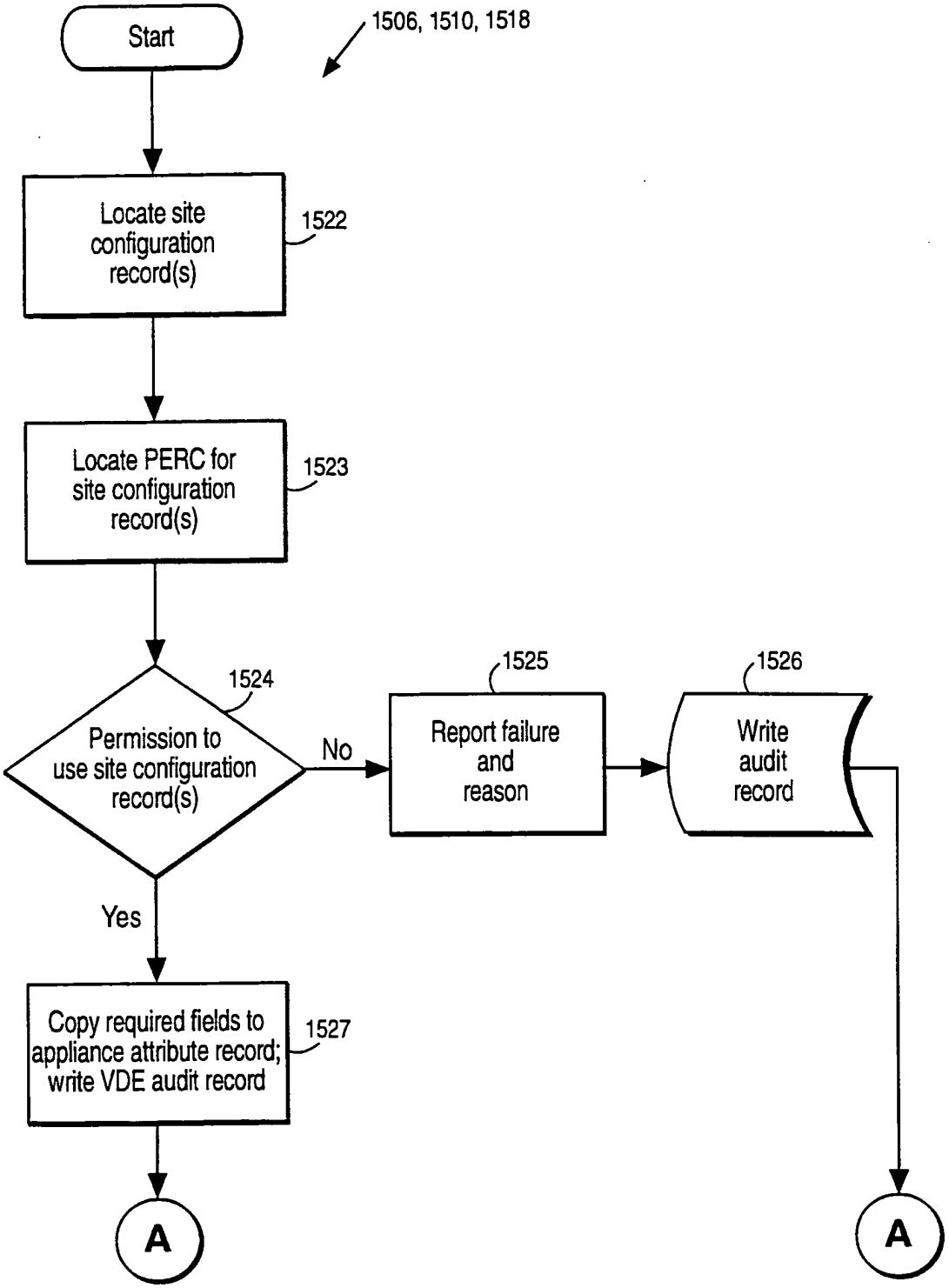
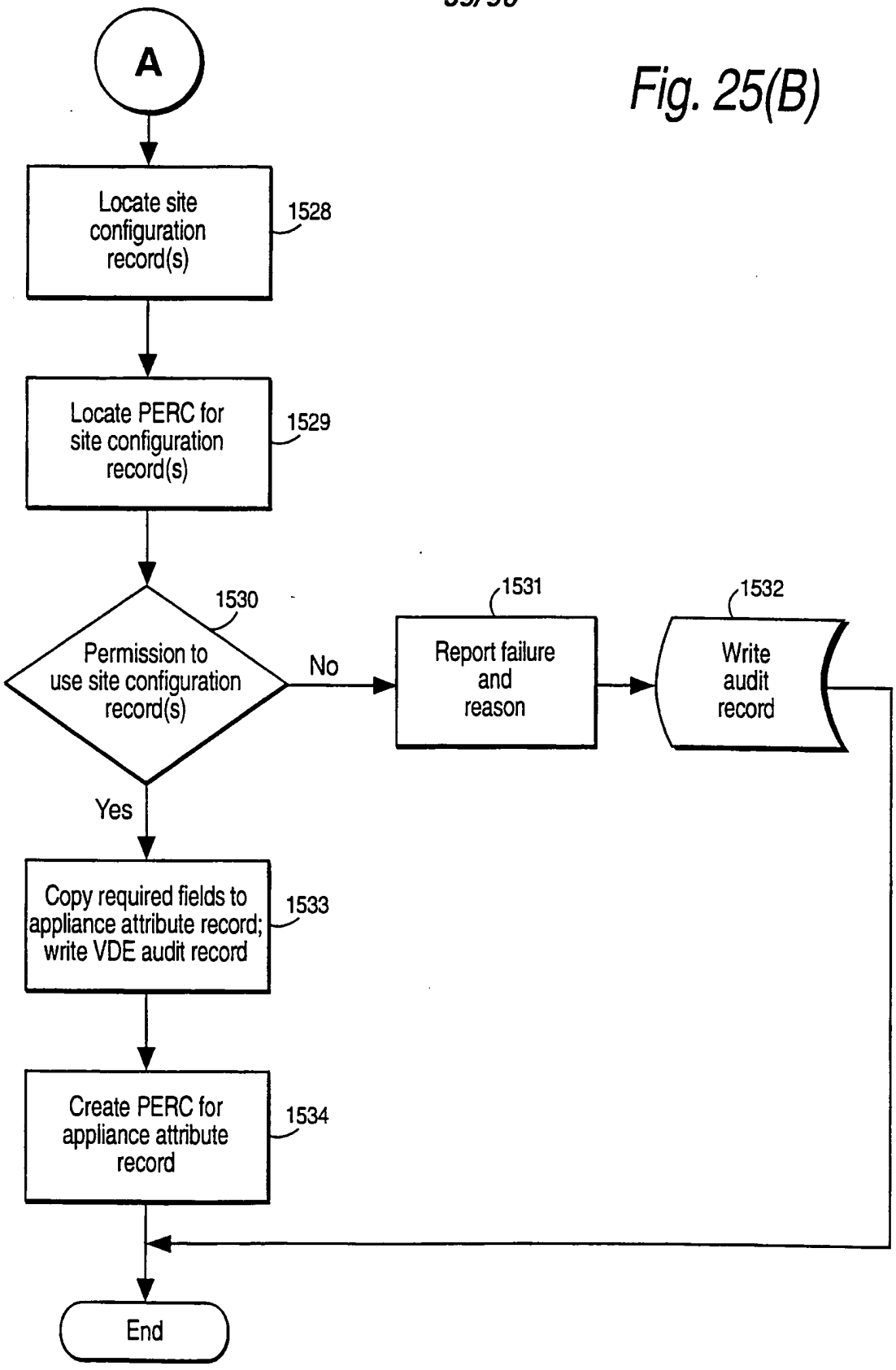


Fig. 25(A)

Example Create Appliance Attribute Data Method steps

Petitioner Apple Inc. - Exhibit 1004, p. 5701

Fig. 25(B)



Petitioner Apple Inc. - Exhibit 1004, p. 5702

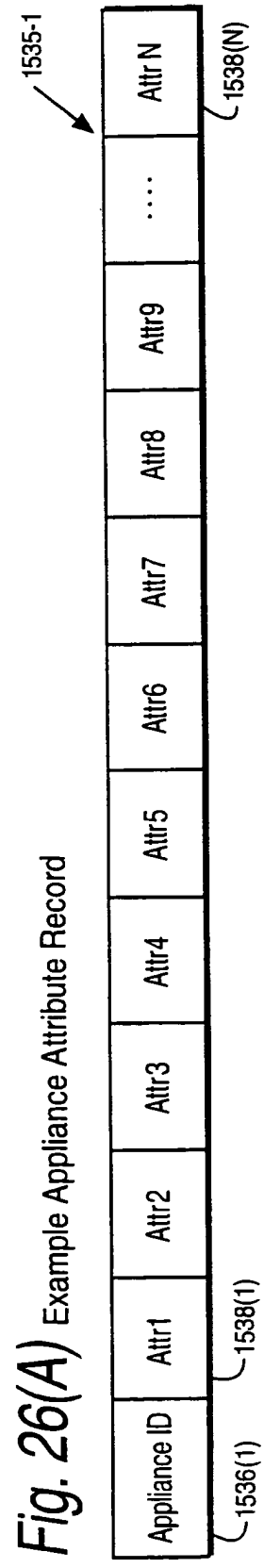


Fig. 26(A) Example Appliance Attribute Record

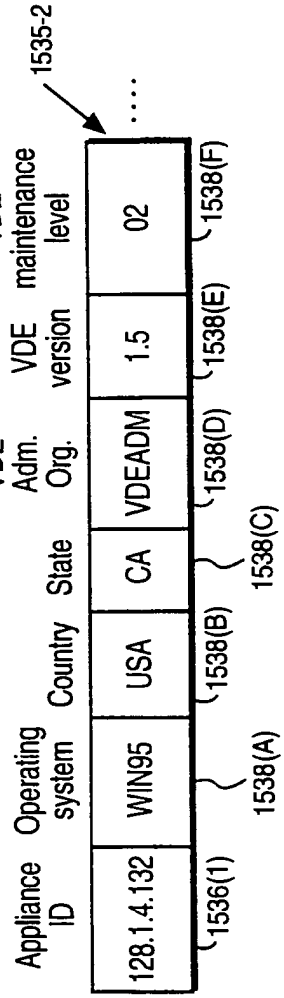


Fig. 26(B)



Fig. 26(C) Example Appliance Attribute Record

Petitioner Apple Inc. - Exhibit 1004, p. 5703

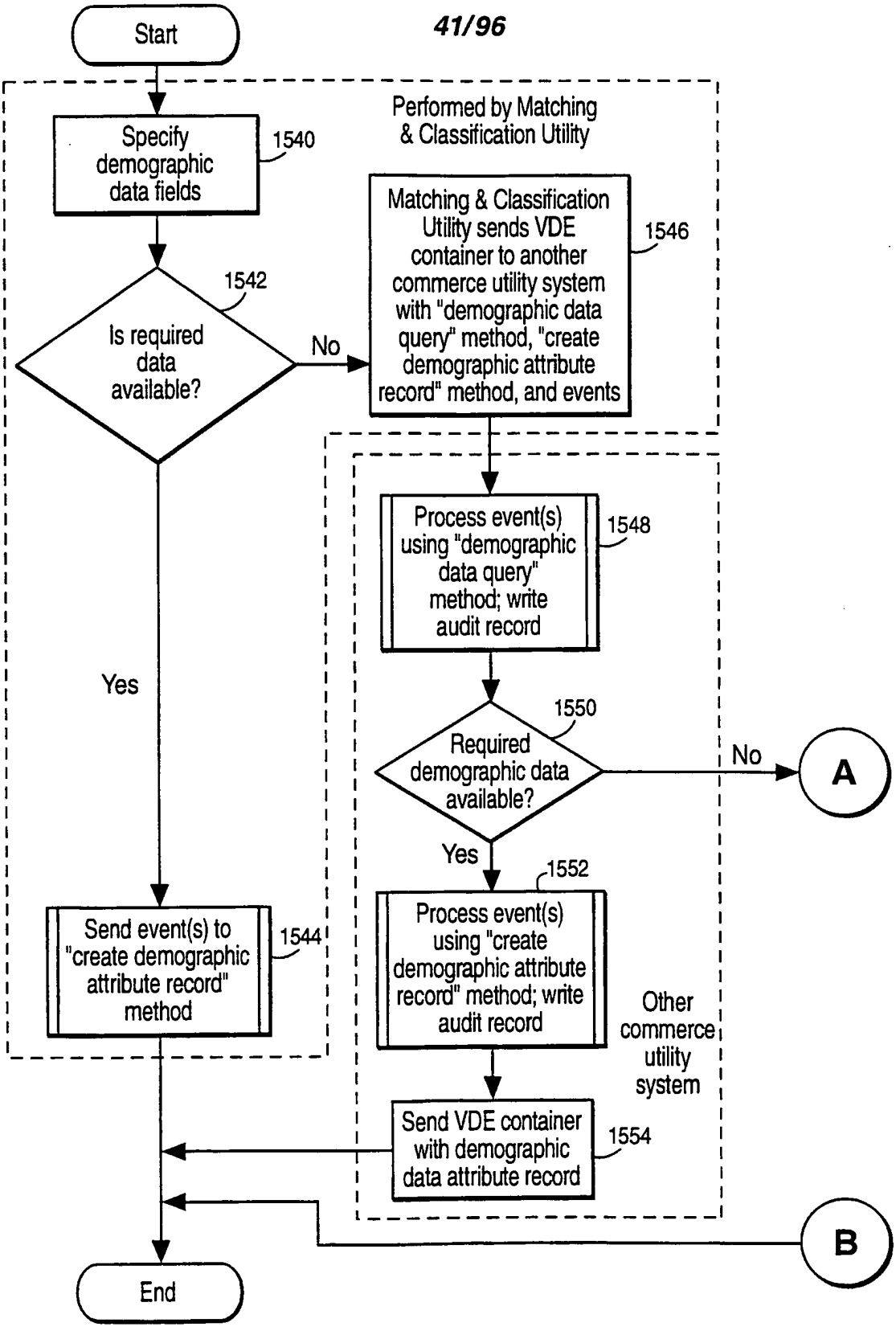


Fig. 27(A) Example Steps for Collecting Demographic Data

Petitioner Apple Inc. - Exhibit 1004, p. 5704

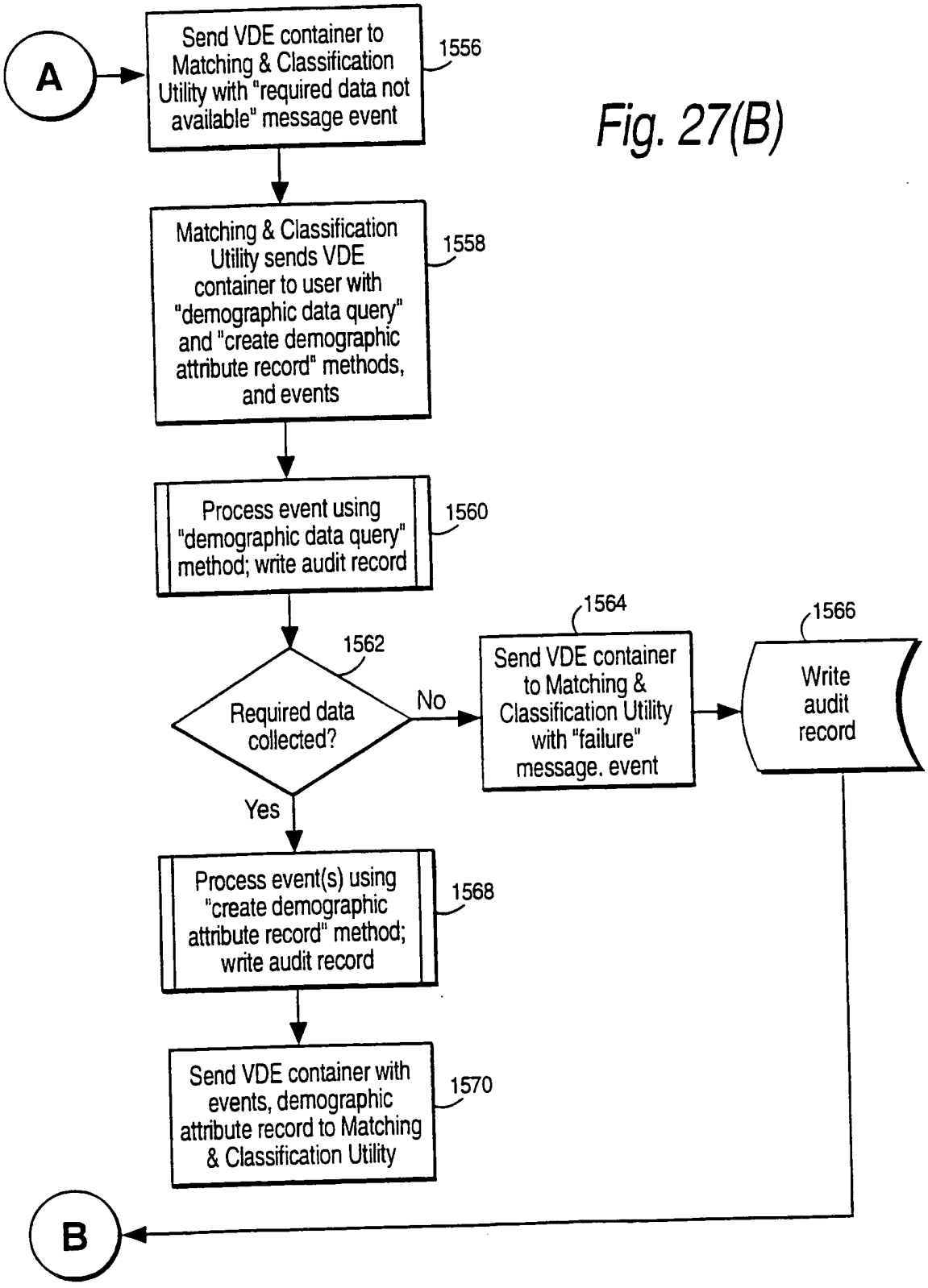


Fig. 27(B)

Petitioner Apple Inc. - Exhibit 1004, p. 5705

Demographic Information Questionnaire

Name: _____

Address: _____

Address: _____

City: _____ State: _____ Zip: _____ - _____

Gender (M/F) _____ Date of birth: _____ /_____ /_____

Education:

- ☐ Have not graduated high school
- ☐ High school graduate
- ☐ Some college
- ☐ College degree
- ☐ Some graduate school
- ☐ Advanced degree

All Information Will Be Treated As Confidential

Fig. 28 Example Demographic Questionnaire "Pop-Up" Screen

Petitioner Apple Inc. - Exhibit 1004, p. 5706

Fig. 29(A) Example User Demographic Attribute Information Record

Fig. 29(B) Example Demographic Attribute Record

Fig. 29(C) Example Demographic Attribute Record

Petitioner Apple Inc. - Exhibit 1004, p. 5707

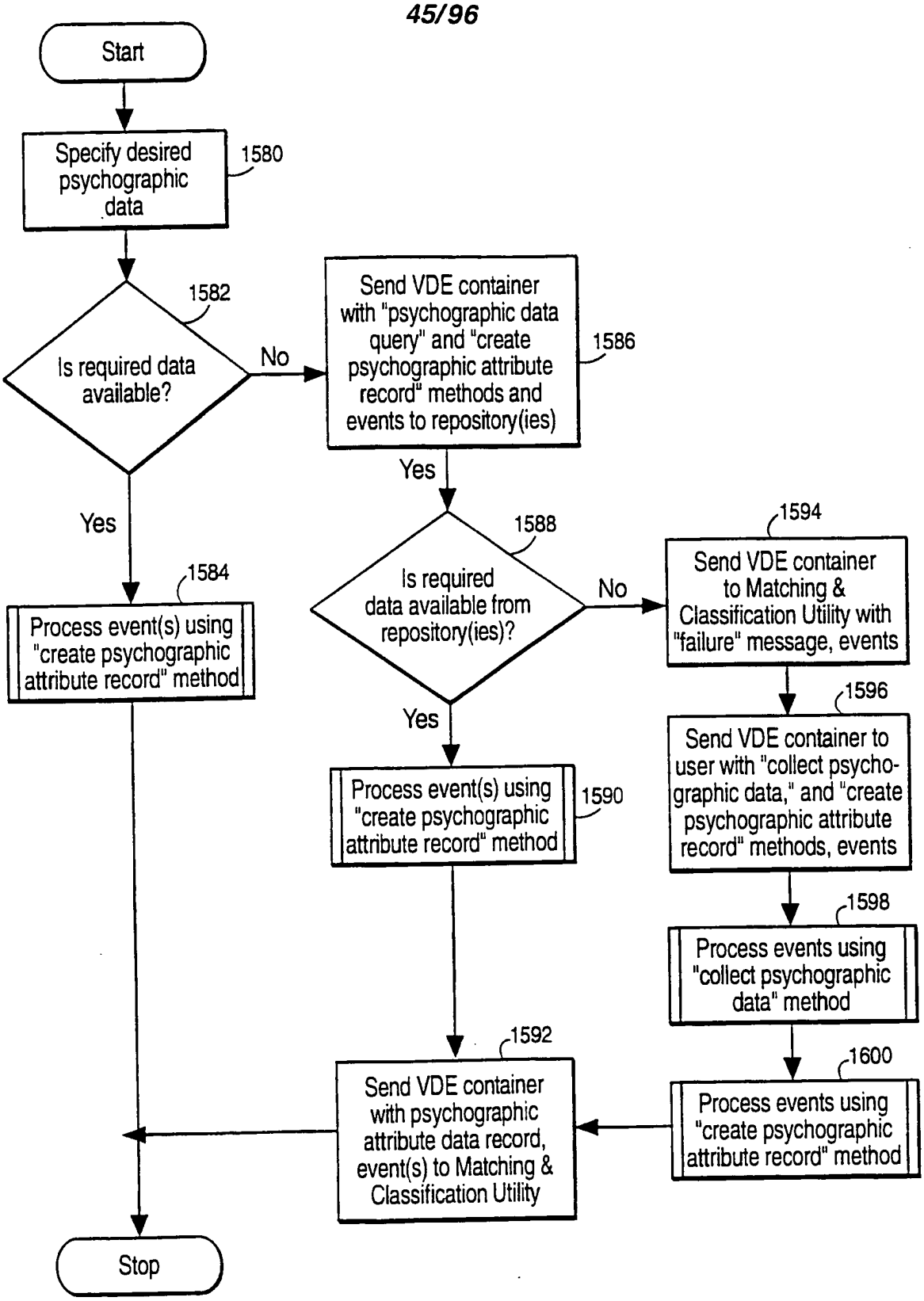


Fig. 30 Example Steps for Collecting Psychographic Data

Petitioner Apple Inc. - Exhibit 1004, p. 5708

Today's Anonymous Questionnaire
Thanks for taking the time to answer these questions
We'll put $2.00 in your VDE budget

1. Do you feel sad, blue, unhappy or "down in the dumps"?

☐ A. Never
☐ B. Rarely
☐ C. Sometimes
☐ D. Very Often
☐ E. Most of the time

2. Do you feel tired, having little energy, unable to concentrate?

☐ A. Never
☐ B. Rarely
☐ C. Sometimes
☐ D. Very Often
☐ E. Most of the time

3. Do you feel uneasy, restless or irritable?

☐ A. Never
☐ B. Rarely
☐ C. Sometimes
☐ D. Very Often
☐ E. Most of the time

4. Do you have trouble sleeping or eating (too little or too much)?

☐ A. Never
☐ B. Rarely
☐ C. Sometimes
☐ D. Very Often
☐ E. Most of the time

☐ Click here for more questions

All Information Will Be Treated As Confidential

Fig. 31 Example Psychographic Questionnaire "Pop-Up" Screen

Petitioner Apple Inc. - Exhibit 1004, p. 5709

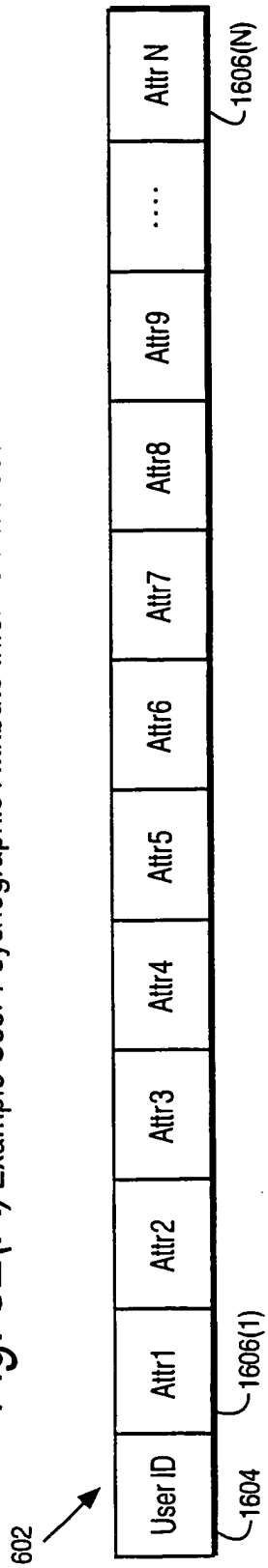


Fig. 32(A) Example User Psychographic Attribute Information Record

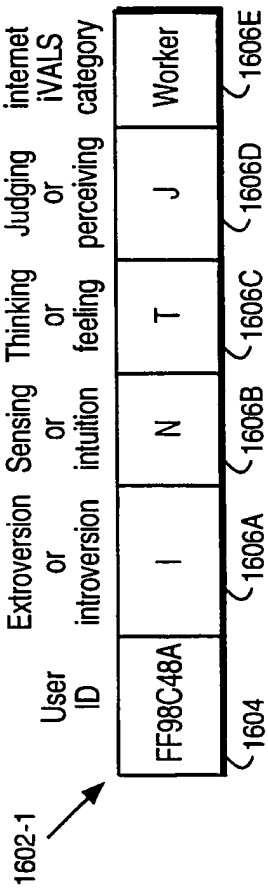


Fig. 32(B) Example User Psychographic Attribute Record

Petitioner Apple Inc. - Exhibit 1004, p. 5710

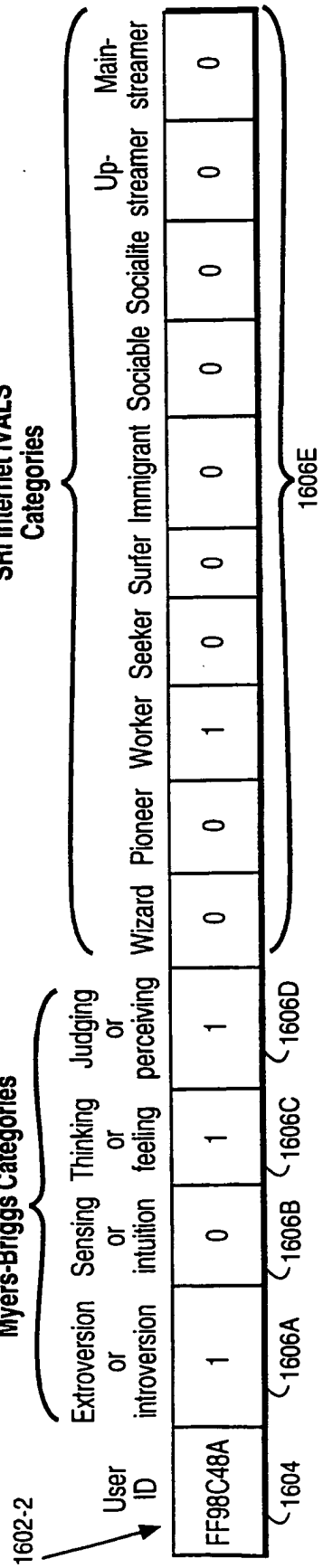


Fig. 32(C) Example Psychographic Attribute Record

Petitioner Apple Inc. - Exhibit 1004, p. 5711

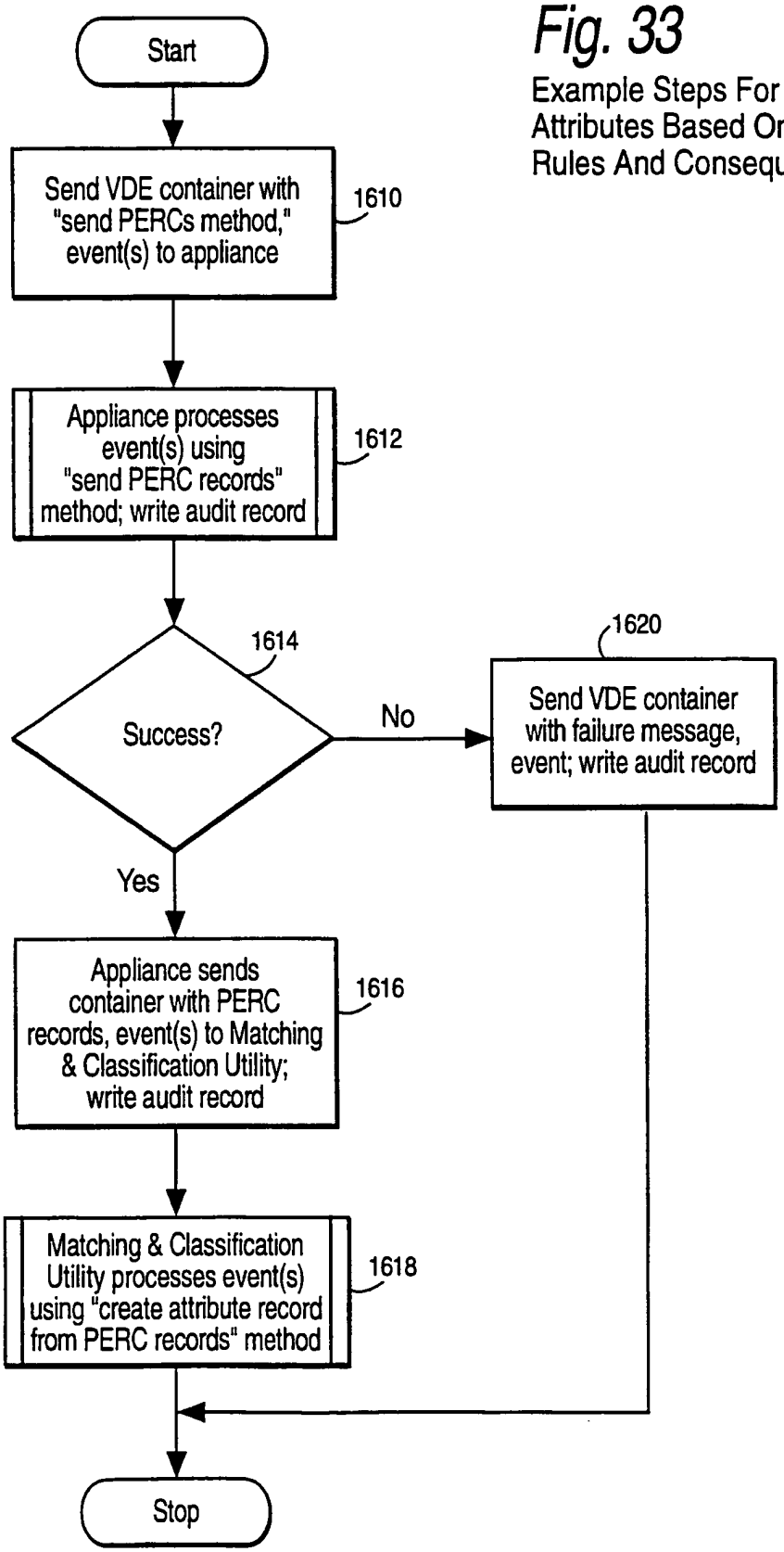


Fig. 33

Example Steps For Determining Attributes Based On Available Rules And Consequences

Petitioner Apple Inc. - Exhibit 1004, p. 5712

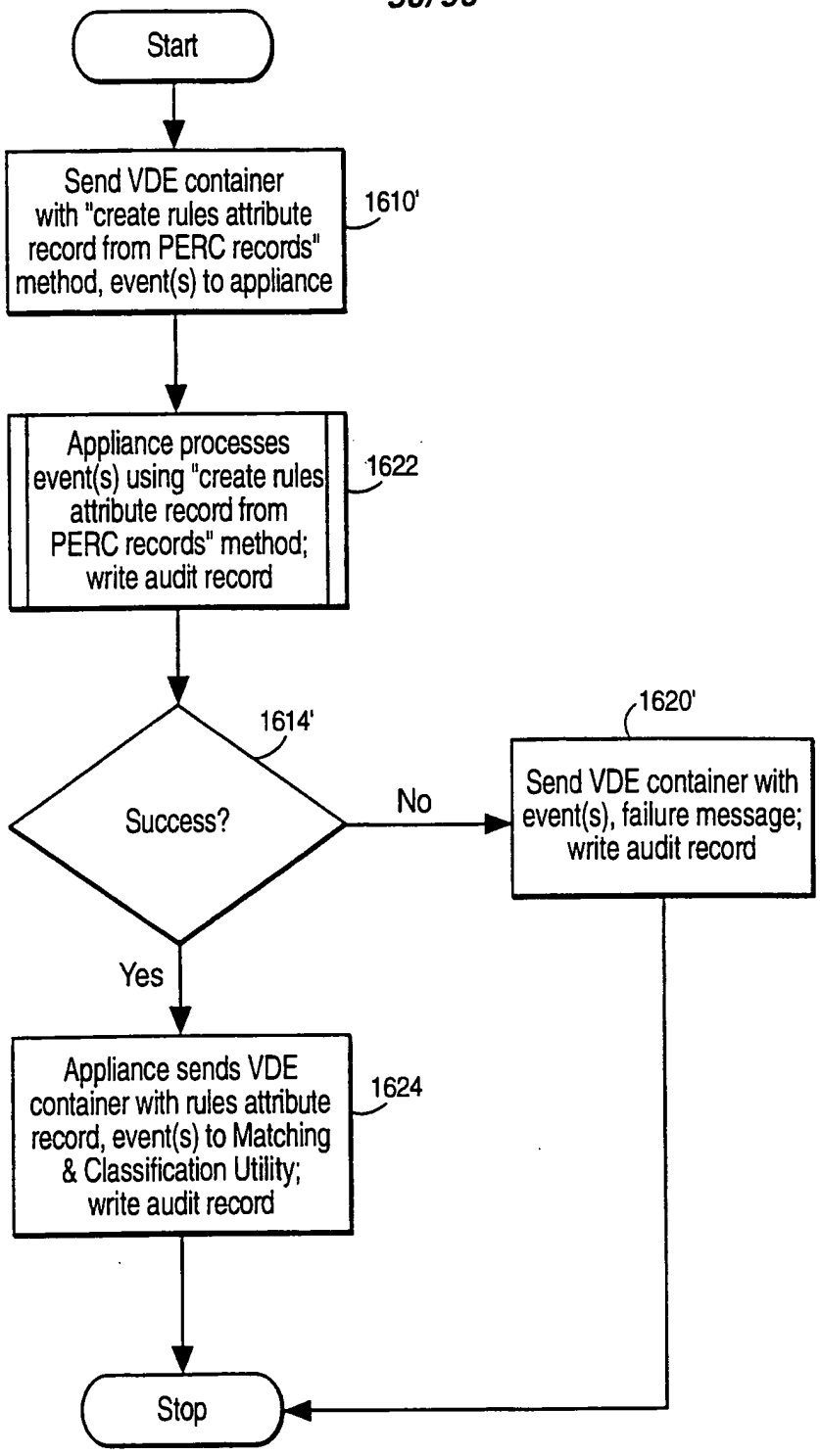


Fig. 34

Example Steps For Determining Attributes Based On Available Rules And Consequences

Petitioner Apple Inc. - Exhibit 1004, p. 5713

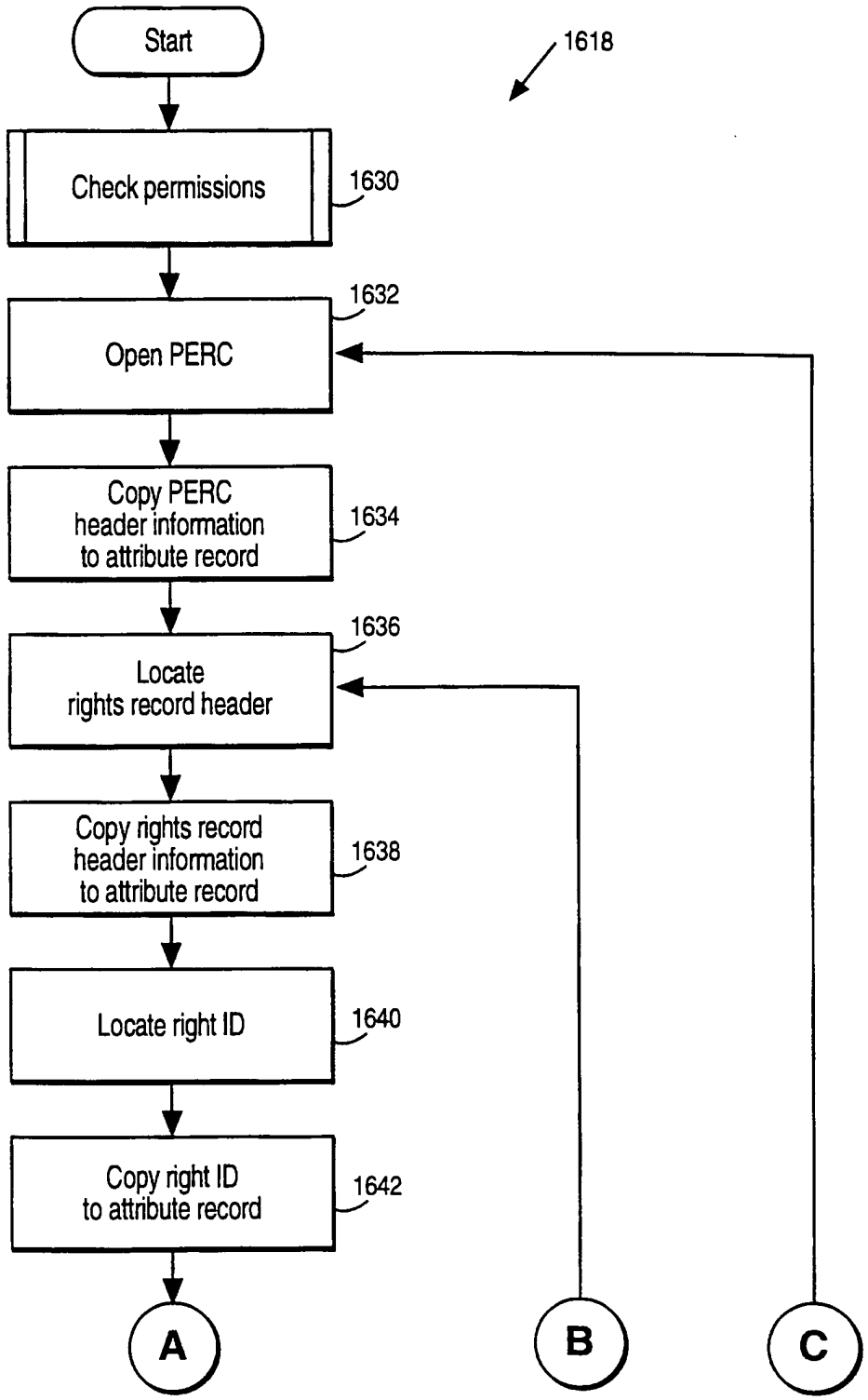


Fig. 35(A)

Construct Attribute Records From PERC Records Example Method

Petitioner Apple Inc. - Exhibit 1004, p. 5714

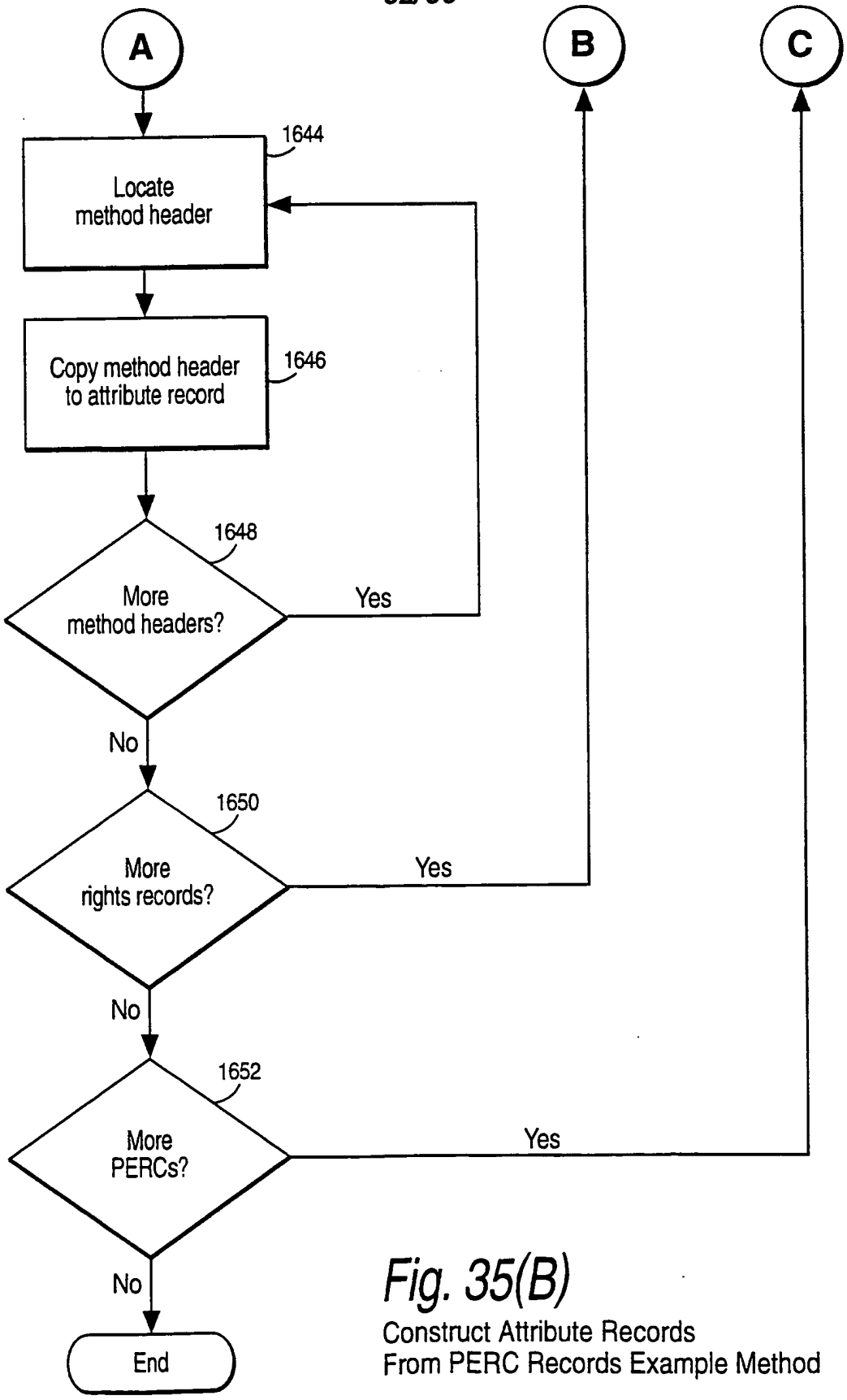


Fig. 35(B)

Construct Attribute Records From PERC Records Example Method

Petitioner Apple Inc. - Exhibit 1004, p. 5715

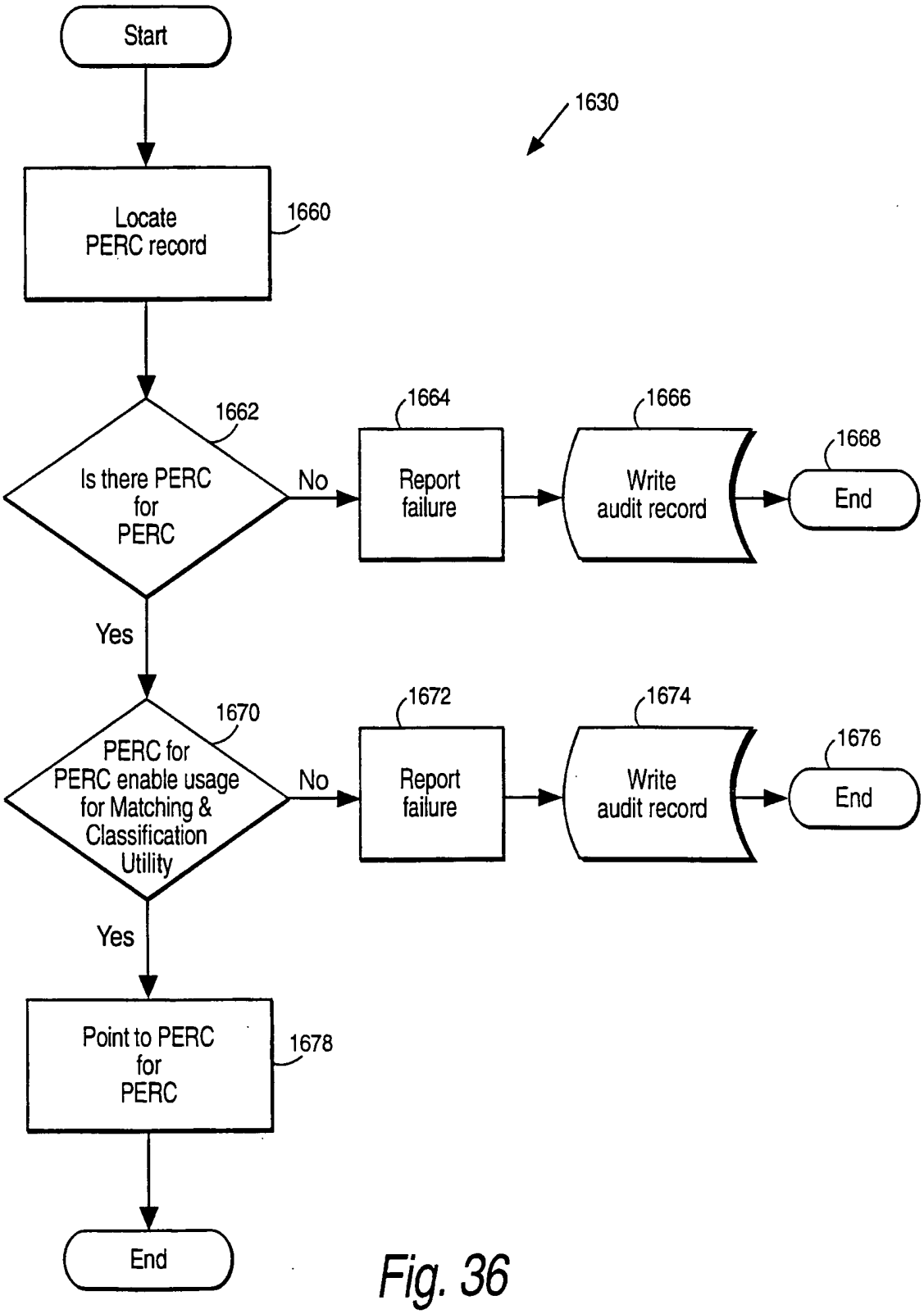


Fig. 36

Check Permissions Record Example Steps

Petitioner Apple Inc. - Exhibit 1004, p. 5716

Fig. 37(A) Example Rights Attribute Record From PERCs

Fig. 37(B) Example Attribute Record

Fig. 37(C) Example Attribute Records From PERC Record

Petitioner Apple Inc. - Exhibit 1004, p. 5717

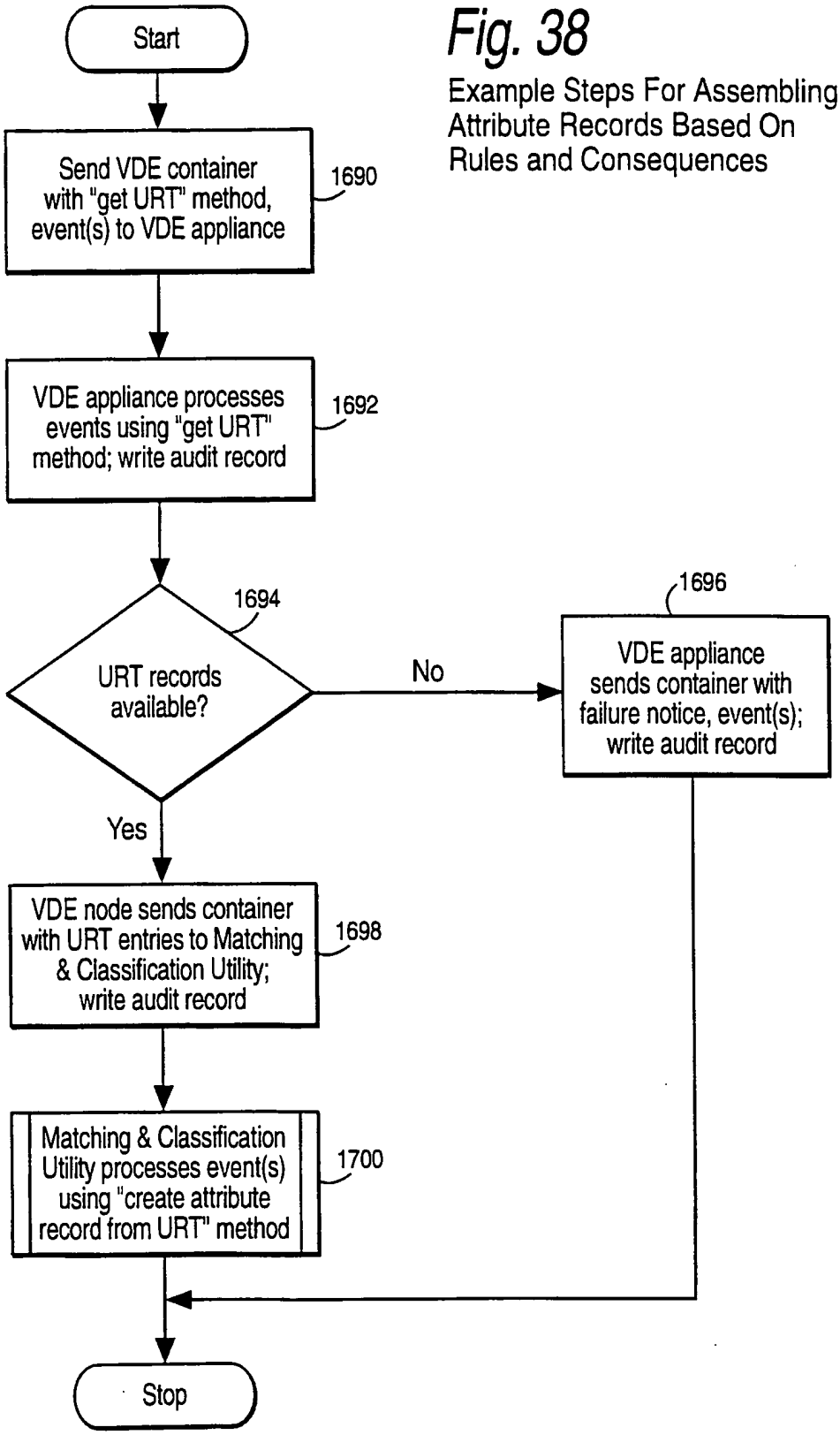


Fig. 38

Example Steps For Assembling Attribute Records Based On Rules and Consequences

Petitioner Apple Inc. - Exhibit 1004, p. 5718

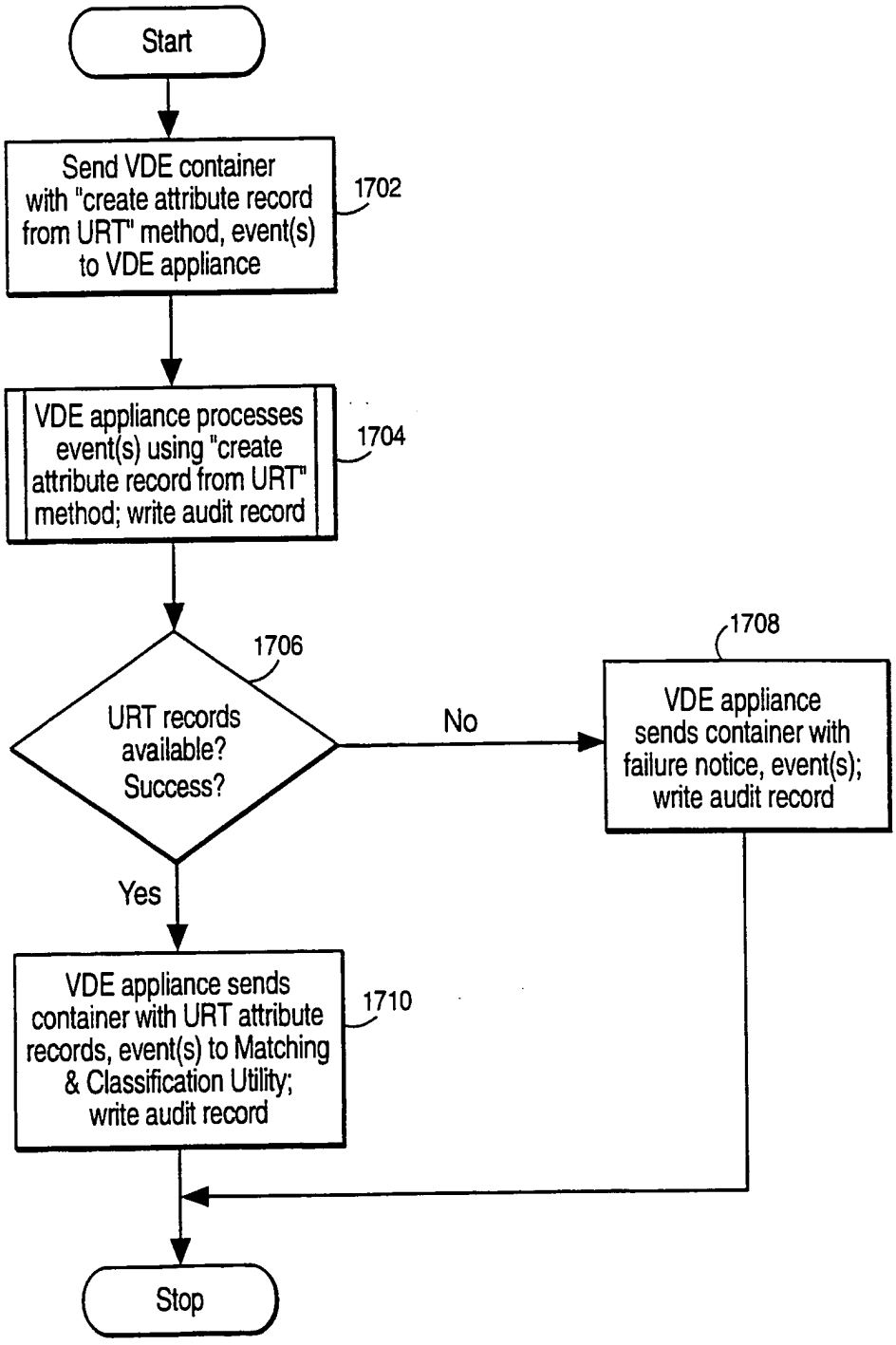


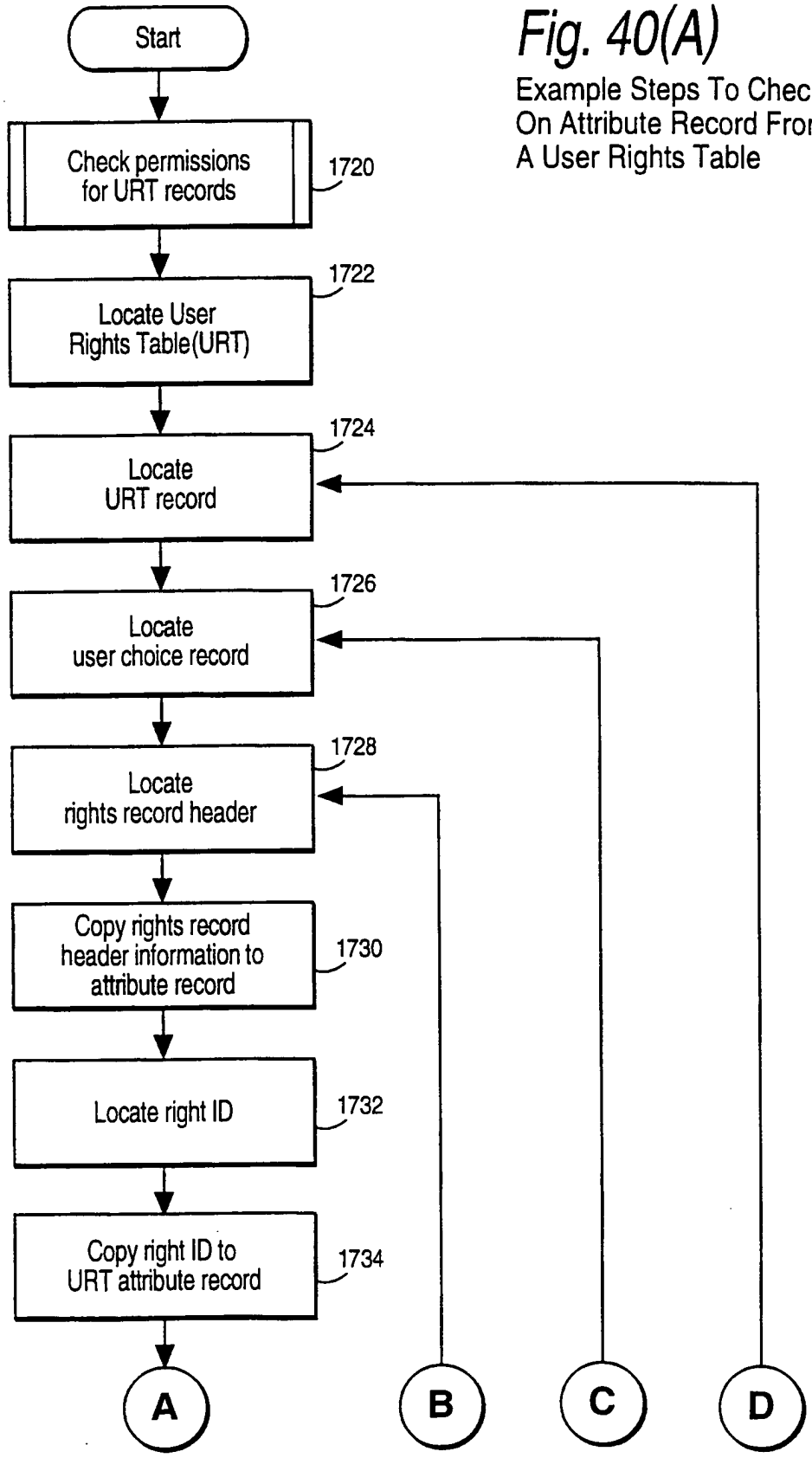
Fig. 39

Example Steps For Assembling Attribute Records Based On Rules and Consequences

Petitioner Apple Inc. - Exhibit 1004, p. 5719

Fig. 40(A)

Example Steps To Check On Attribute Record From A User Rights Table



Petitioner Apple Inc. - Exhibit 1004, p. 5720

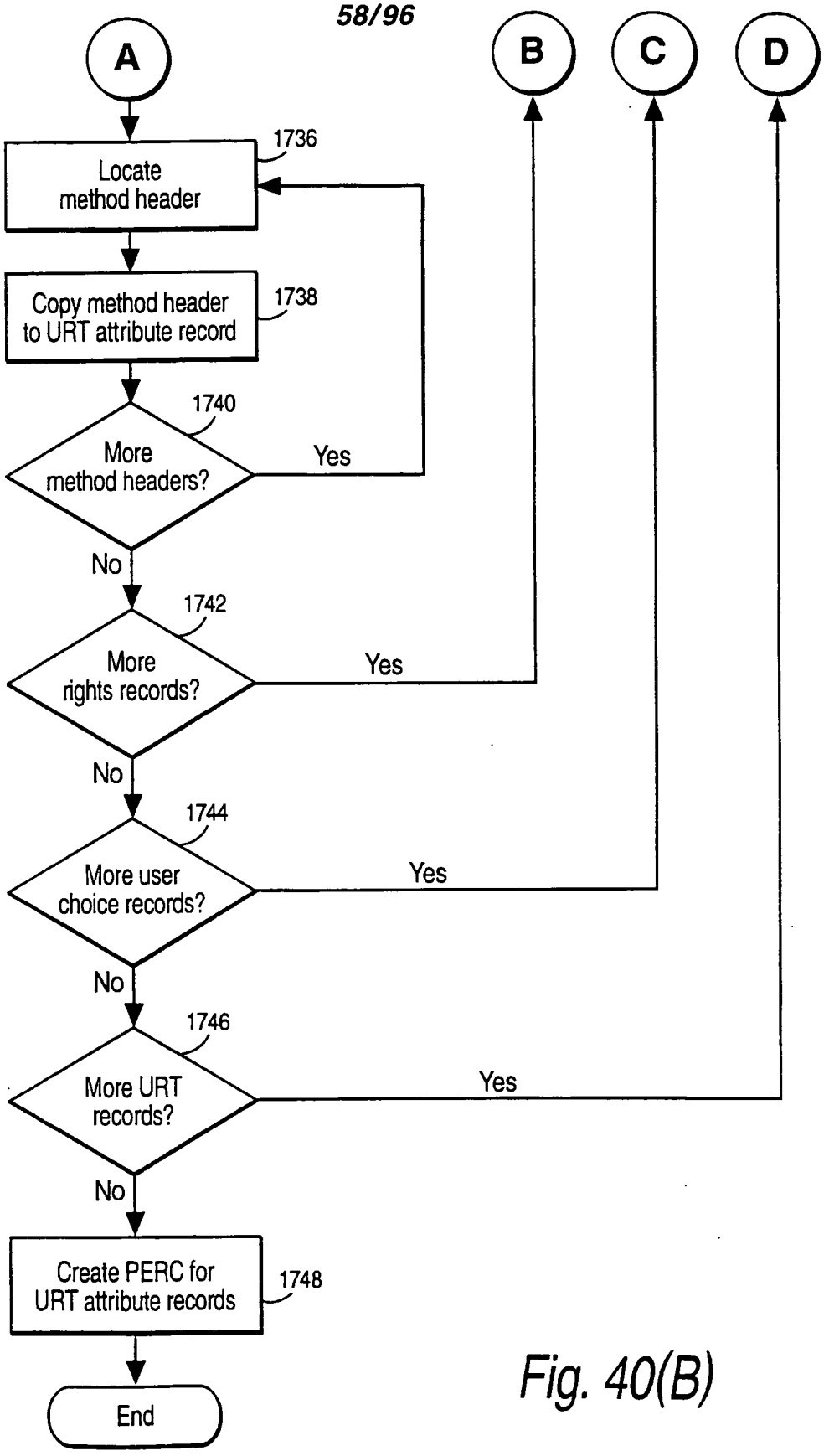
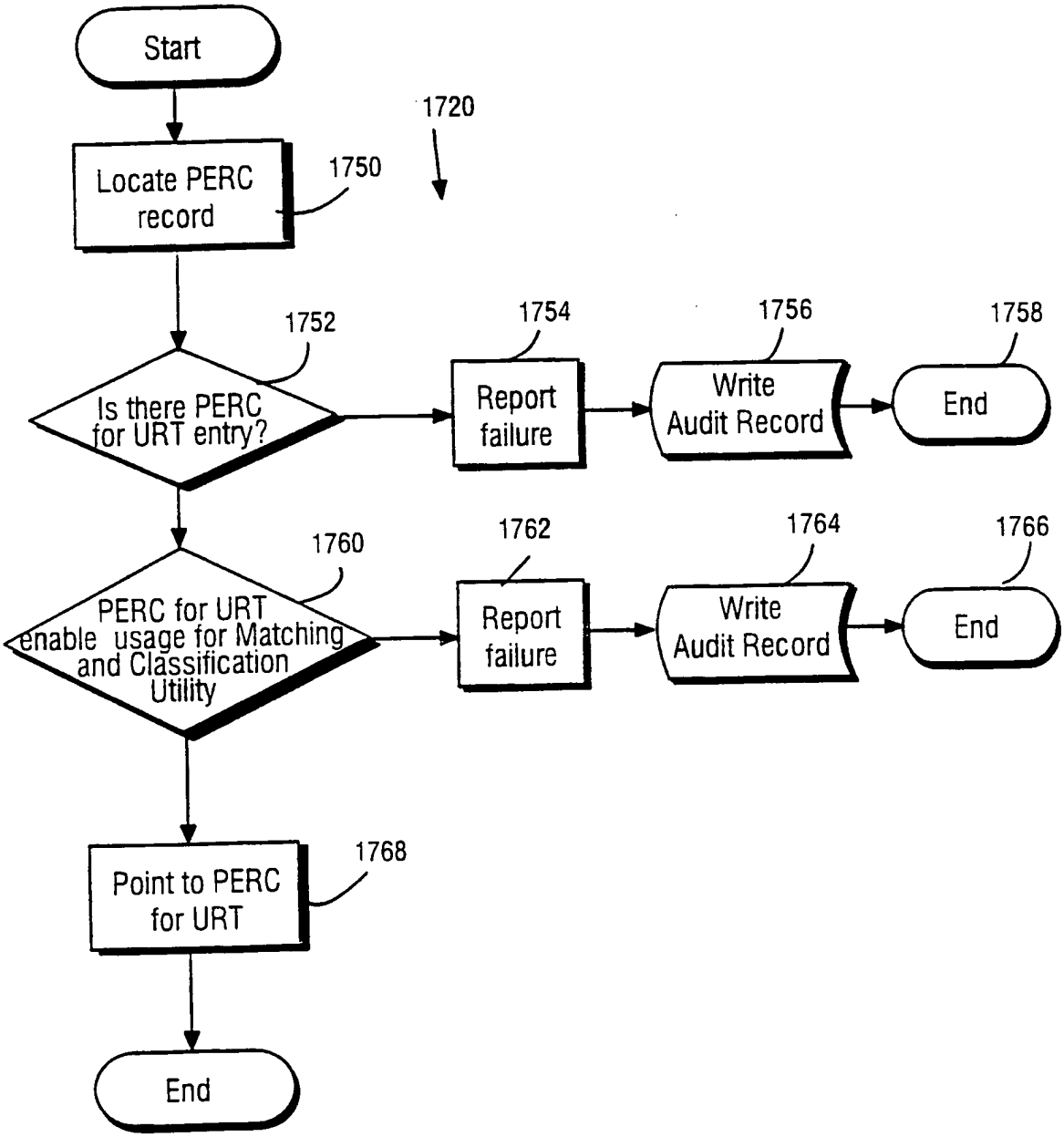


Fig. 40(B)

Petitioner Apple Inc. - Exhibit 1004, p. 5721

Fig. 41

Contruct attribute records from PERC records method example



Petitioner Apple Inc. - Exhibit 1004, p. 5722

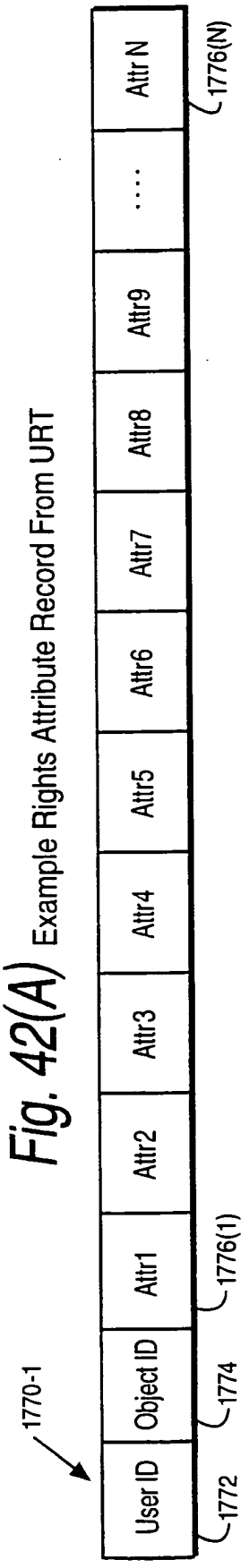


Fig. 42(A) Example Rights Attribute Record From URT

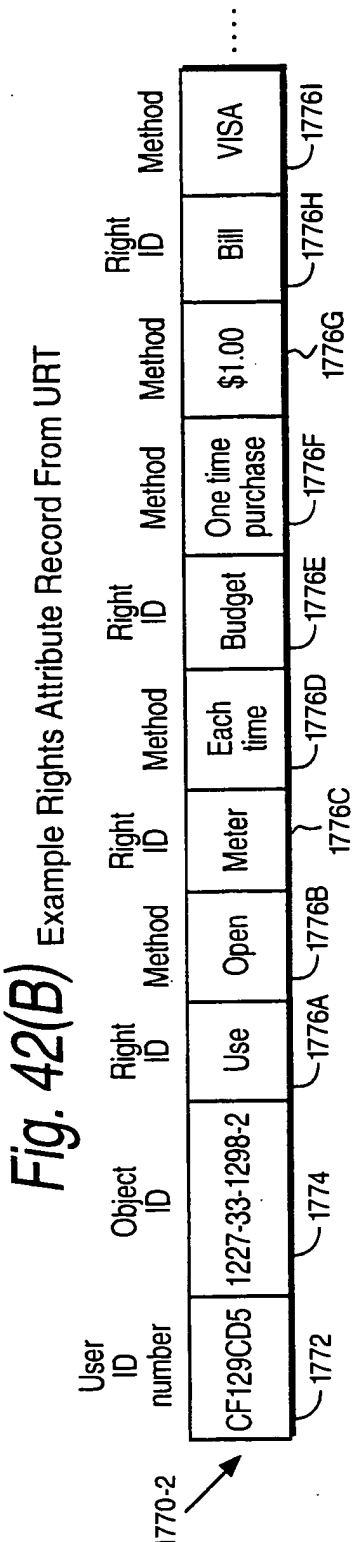


Fig. 42(B) Example Rights Attribute Record From URT

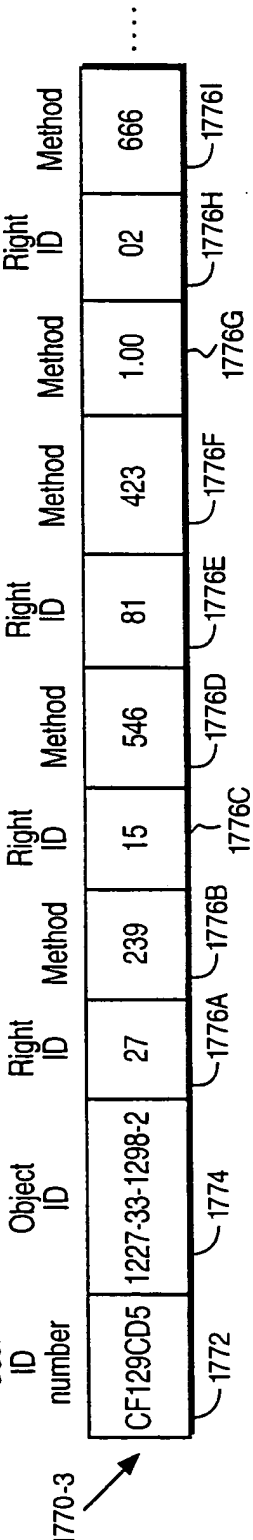
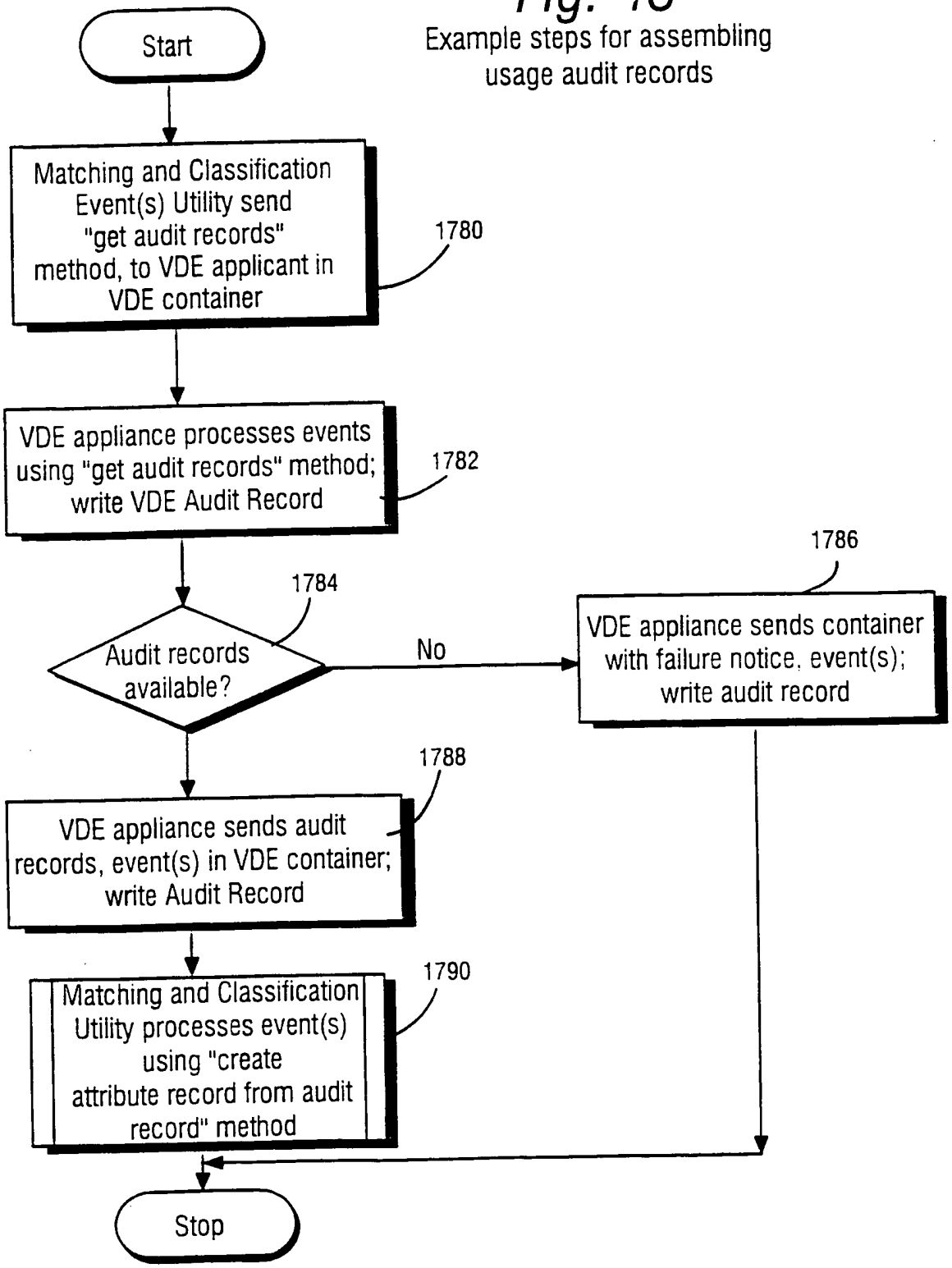


Fig. 42(C) Example User Choice Attribute Records (From URT)

Petitioner Apple Inc. - Exhibit 1004, p. 5723

Fig. 43

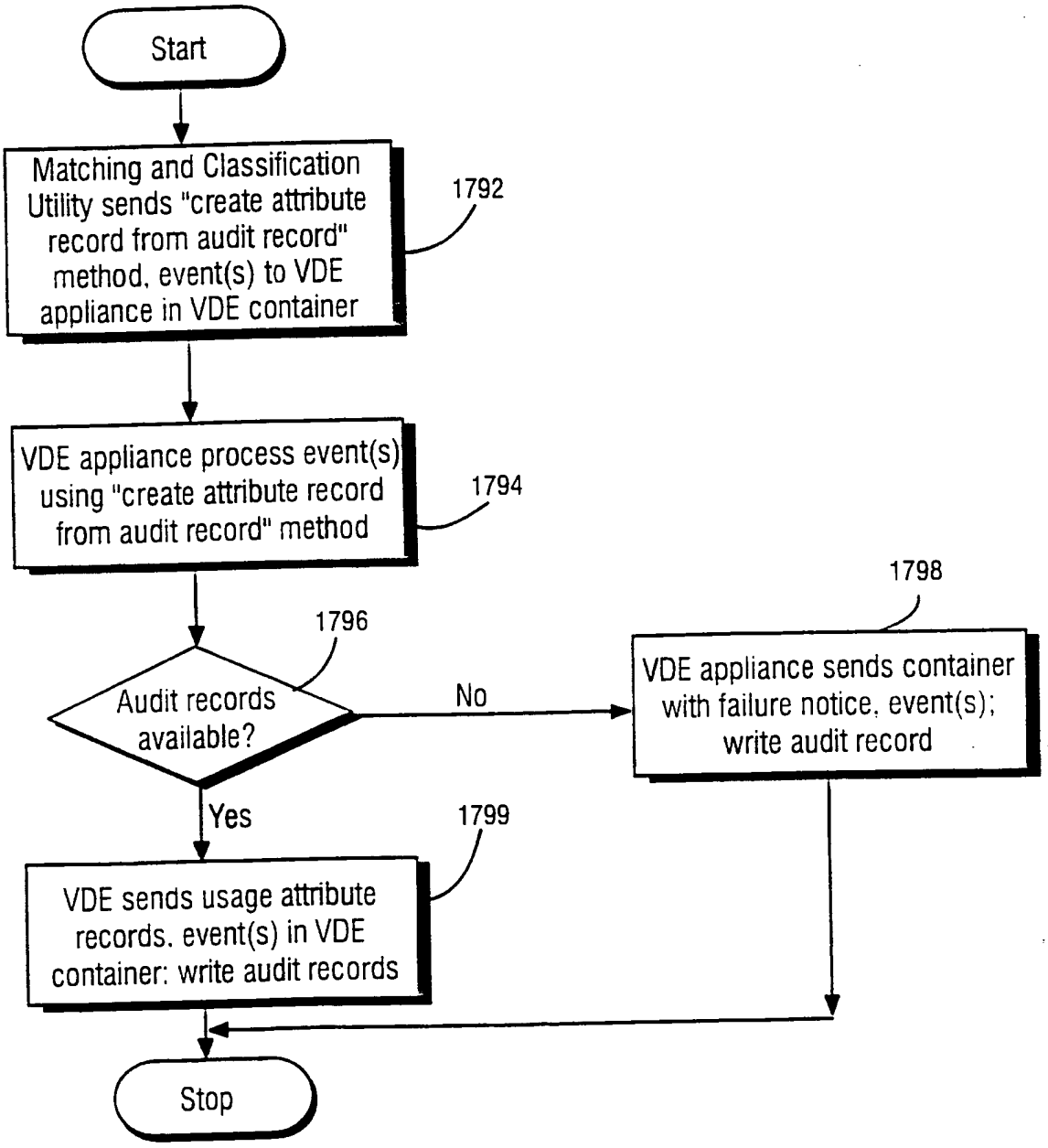
Example steps for assembling usage audit records



Petitioner Apple Inc. - Exhibit 1004, p. 5724

Fig. 44

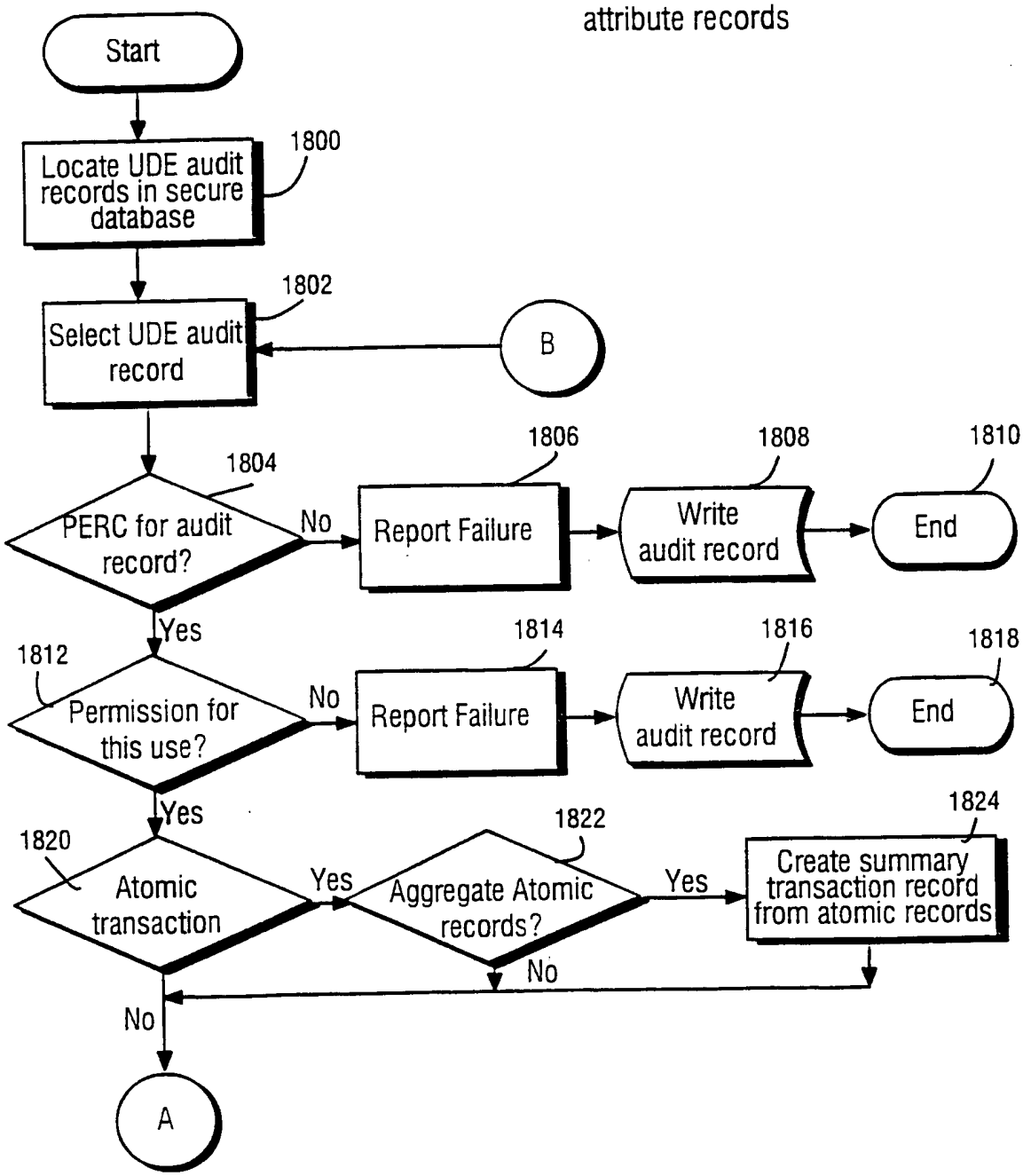
Example steps for assembling usage audit records



Petitioner Apple Inc. - Exhibit 1004, p. 5725

Fig. 45(A)

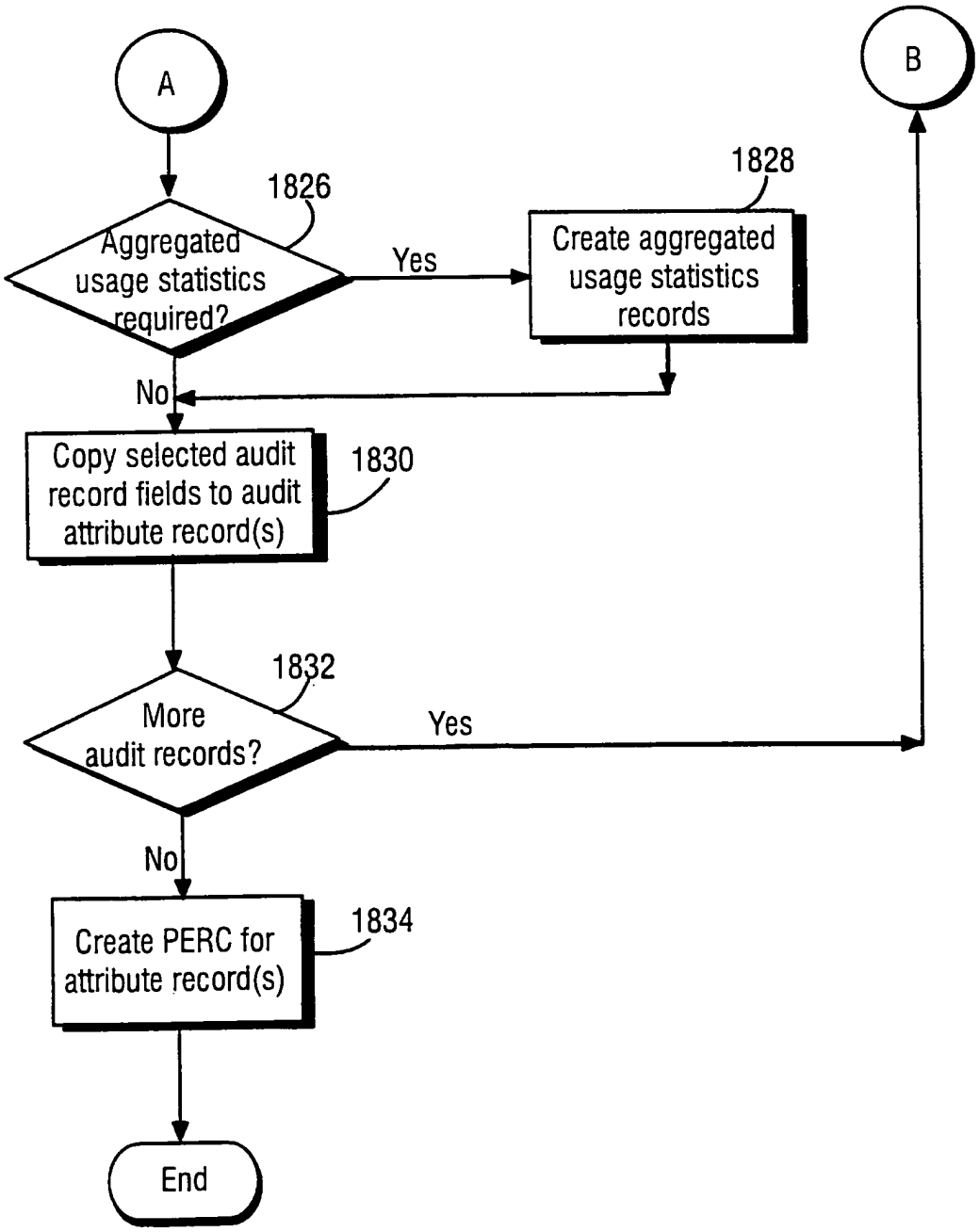
Example steps to create audit attribute records



Petitioner Apple Inc. - Exhibit 1004, p. 5726

Fig. 45(B)

Example steps to create audit attribute records



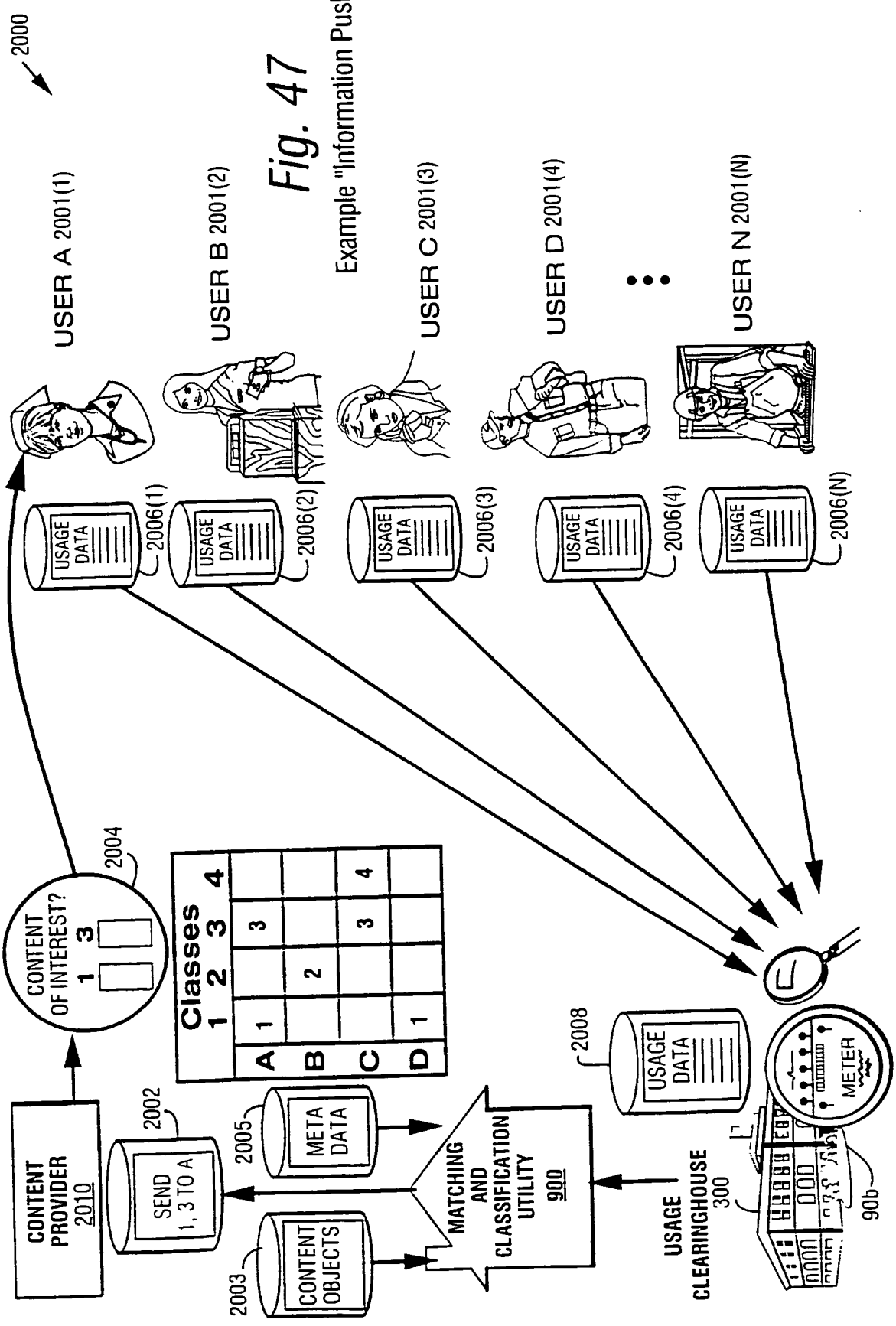
Petitioner Apple Inc. - Exhibit 1004, p. 5727

Fig. 46(A) Example Usage Attribute Record From UDE Audit Record

Fig. 46(B) Example Usage Attribute/Statistics Records From UDE Audit Records

Petitioner Apple Inc. - Exhibit 1004, p. 5728

Fig. 47
Example "Information Push"



Petitioner Apple Inc. - Exhibit 1004, p. 5729

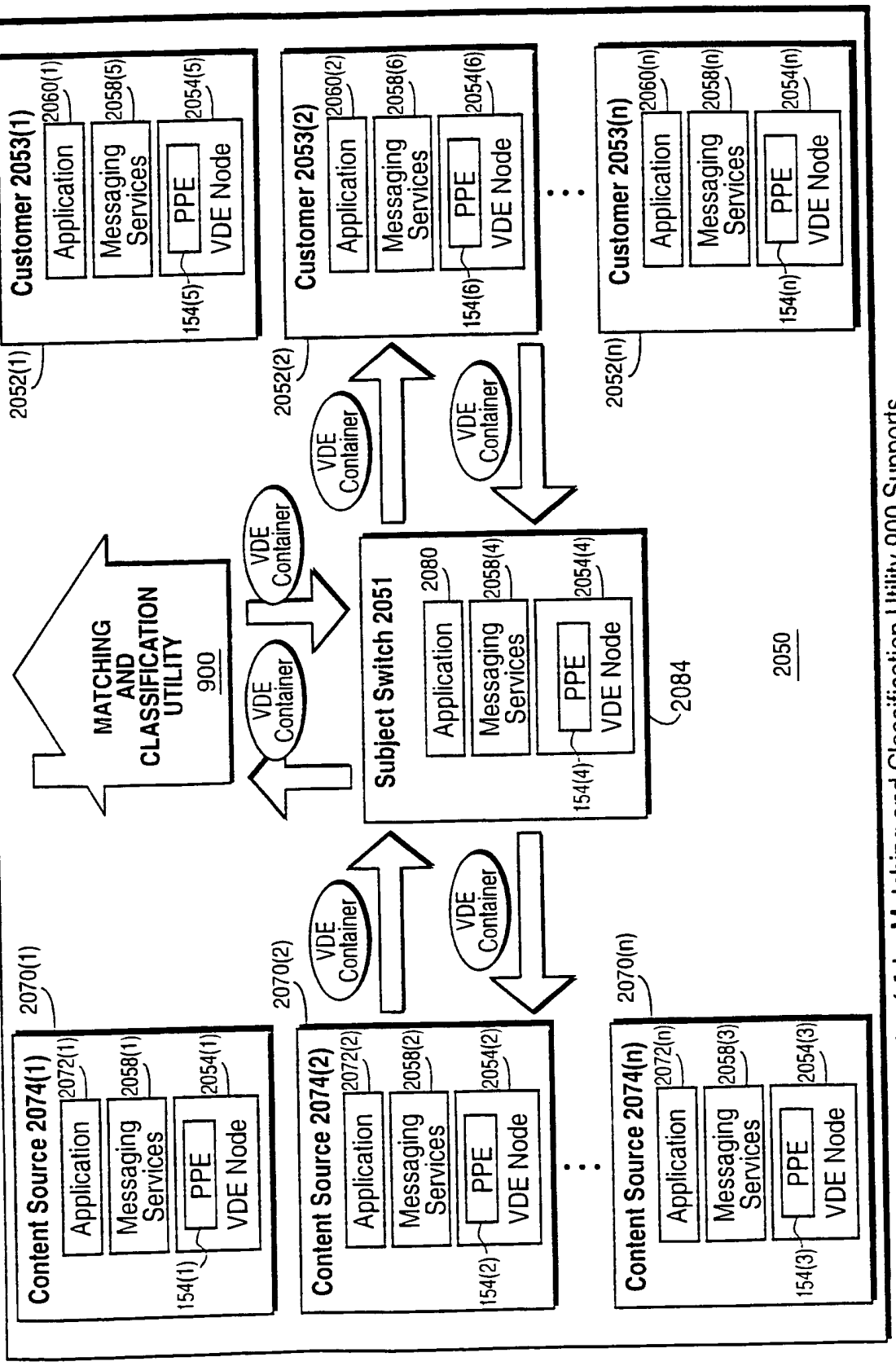


Fig. 47(A) Matching and Classification Utility 900 Supports "Push" models using Subject Switching and Messaging Services

Petitioner Apple Inc. - Exhibit 1004, p. 5730

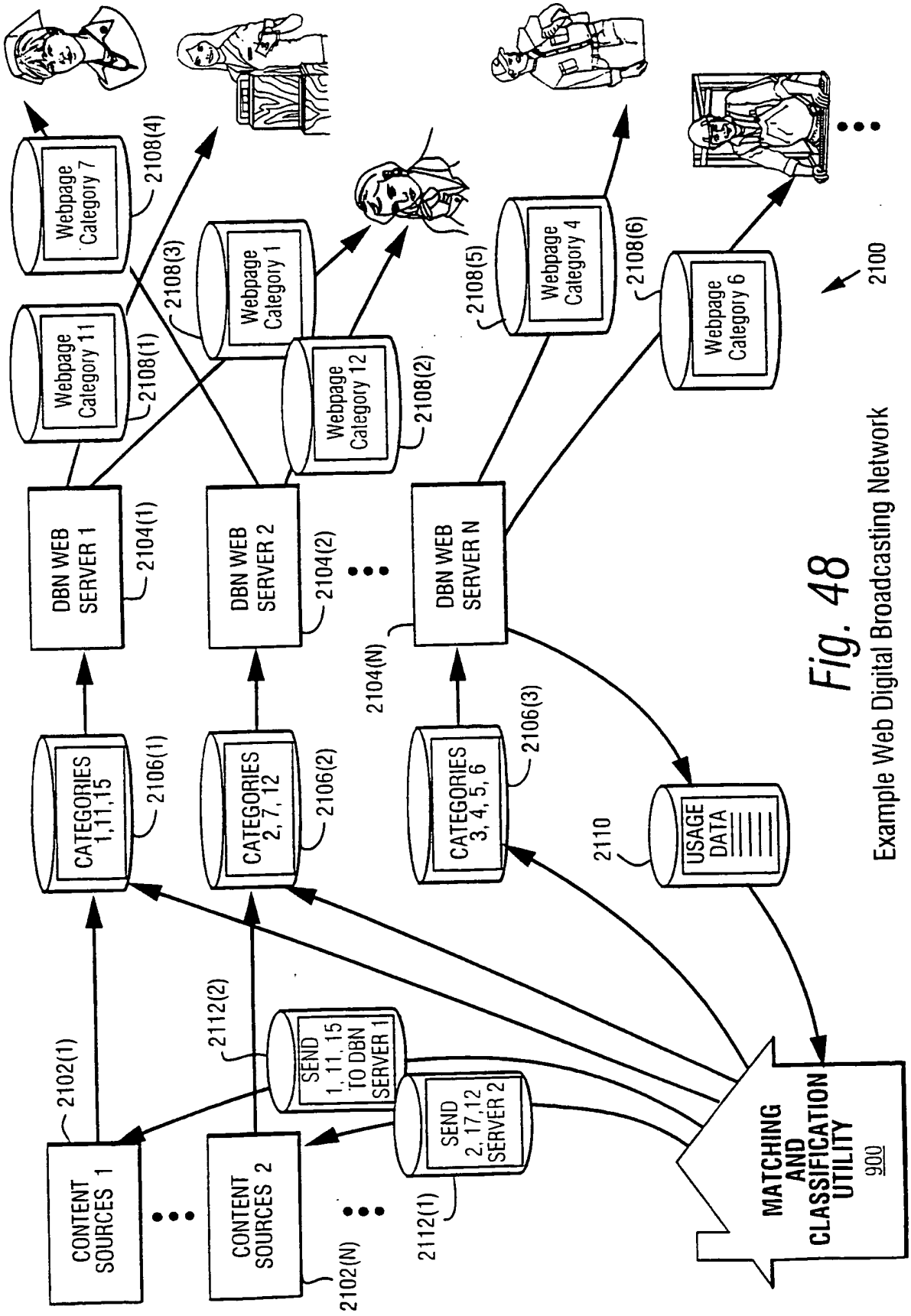


Fig. 48

Example Web Digital Broadcasting Network

Petitioner Apple Inc. - Exhibit 1004, p. 5731

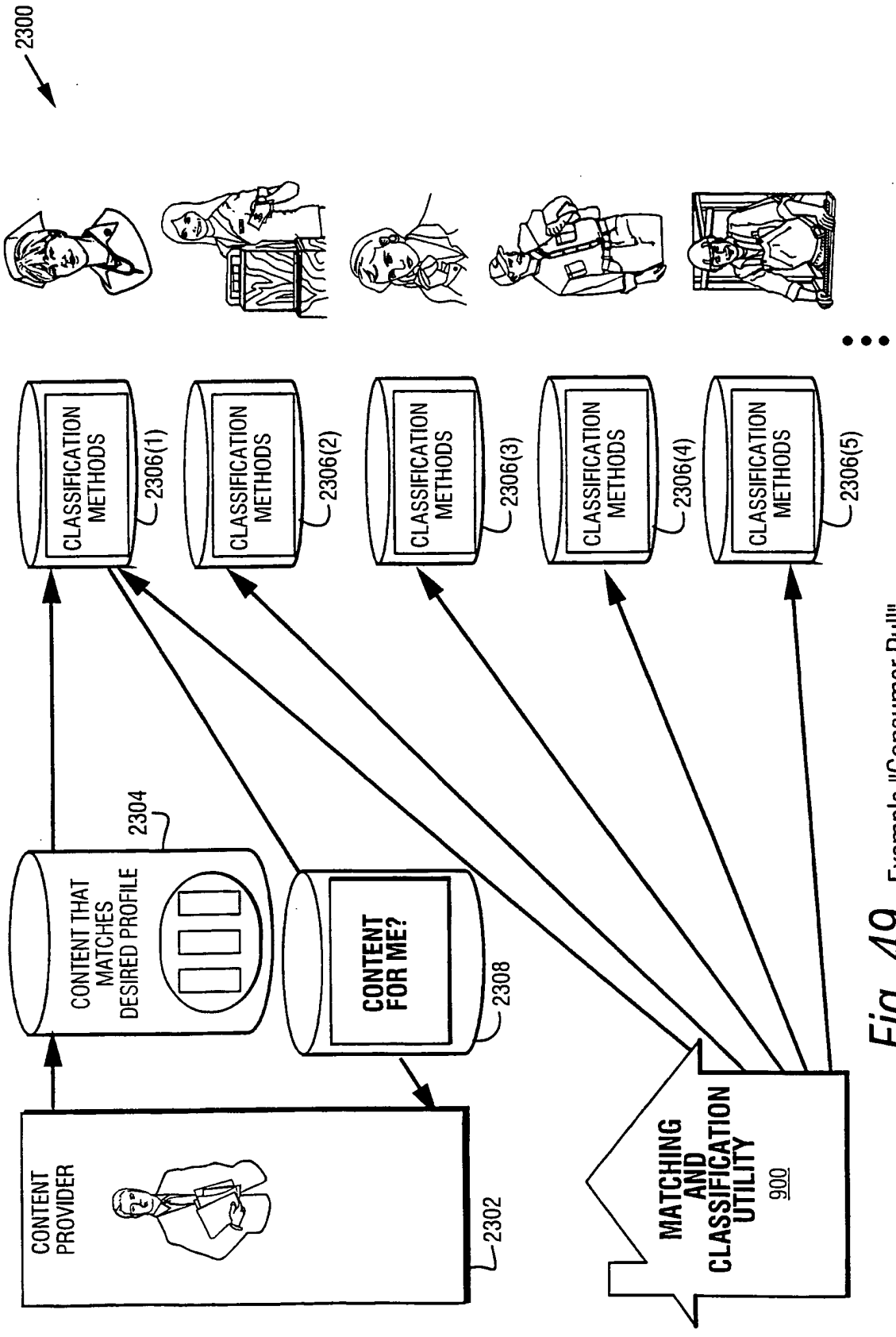


Fig. 49 Example "Consumer Pull"

Petitioner Apple Inc. - Exhibit 1004, p. 5732

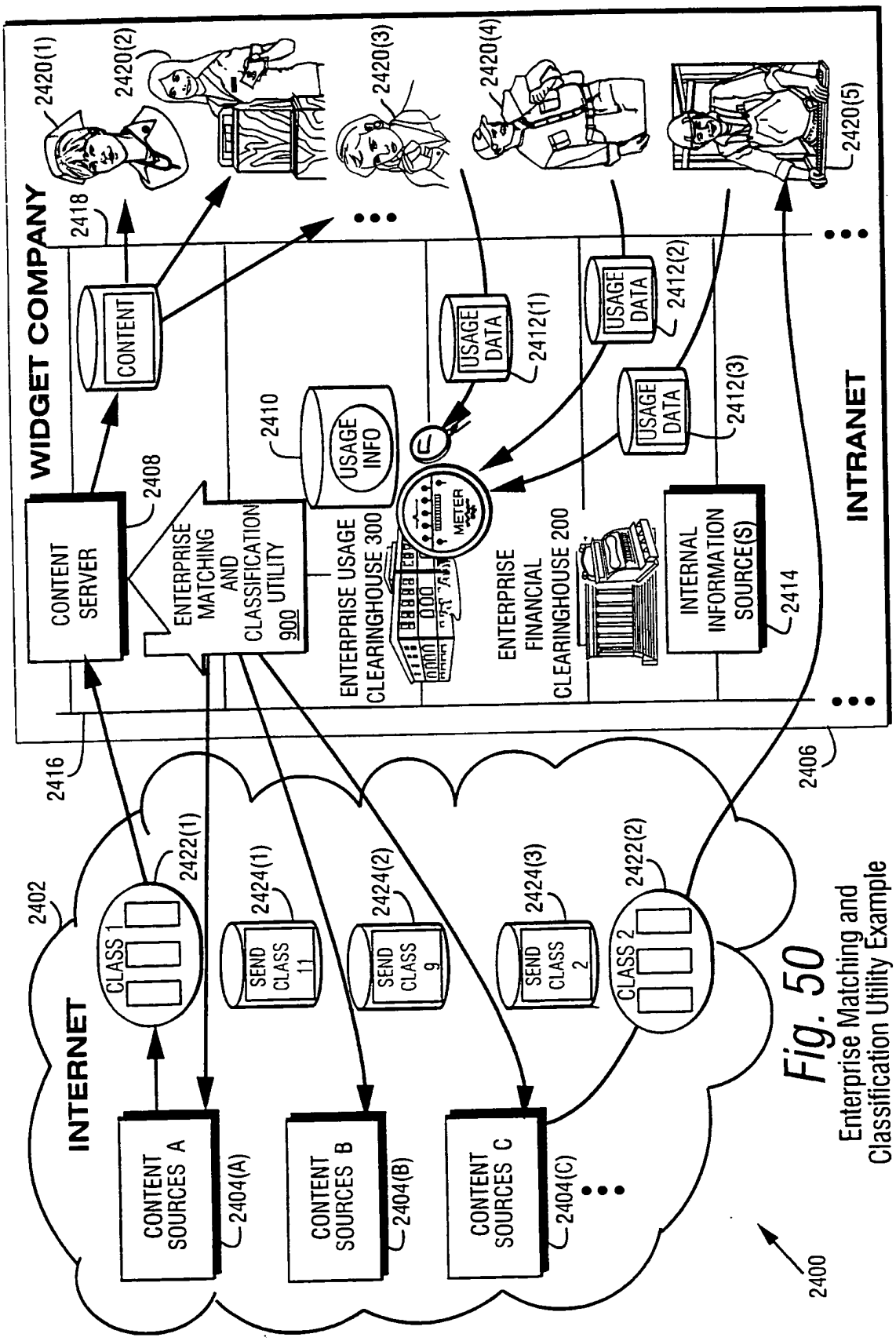


Fig. 50
Enterprise Matching and Classification Utility Example

Petitioner Apple Inc. - Exhibit 1004, p. 5733

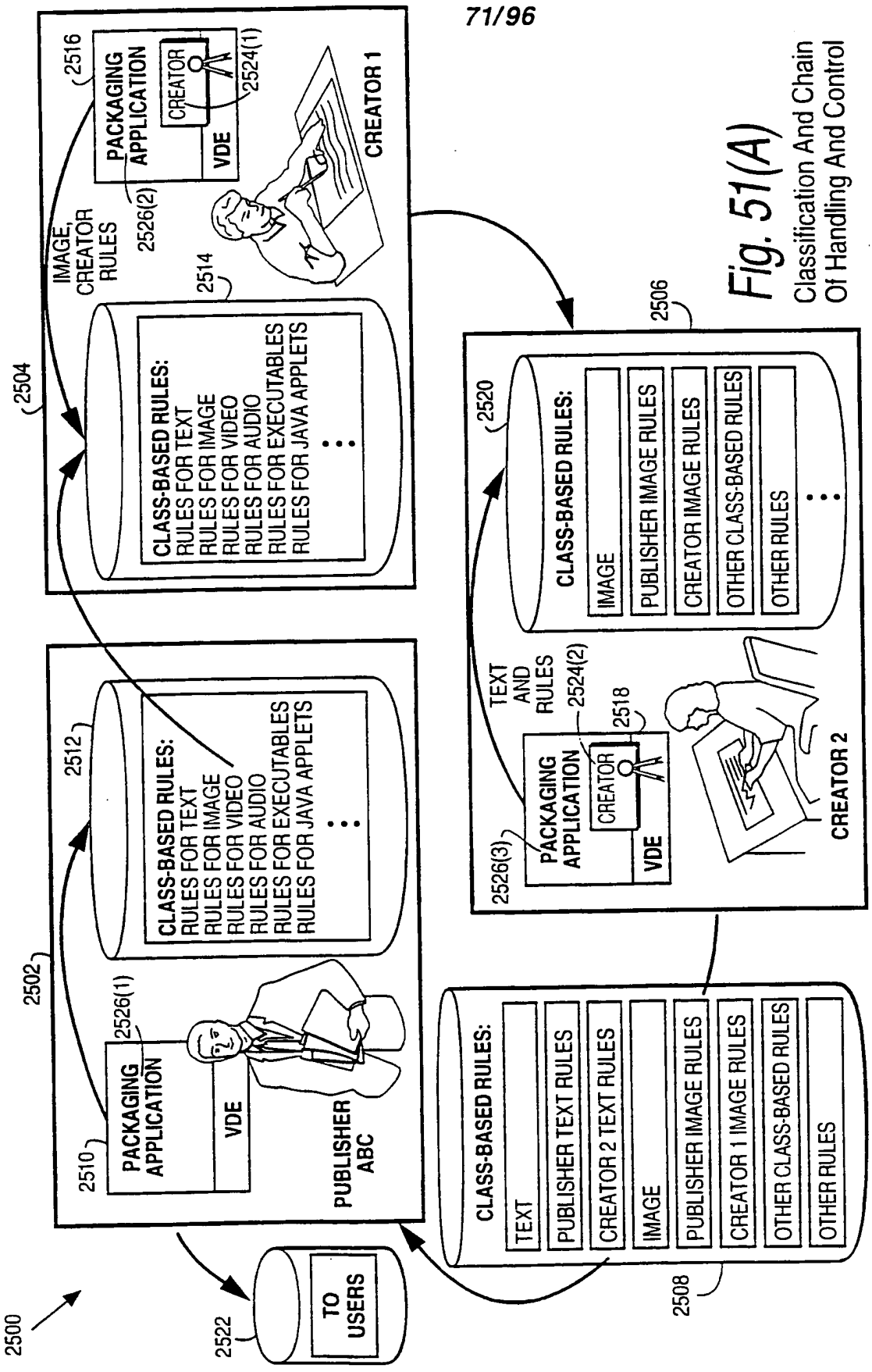


Fig. 51(A)
Classification And Chain Of Handling And Control

Petitioner Apple Inc. - Exhibit 1004, p. 5734

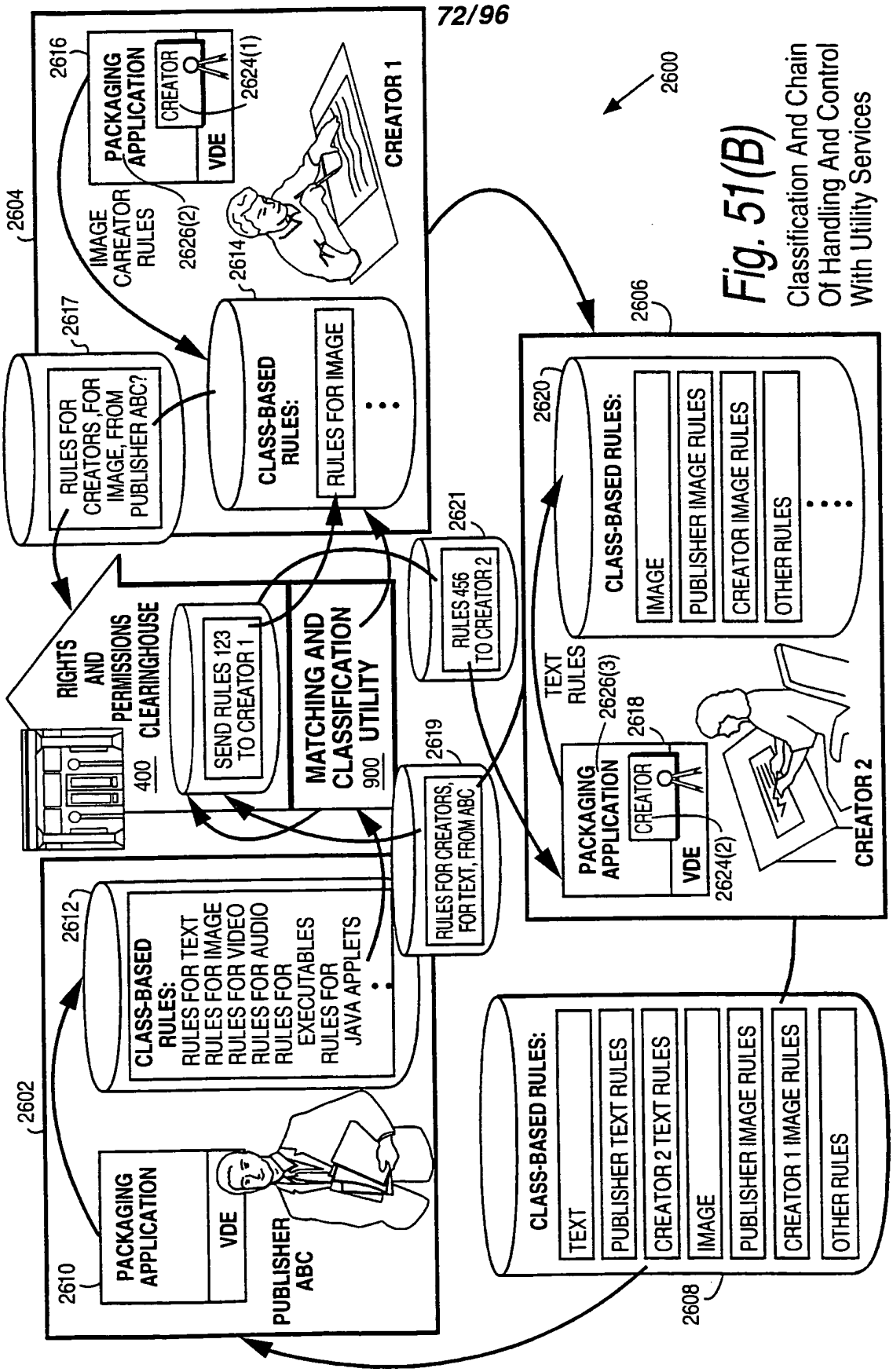


Fig. 51(B)
Classification And Chain Of Handling And Control With Utility Services

Petitioner Apple Inc. - Exhibit 1004, p. 5735

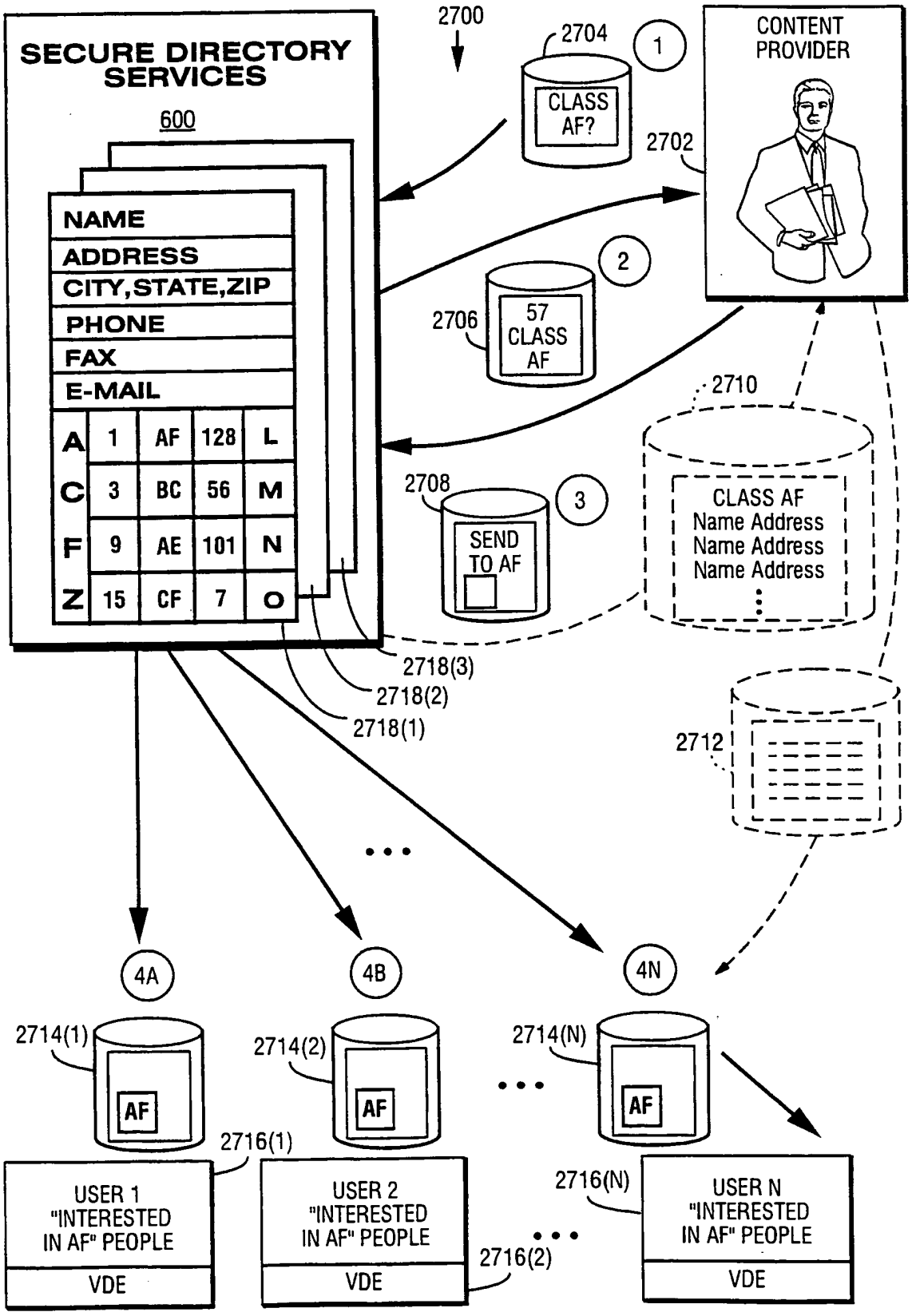


Fig. 52 Secure Directory Services

Petitioner Apple Inc. - Exhibit 1004, p. 5736

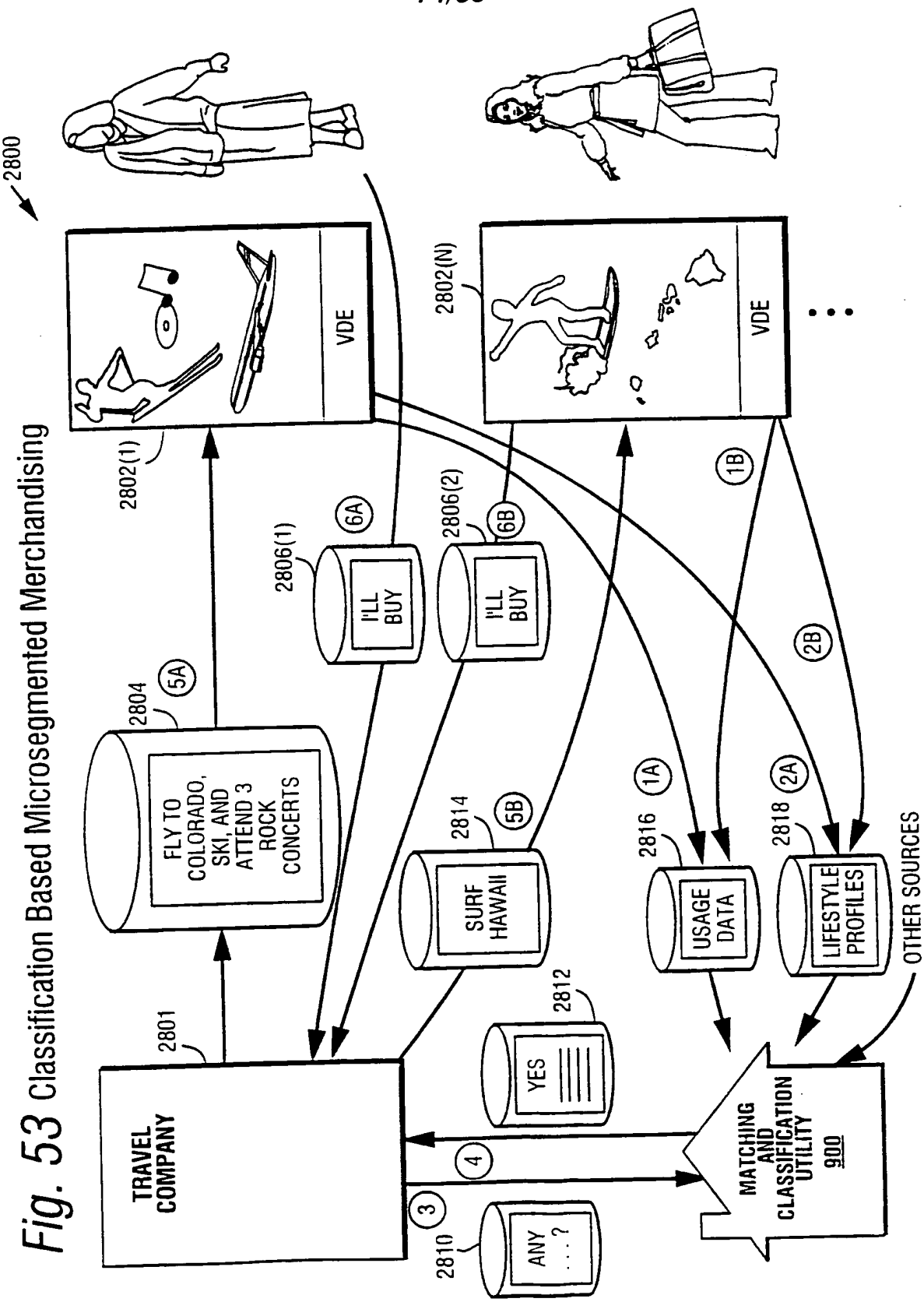


Fig. 53 Classification Based Microsegmented Merchandising

Petitioner Apple Inc. - Exhibit 1004, p. 5737

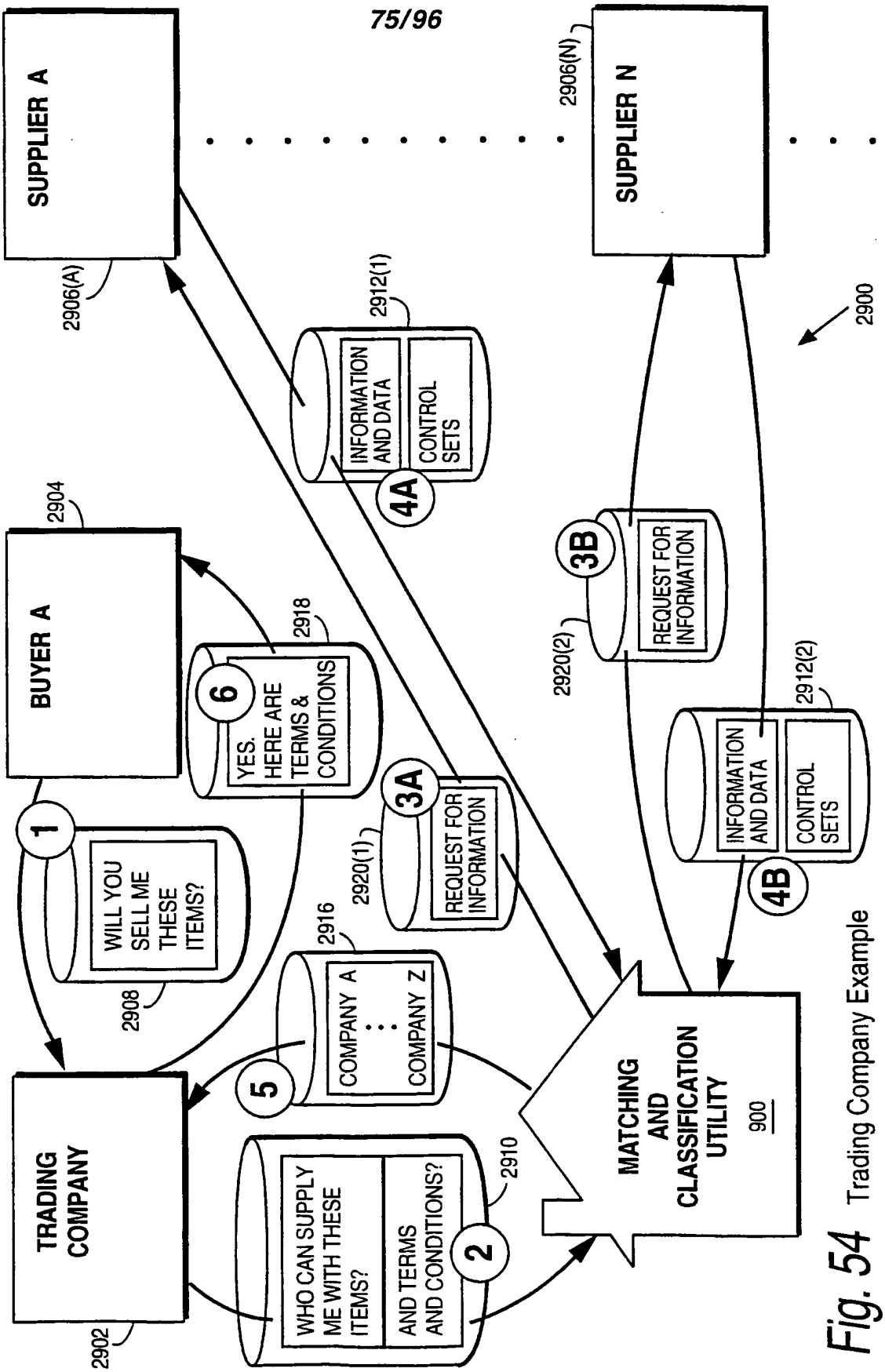


Fig. 54 Trading Company Example

Petitioner Apple Inc. - Exhibit 1004, p. 5738

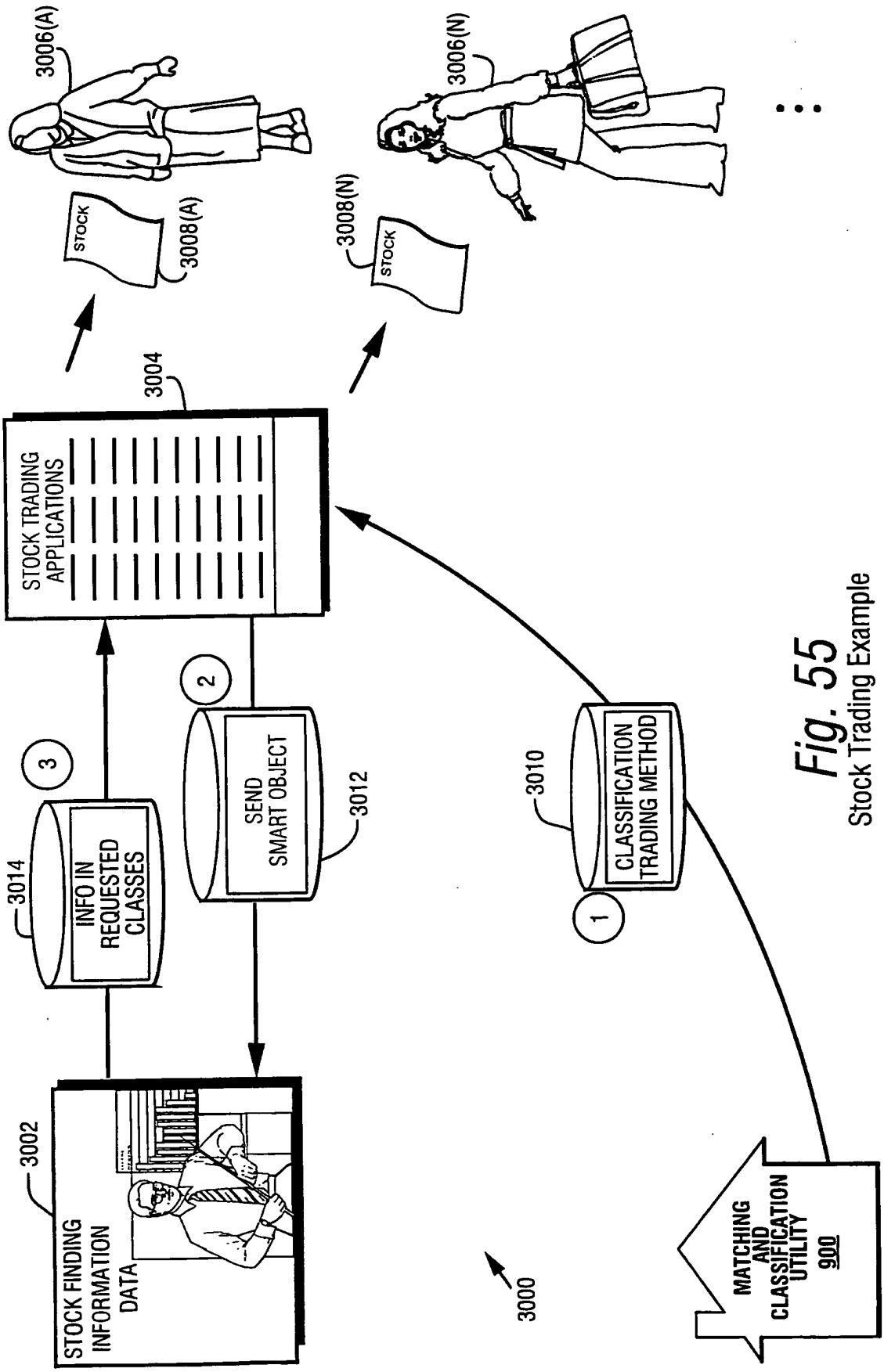


Fig. 55
Stock Trading Example

Petitioner Apple Inc. - Exhibit 1004, p. 5739

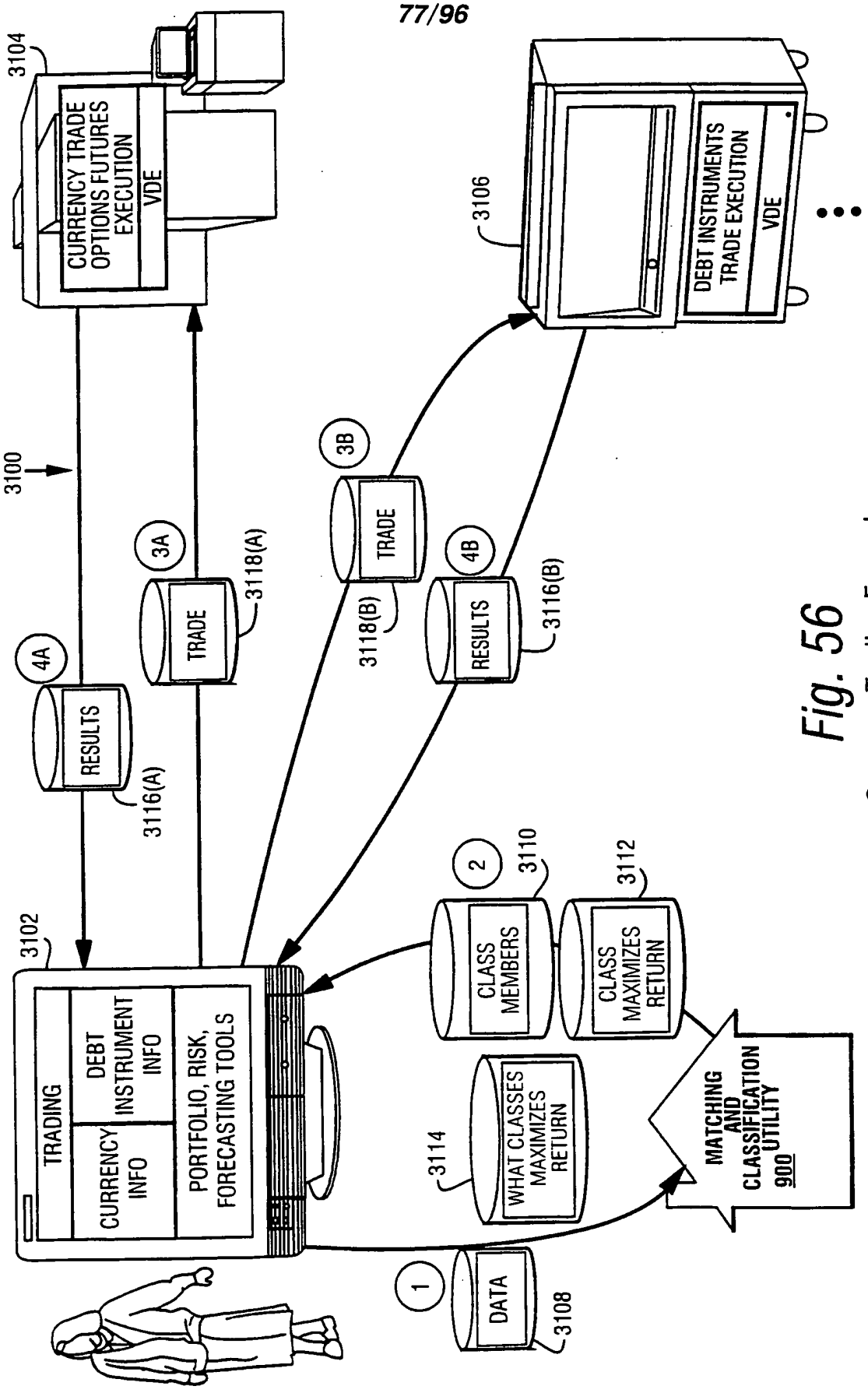


Fig. 56
Currency Trading Example

Petitioner Apple Inc. - Exhibit 1004, p. 5740

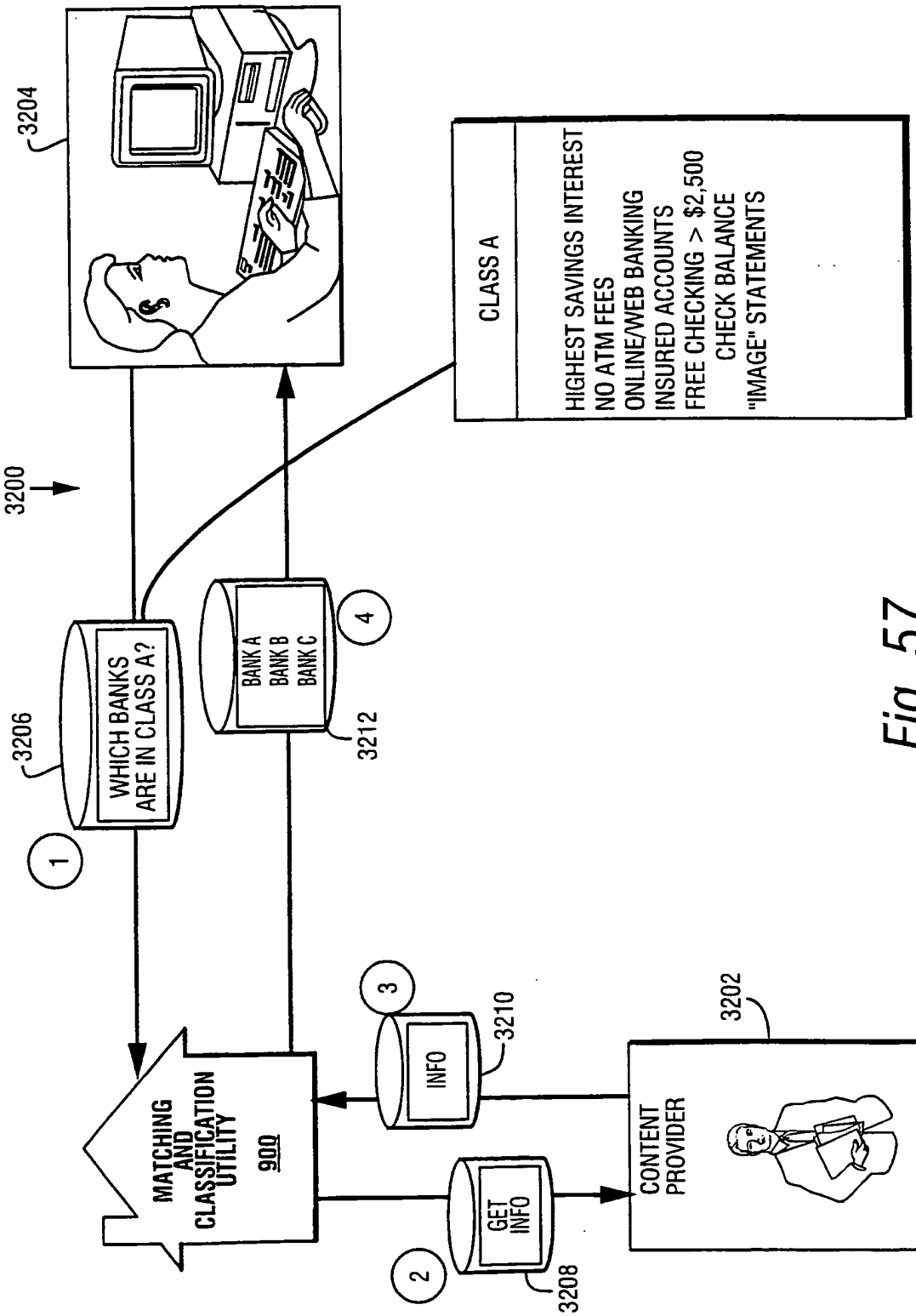


Fig. 57

Consumer Banking Example Locating Services In A Class!

Petitioner Apple Inc. - Exhibit 1004, p. 5741

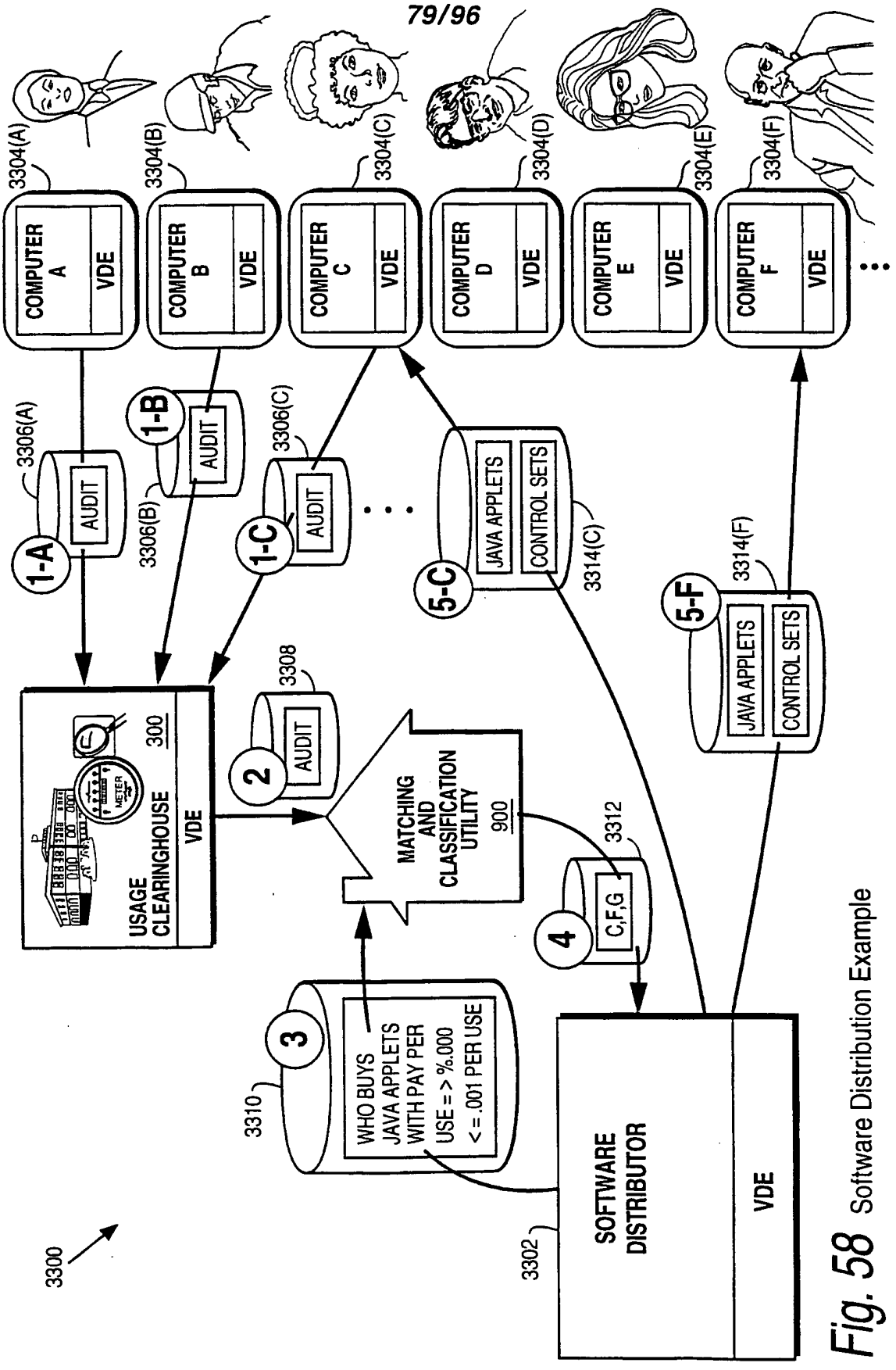


Fig. 58 Software Distribution Example

Petitioner Apple Inc. - Exhibit 1004, p. 5742

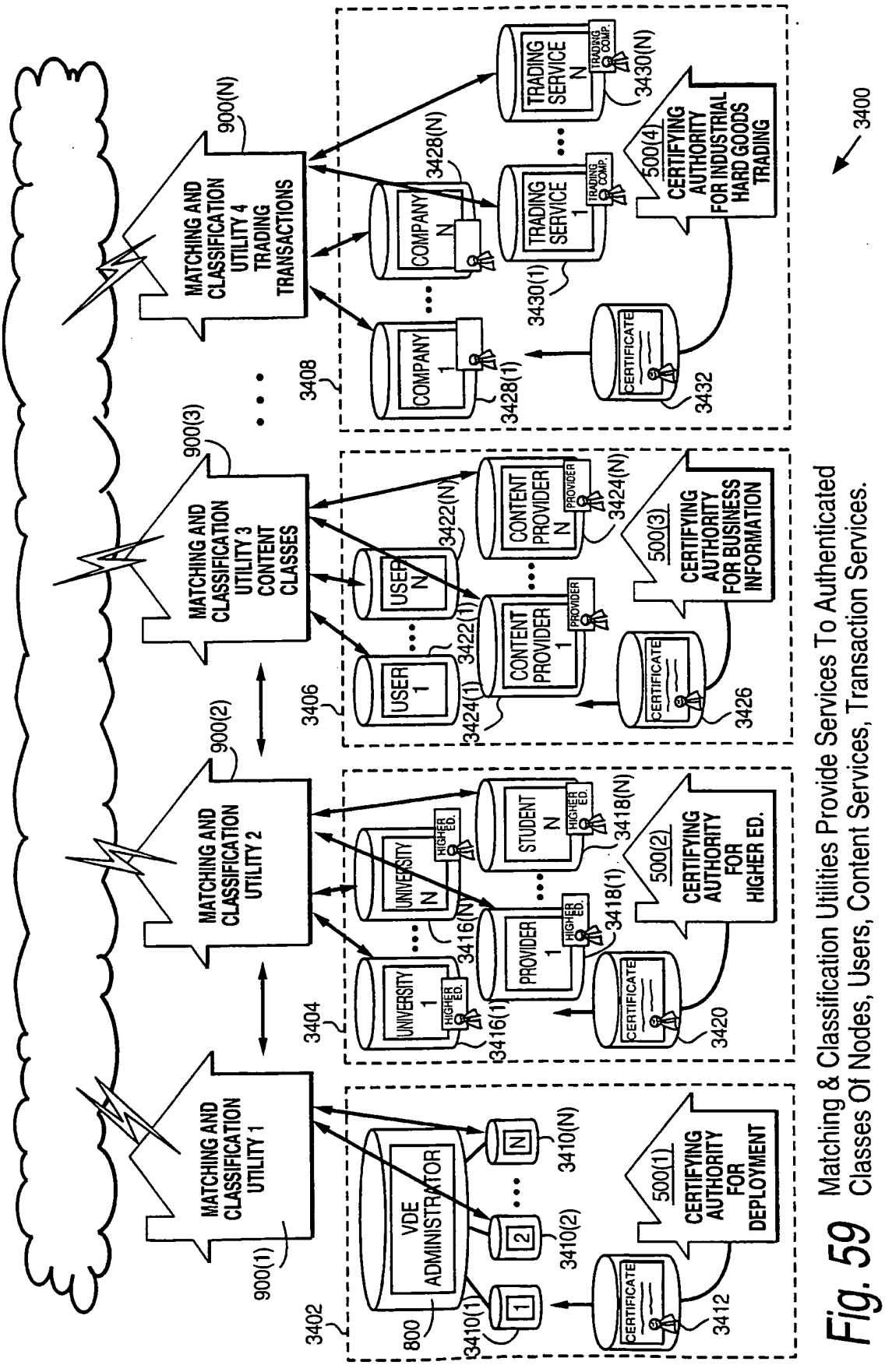


Fig. 59 Matching & Classification Utilities Provide Services To Authenticated Classes Of Nodes, Users, Content Services, Transaction Services.

Petitioner Apple Inc. - Exhibit 1004, p. 5743

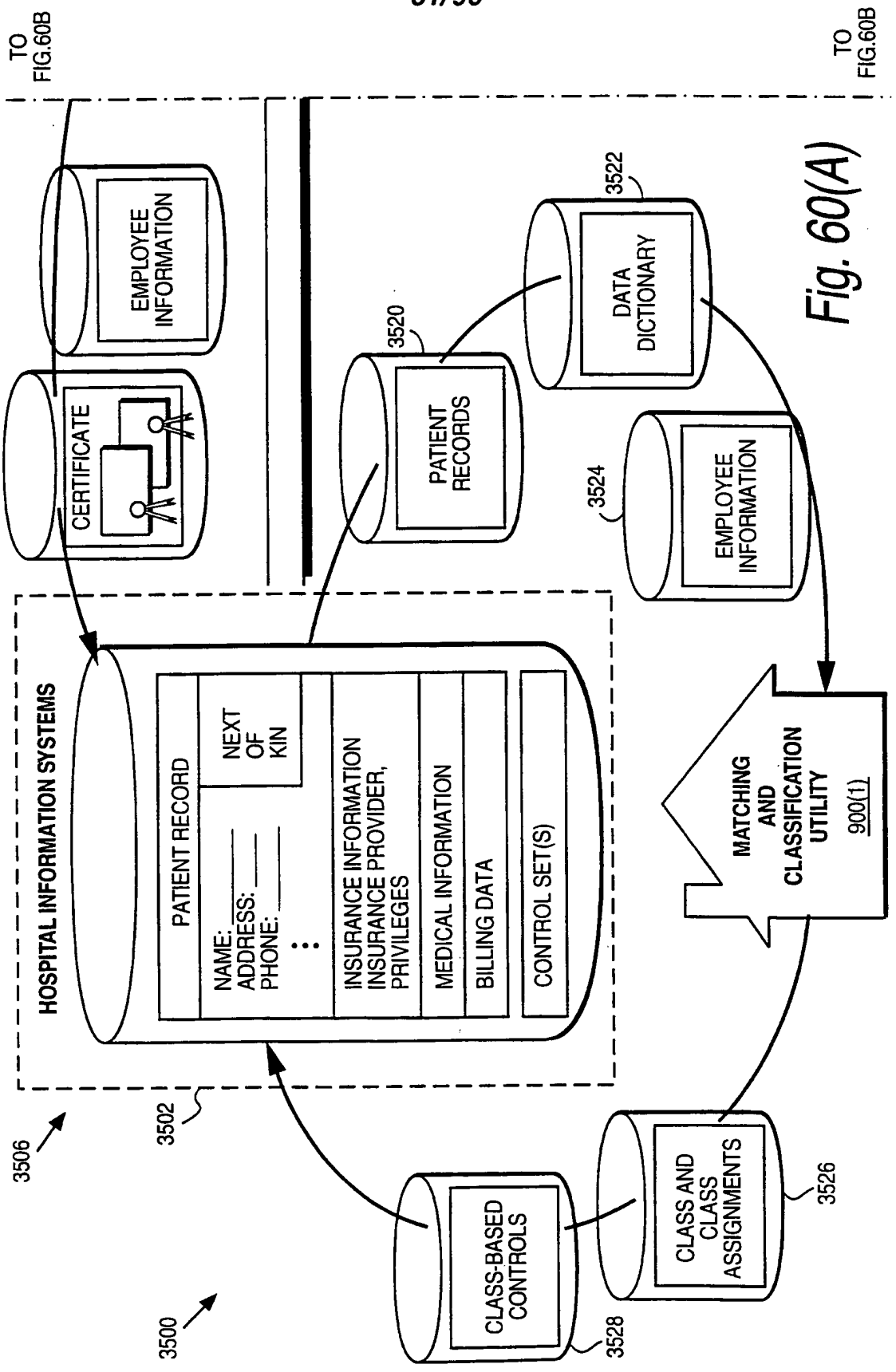


Fig. 60(A)

Petitioner Apple Inc. - Exhibit 1004, p. 5744

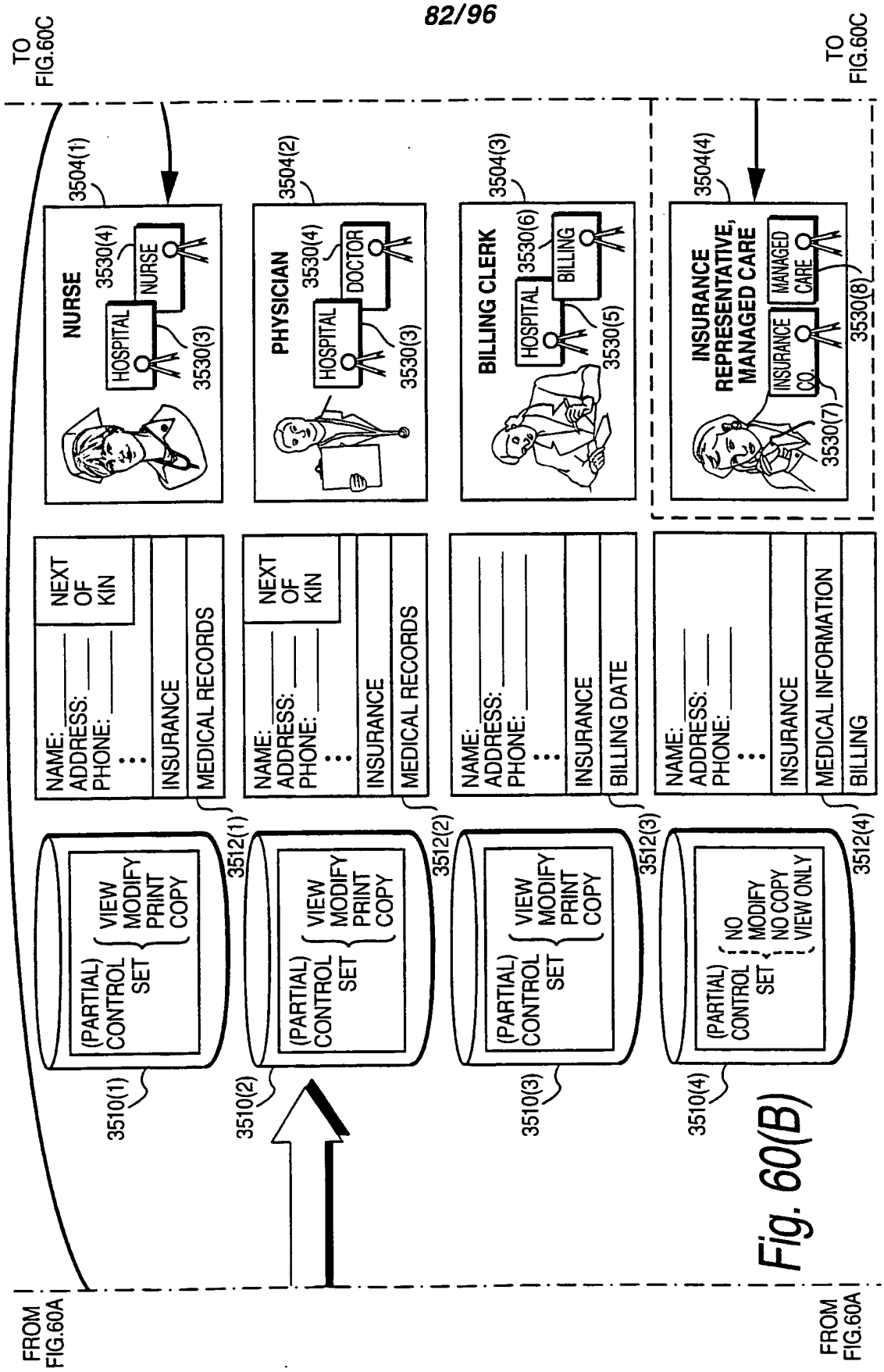


Fig. 60(B)

Petitioner Apple Inc. - Exhibit 1004, p. 5745

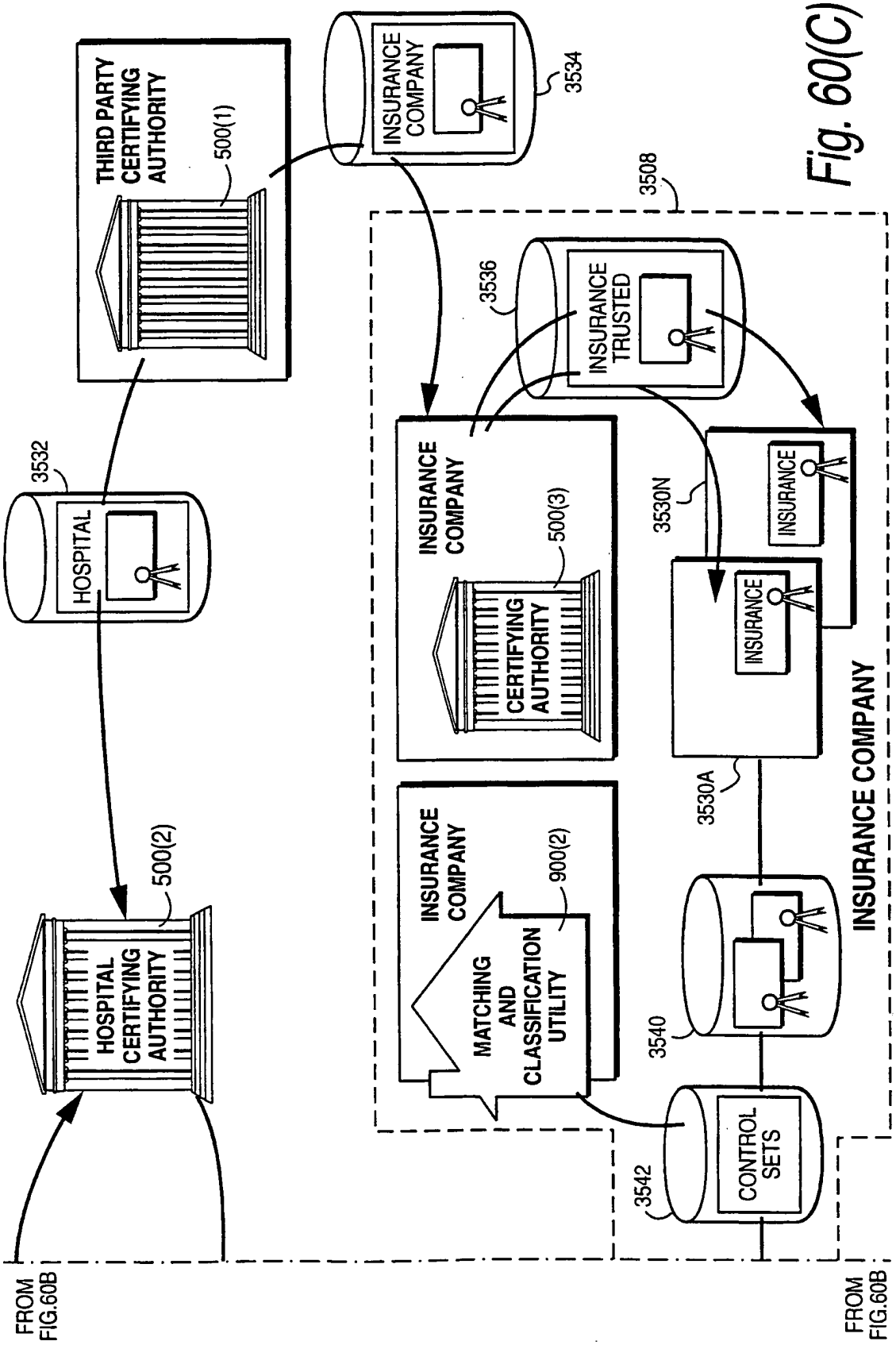


Fig. 60(C)

Petitioner Apple Inc. - Exhibit 1004, p. 5746

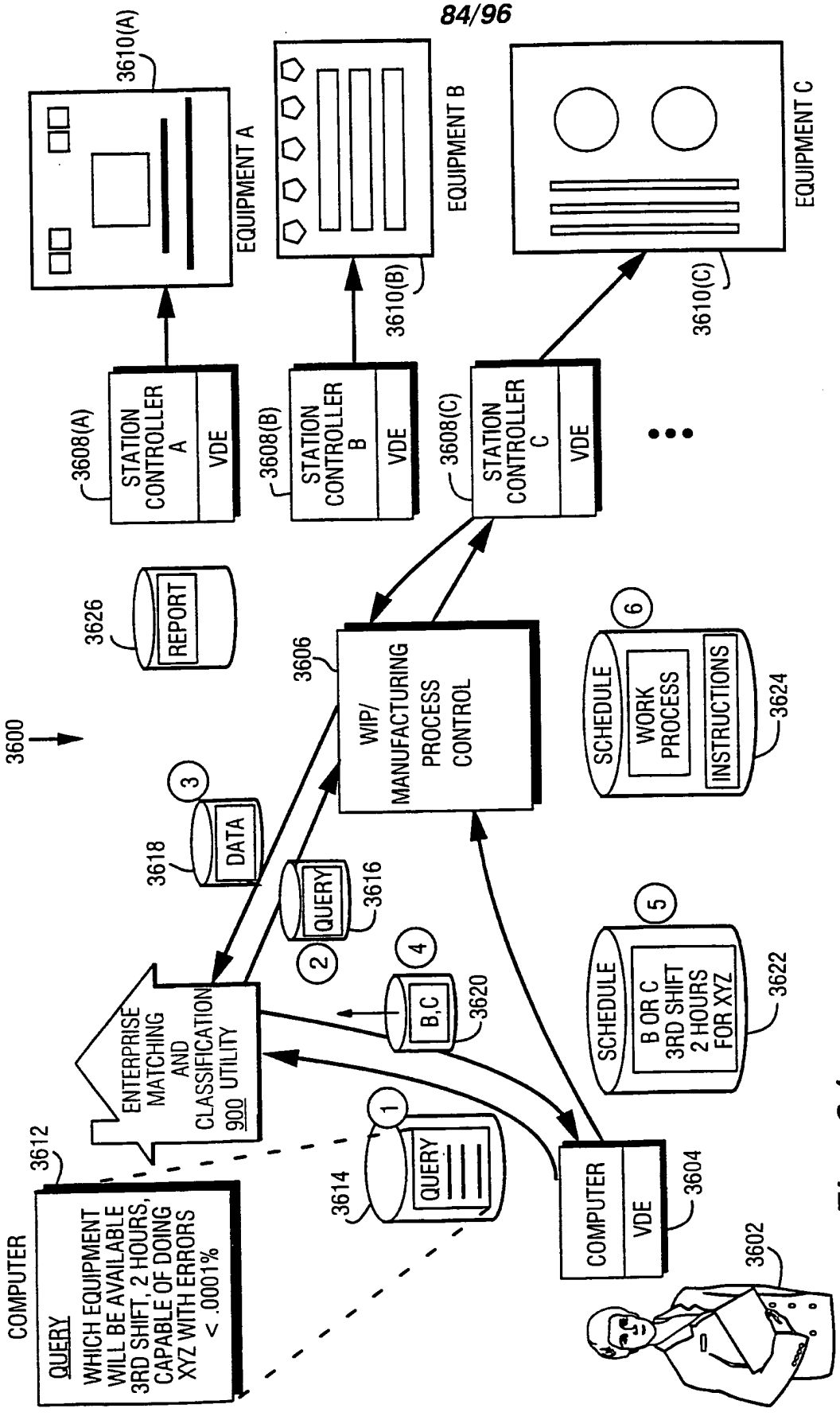


Fig. 61 Workflow Example

Petitioner Apple Inc. - Exhibit 1004, p. 5747

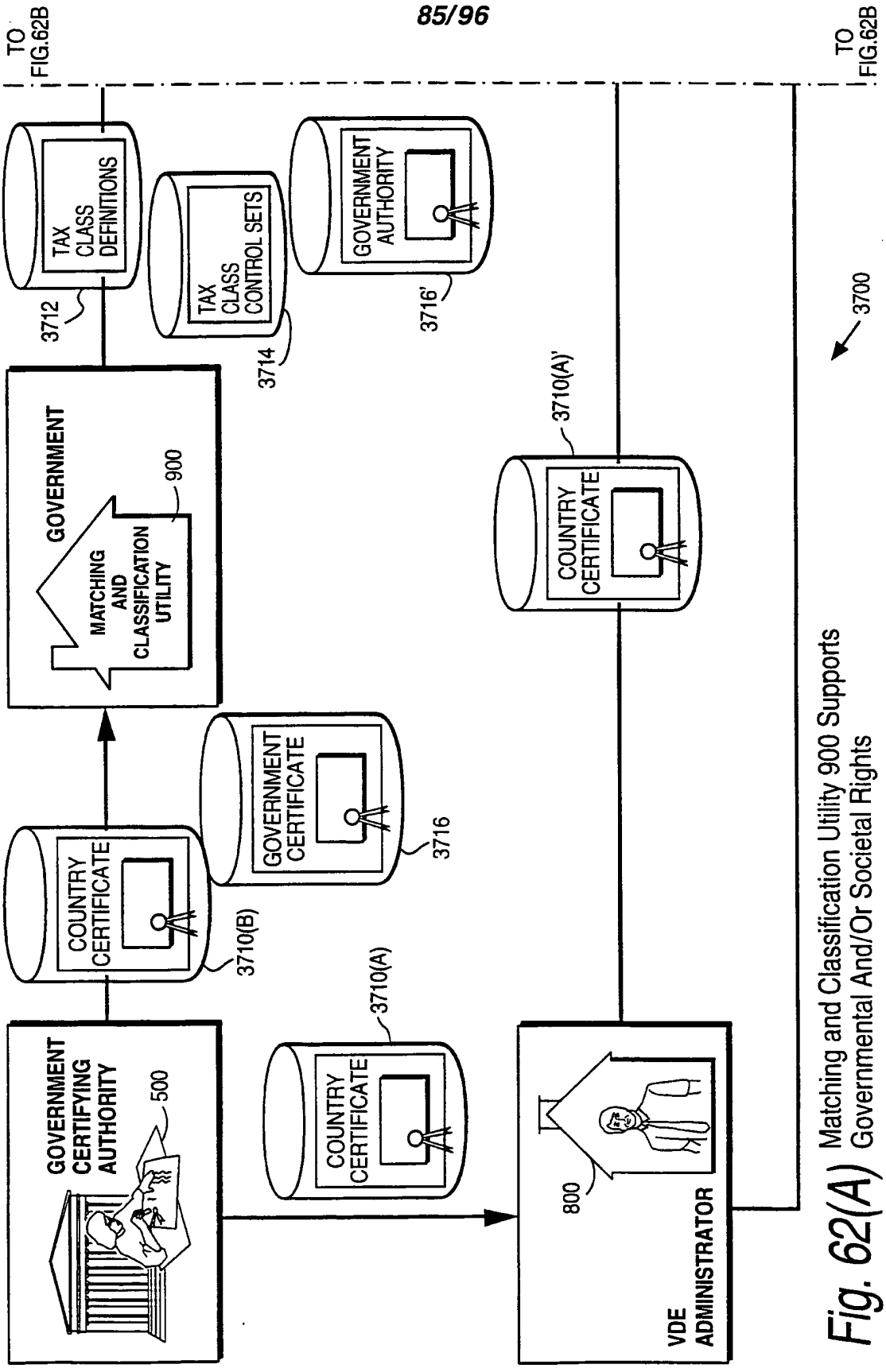


Fig. 62(A) Matching and Classification Utility 900 Supports Governmental And/Or Societal Rights

Petitioner Apple Inc. - Exhibit 1004, p. 5748

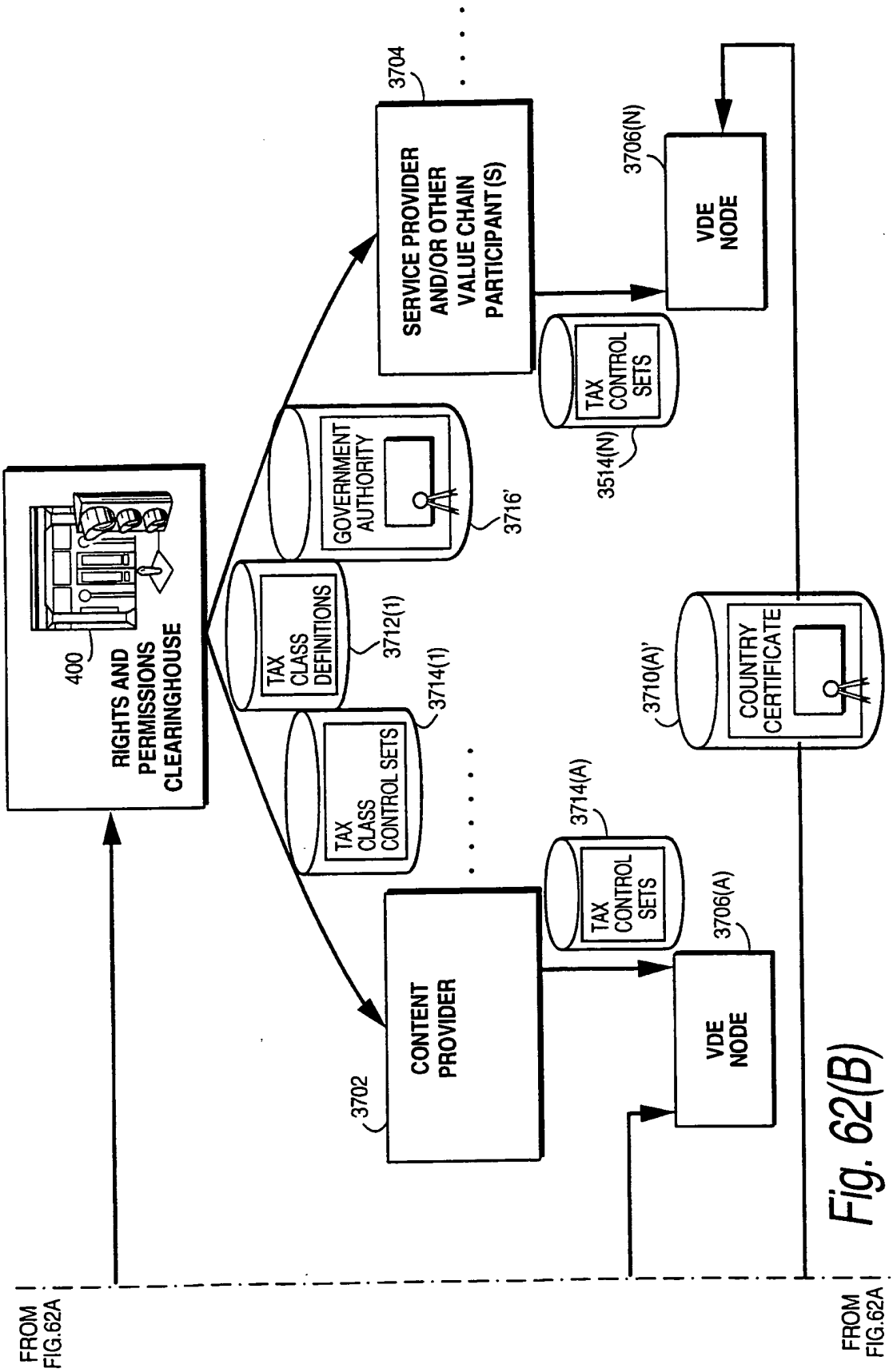


Fig. 62(B)

Petitioner Apple Inc. - Exhibit 1004, p. 5749

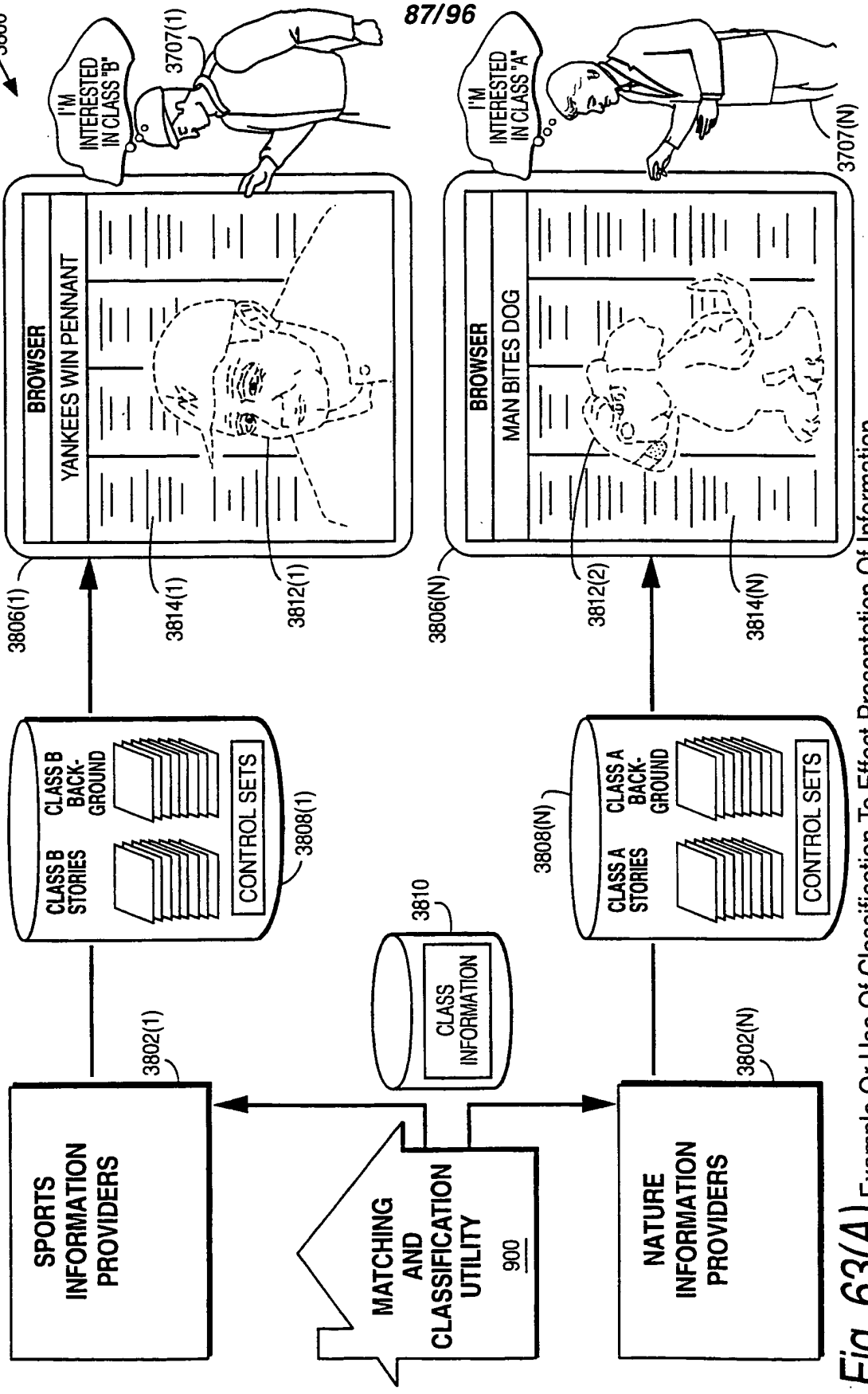


Fig. 63(A) Example Or Use Of Classification To Effect Presentation Of Information.

Petitioner Apple Inc. - Exhibit 1004, p. 5750

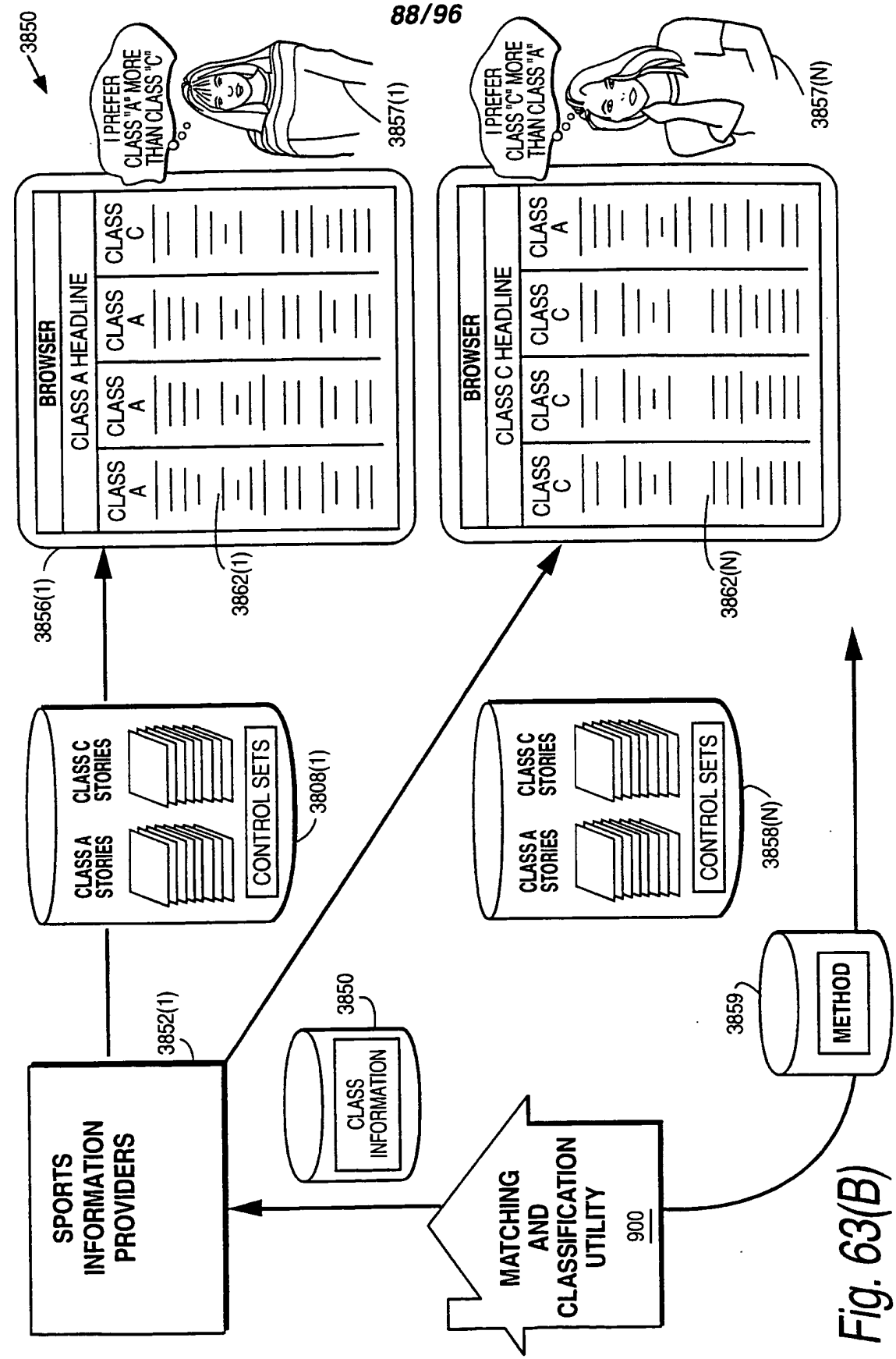


Fig. 63(B)

Petitioner Apple Inc. - Exhibit 1004, p. 5751

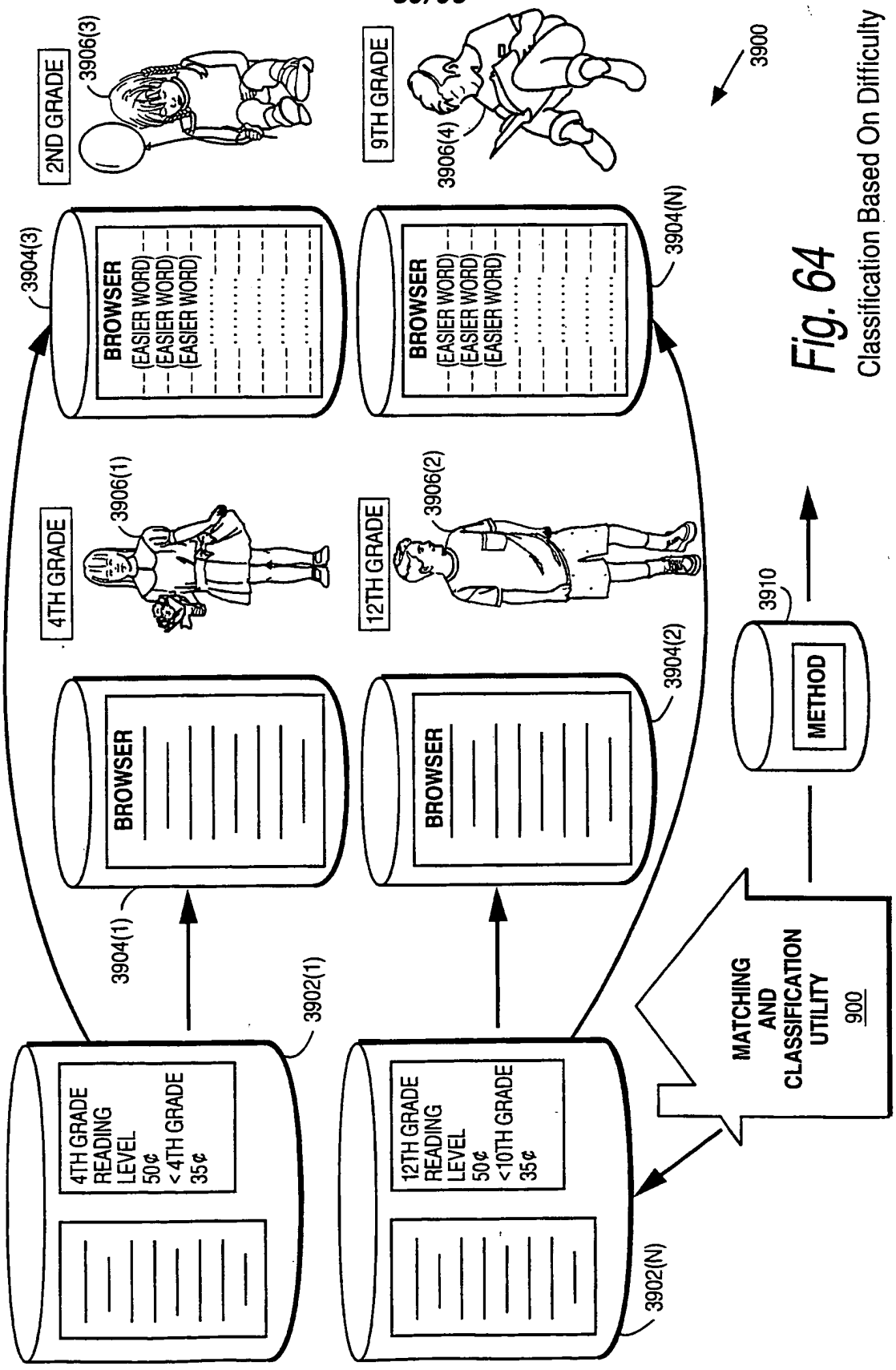


Fig. 64
Classification Based On Difficulty

Petitioner Apple Inc. - Exhibit 1004, p. 5752

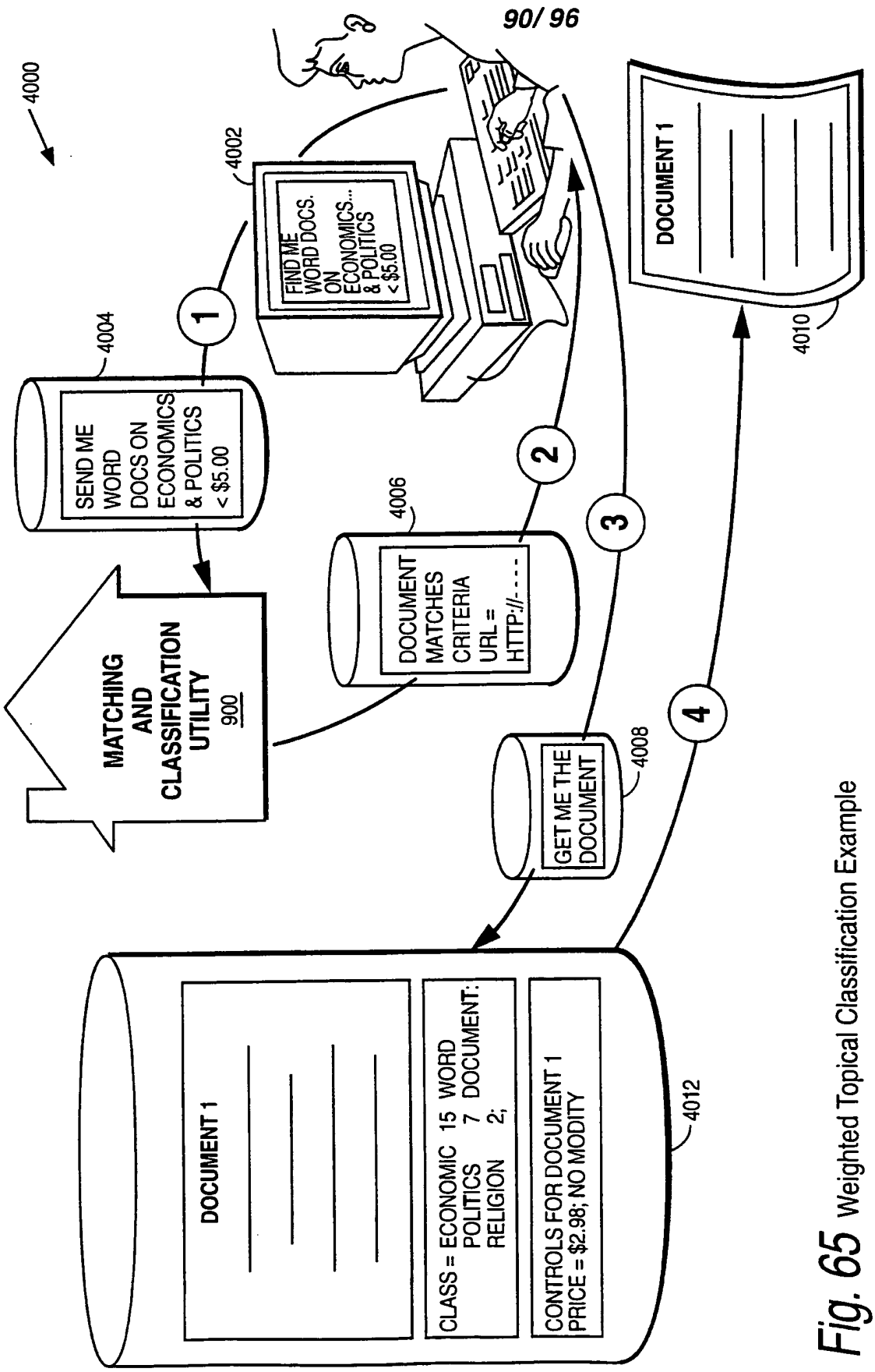


Fig. 65 Weighted Topical Classification Example

Petitioner Apple Inc. - Exhibit 1004, p. 5753

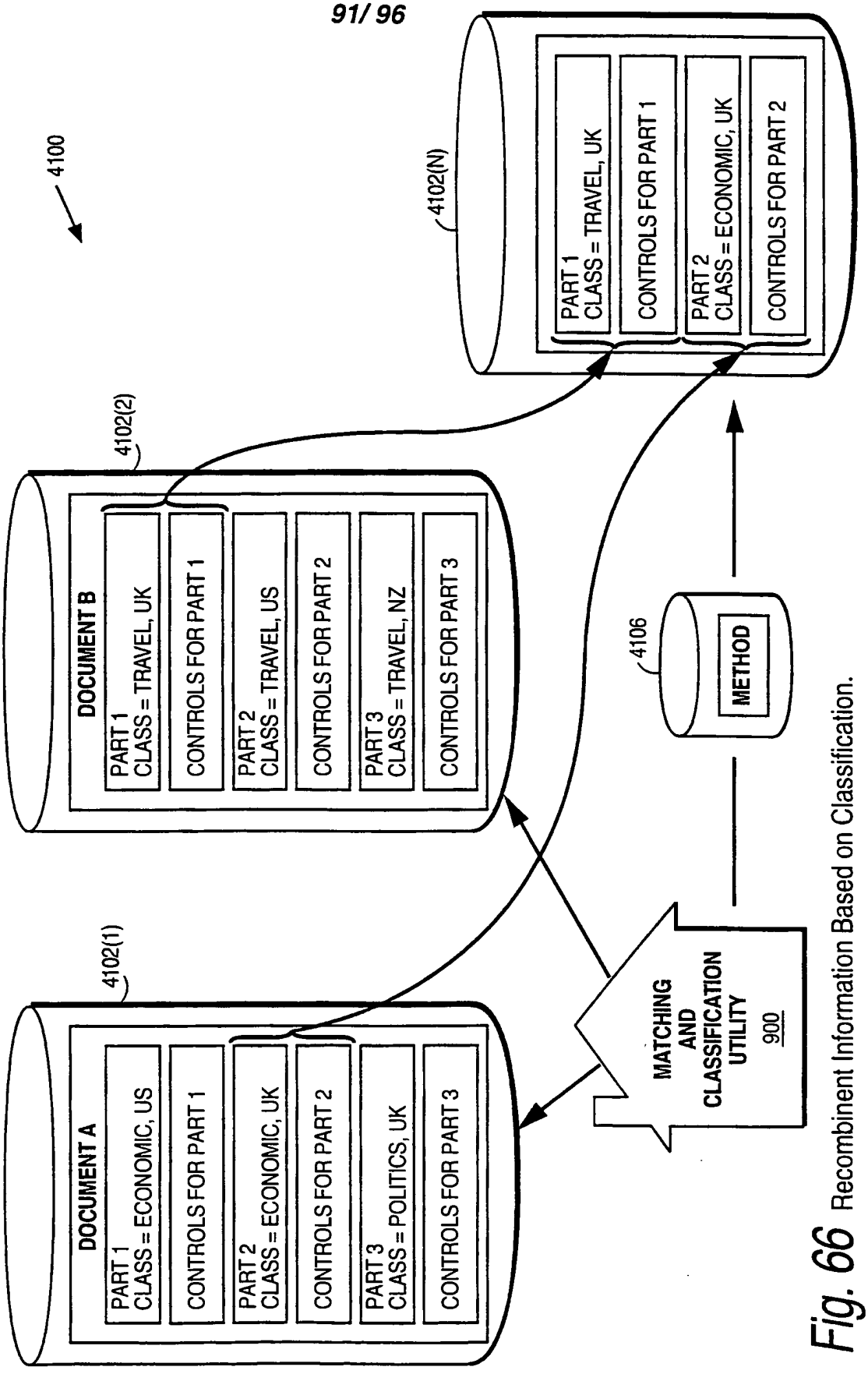


Fig. 66 Recombinent Information Based on Classification.

Petitioner Apple Inc. - Exhibit 1004, p. 5754

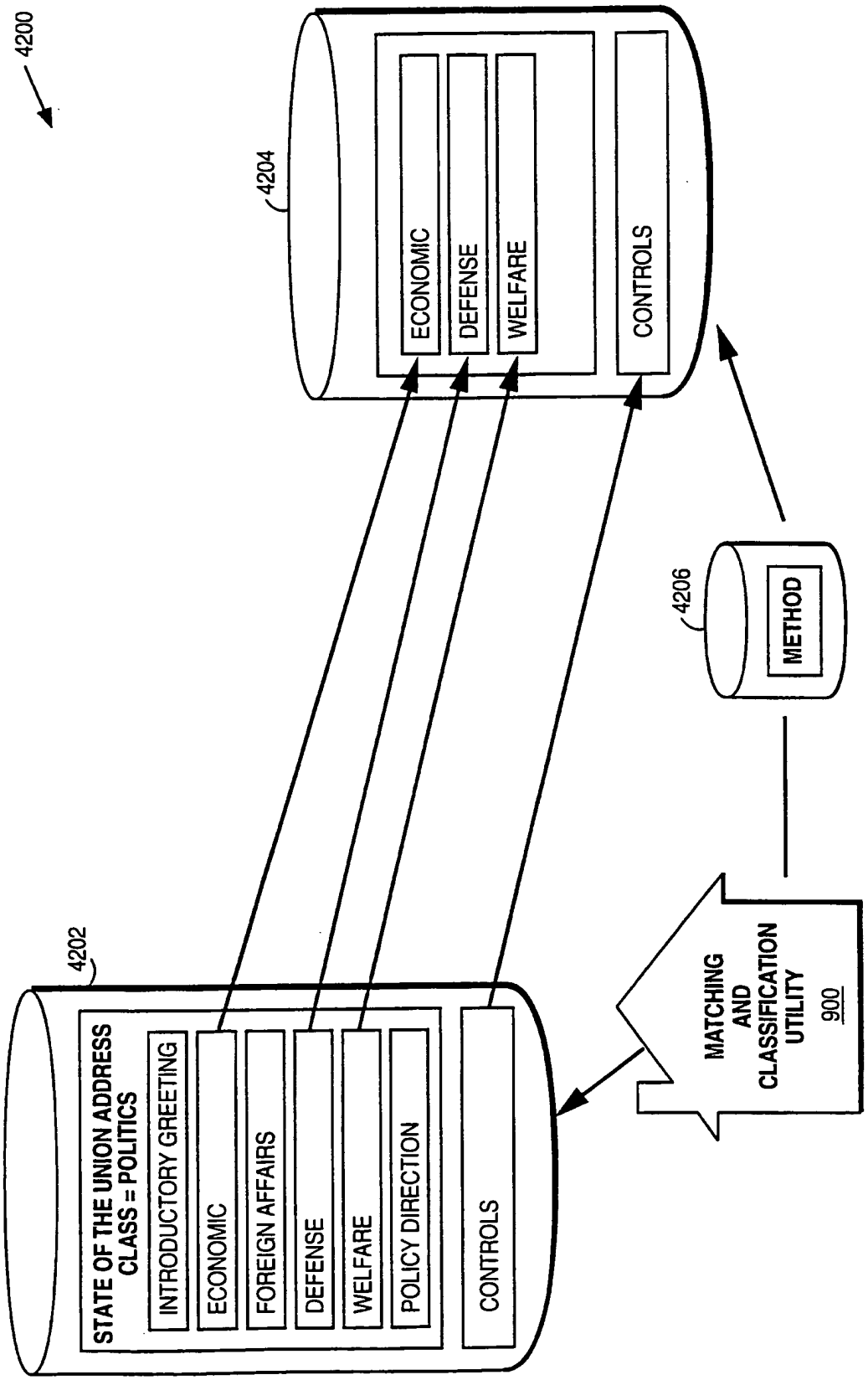
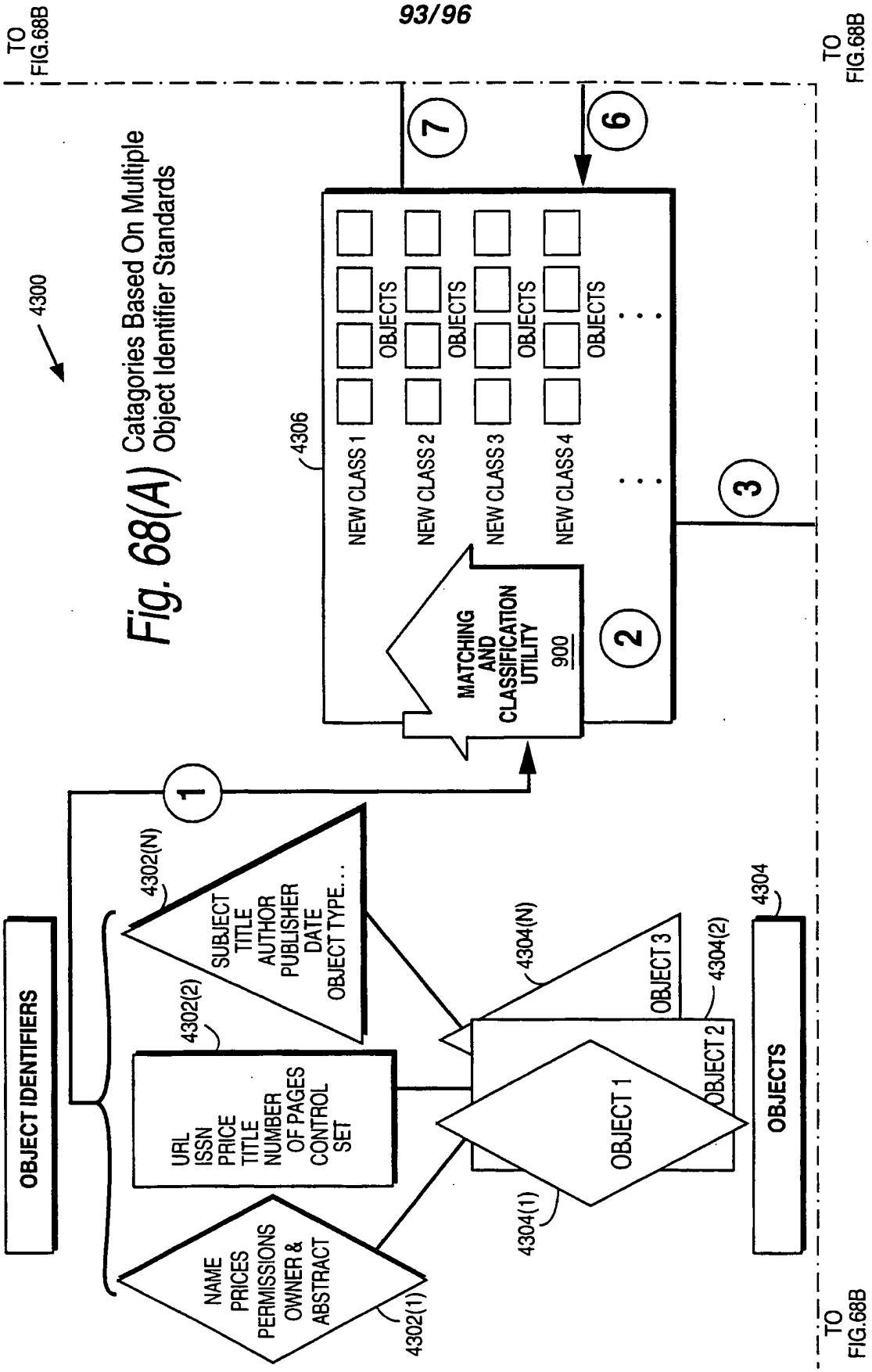


Fig. 67 Nested Classification.

Petitioner Apple Inc. - Exhibit 1004, p. 5755



Petitioner Apple Inc. - Exhibit 1004, p. 5756

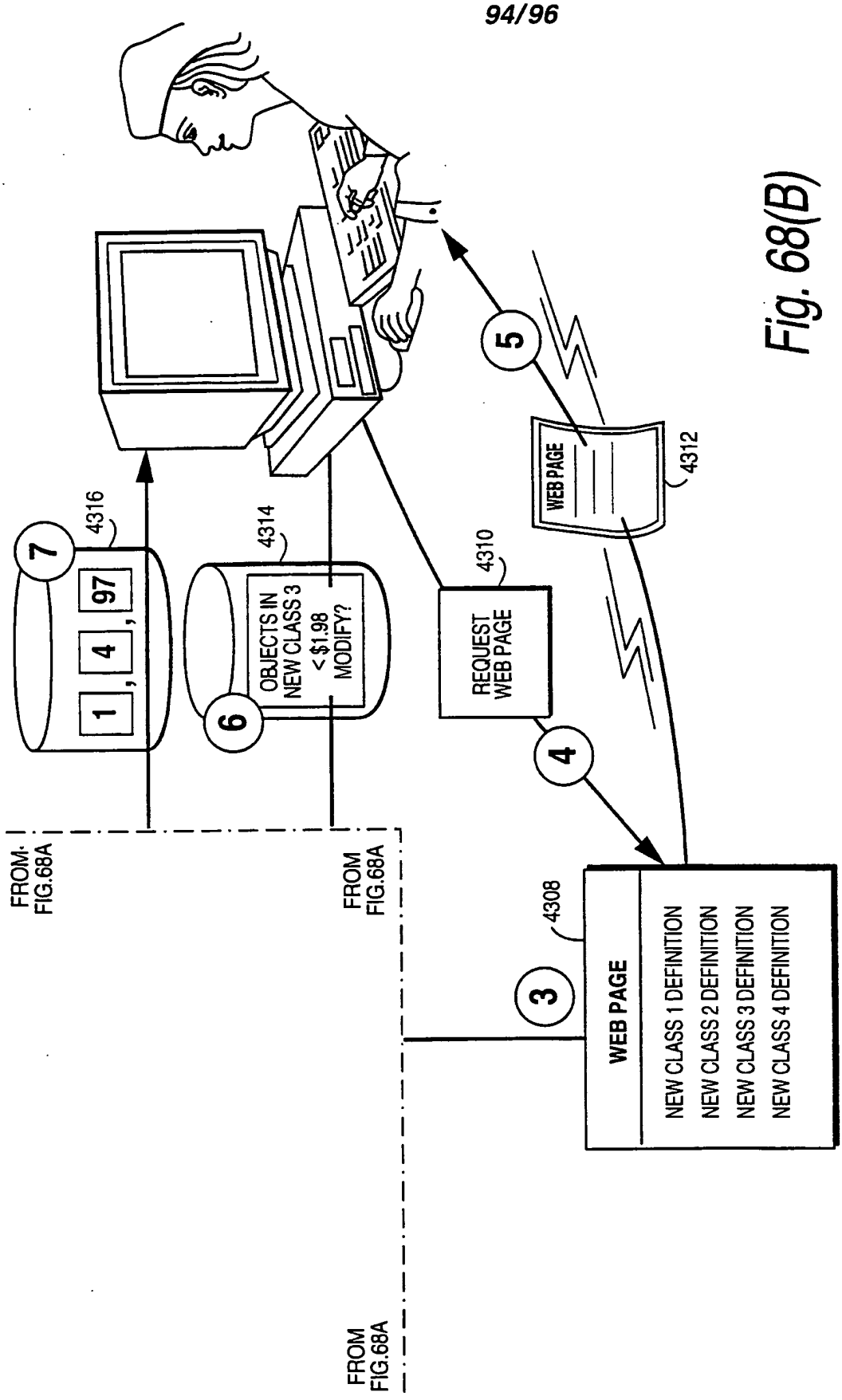


Fig. 68(B)

Petitioner Apple Inc. - Exhibit 1004, p. 5757

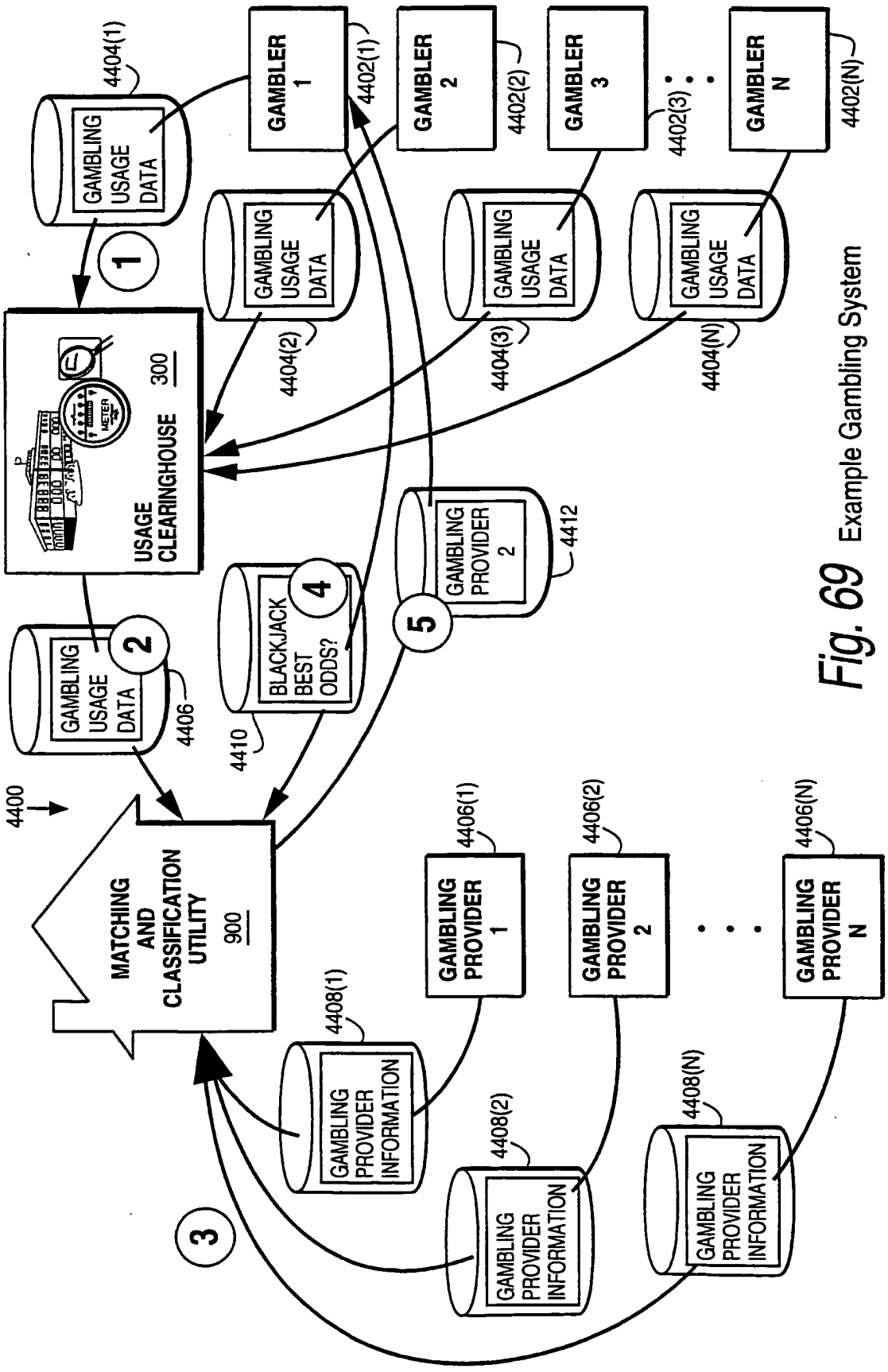


Fig. 69 Example Gambling System

Petitioner Apple Inc. - Exhibit 1004, p. 5758

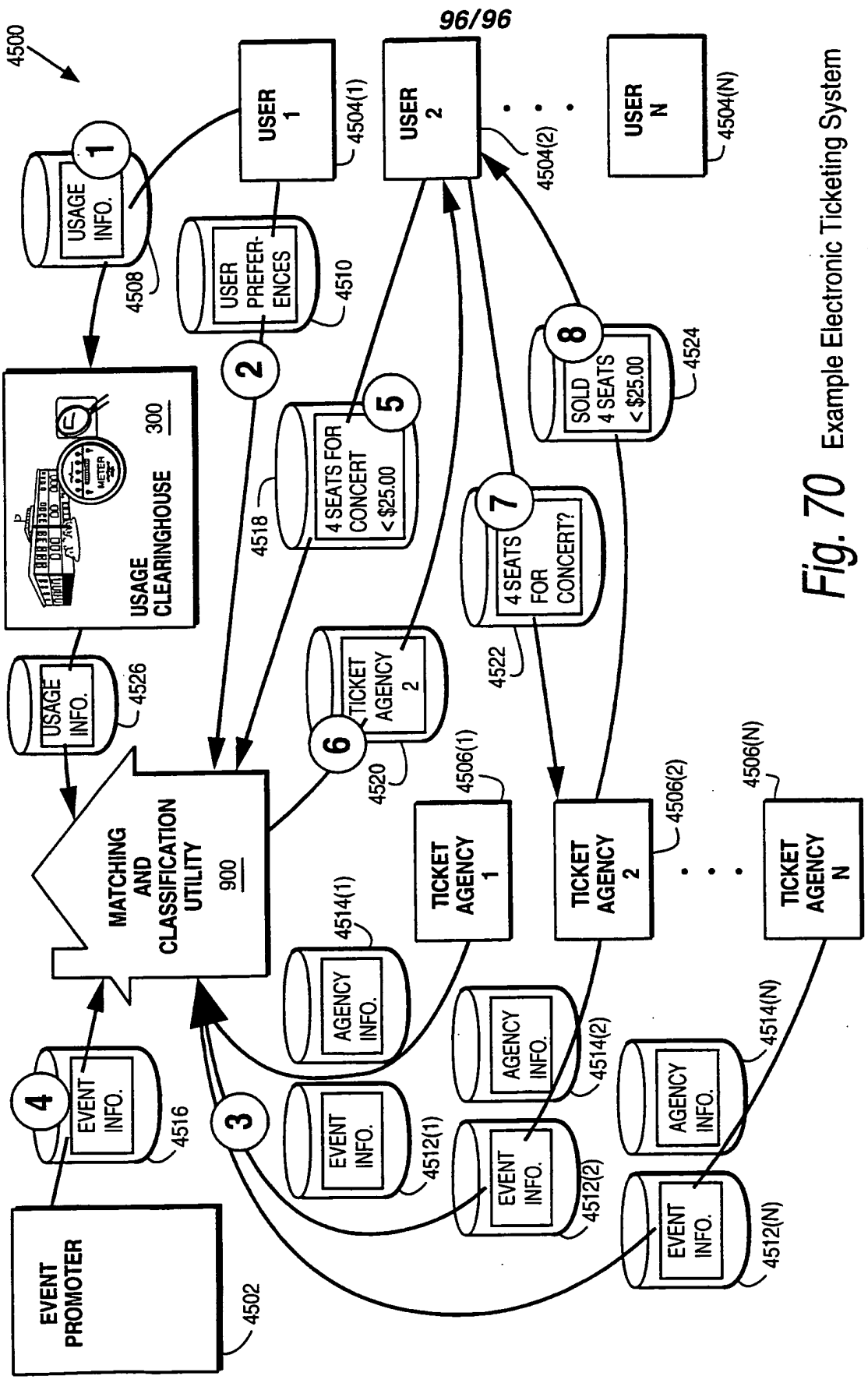


Fig. 70 Example Electronic Ticketing System

Petitioner Apple Inc. - Exhibit 1004, p. 5759

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 December 2000 (07.12.2000)

PCT

(10) International Publication Number
WO 00/73922 A2

(51) International Patent Classification[7]: G06F 17/00

(21) International Application Number: PCT/US00/11078

(22) International Filing Date: 25 April 2000 (25.04.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/323,635 1 June 1999 (01.06.1999) US

(71) Applicant: ENTERA, INC. [US/US]; 40971 Encyclopedia Circle, Fremont, CA 94538 (US).

(72) Inventor: SCHARBER, John, M.; 1616 Placer Circle, Livermore, CA 94550 (US).

(74) Agents: FAHMI, Tarek, N. et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 00/73922 A2

(54) Title: CONTENT DELIVERY SYSTEM

(57) Abstract: Disclosed is a network content delivery system configured to: select a first content routing technique for processing a first set of network content; and select a second content routing technique for processing a second set of network content, wherein the first and second content routing techniques are selected based on one or more content routing variables. Also disclosed is a content delivery system comprising: a network node for storing network content; a first transmission medium communicatively coupled to the network node for transmitting a first set of network content to the network node; and a second transmission medium communicatively coupled to the network node for transmitting a second set of network content to the network node, wherein the first and second sets of network content are selected based on one or more routing variables.

Petitioner Apple Inc. - Exhibit 1004, p. 5760

CONTENT DELIVERY SYSTEM
1

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to the transmission and storage of digital information across a network. More particularly, the invention relates to an improved system and method for caching and/or delivering various types of digital content using a plurality of network protocols.

Description of the Related Art

The World Wide Web (hereinafter "the Web") is a network paradigm which links documents known as "Web pages" locally or remotely across multiple network nodes (i.e., Web servers). A single Web page may have links (a.k.a., "hyperlinks") which point to numerous other Web pages. When a user points and clicks on a link using a cursor control device such as a mouse, the user can jump from the initial page to another page, regardless of where the Web pages are actually located. For example, the initial Web page might be stored on a Web server in New York and the second page (accessed via the hyperlink in the first page) might be stored on a Web server in California.

The underlying principles of the Web were developed 1989 at the European Center for Nuclear Research (CERN) in Geneva. By 1994 there were approximately 500 Web servers on the Internet. Today there are more than a million, with new sites starting up at an extraordinary rate. In sum, the Web has become the center of Internet activity and is the primary reason for the explosive growth of the Internet over the past several years.

In addition to providing a simple point-and-click interface to vast amounts of information on the Internet, the Web is quickly turning into a content delivery system. Well known Internet browsers such as Netscape Navigator™ and Microsoft Internet Explorer™ frequently provide plug-in software which allow additional features to be incorporated into the browser program. These include, for example, support for audio and video streaming, telephony, and videoconferencing.

The unparalleled increase in Web usage combined with the incorporation of high bandwidth applications (i.e., audio and video) into browser programs has created serious

performance/bandwidth problems for most Internet Service Providers (hereinafter "ISPs"). Moreover, the network traffic resulting from non-Web-based Internet services such as Internet News (commonly known as "Usenet" News) has increased on the same scale as the increase in Web traffic, thereby further adding to the bandwidth problems experienced by most ISPs.

These issues will be described in more detail with respect to **Figure 1** which illustrates an ISP 100 with a link 160 to a larger network 150 (e.g., the Internet) through which a plurality of clients 130, 120 can access a plurality of Web servers 140-144 and/or News servers 146-148. Maintaining a link 160 to the Internet 150 with enough bandwidth to handle the continually increasing traffic requirements of its clients 120, 130 represents a significant cost for ISP. At the same time, ISP 110 must absorb this cost in order to provide an adequate user experience for its clients 120, 130.

One system which is currently implemented to reduce network traffic across link 160 is a proxy server 210 with a Web cache 220, illustrated in **Figure 2**. When client 120 initially clicks on a hyperlink and requests a Web page (shown as address "www.isp.com/page.html") stored on Web server 144, client 120 will use proxy server 210 as a "proxy agent." This means that proxy server 210 will make the request for the Web page on behalf of client 120 as shown. Once the page has been retrieved and forwarded to client 120, proxy server will store a copy of the Web page locally in Web cache 220. Thus, when client 120 or another client – e.g., client 130 – makes a subsequent request for the same Web page, proxy server 210 will immediately transfer the Web page from its Web cache 210 to client 130. As a result, the speed with which client 130 receives the requested page is substantially increased, and at the same time, no additional bandwidth is consumed across Internet link 160.

While the foregoing proxy server configuration alleviates some of the network traffic across Internet link 160, several problems remain. One problem is that prior Web cache configurations do not have sufficient intelligence to deal with certain types of Web pages (or other Web-based information). For example, numerous Web pages and associated content can only be viewed by a client who pays a subscriber fee. As such, only those clients which provide proper authentication should be permitted to download the information. Today, proxy servers such as proxy server 210 will simply not cache a Web document which requires authentication.

In addition, Web caches do not address the increasing bandwidth problem associated with non-Web based Internet information. In particular, little has been done to alleviate the increasing bandwidth problems created by Usenet news streams. In fact, ISPs today must set aside a substantial amount of bandwidth to provide a continual Usenet news feed to its clients. Moreover, no mechanism is currently available for caching other data transmissions such as the streaming of digital audio and video. The term "streaming" implies a one-way transmission from a server to a client which provides for uninterrupted sound or video. When receiving a streaming transmission, the client will buffer a few seconds of audio or video information before it starts sending the information to a pair of speakers and/or a monitor, thus compensating for momentary delays in packet delivery across the network.

Accordingly, what is needed is a content delivery system which will reduce the bandwidth requirements for ISP 110 while still providing clients 120, 130 with an adequate user experience. What is also needed is a system which will work seamlessly with different types of Web-based and non-Web-based information and which can be implemented on currently available hardware and software platforms. What is also needed is an intelligent content delivery system which is capable of caching all types of Web-based information, including information which requires the authentication of a client before it can be accessed. What is also needed is a content delivery system which is easily adaptable so that it can be easily reconfigured to handle the caching of new Internet information and protocols. Finally, what is needed is a data replication system which runs on a distributed database engine, thereby incorporating well known distributed database procedures for maintaining cache coherency.

SUMMARY OF THE INVENTION

Disclosed is a network content delivery system configured to: select a first content routing technique for processing a first set of network content; and select a second content routing technique for processing a second set of network content, wherein the first and second content routing techniques are selected based on one or more content routing variables.

Also disclosed is a content delivery system comprising: a network node for storing network content; a first transmission medium communicatively coupled to the network node for transmitting a first set of network content to the network node; and a second transmission medium communicatively coupled to the network node for transmitting a second set of

network content to the network node, wherein the first and second sets of network content are selected based on one or more routing variables.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained from the following detailed description in conjunction with the following drawings, in which:

FIG. 1 illustrates generally a network over which an ISP and a plurality of servers communicate.

FIG. 2 illustrates an ISP implementing a proxy server Web cache.

FIG. 3 illustrates one embodiment of the underlying architecture of an Internet content delivery system node.

FIG. 4 illustrates a plurality of Internet content delivery system nodes communicating across a network.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

One embodiment of the present system is a computer comprising a processor and a memory with which software implementing the functionality of the internet content delivery system described herein is executed. Such a computer system stores and communicates (internally or with other computer systems over a network) code and data using machine readable media, such as magnetic disks, random access memory, read only memory, carrier waves, signals, etc. In addition, while one embodiment is described in which the parts of the present invention are implemented in software, alternative embodiments can implement one or more of these parts using any combination of software, firmware and/or hardware.

The underlying architecture of one embodiment of the present internet content delivery system (hereinafter "ICDS") is illustrated in **Figure 3**. A single ICDS node 300 is shown including a cache 330, an ICDS application programming interface (hereinafter "API") 360 which includes a distributed database engine 361, and a plurality of software modules 310-326

which interface with the ICDS API 360. ICDS node 300 may communicate over a network 340 (e.g., the Internet) over communication link 370 and may also interface with a plurality of clients 350-351 and/or other ICDS nodes (e.g., through link 380).

As is known in the art, an API such as ICDS API 360 is comprised of a plurality of subroutines which can be invoked by application software (i.e., software written to operate in conjunction with the particular API). Thus, in **Figure 3** each of the plurality of software modules 310-326 may be uniquely tailored to meet the specific needs of a particular ISP. The modules interface with API 360 by making calls to the API's set of predefined subroutines. Another significant feature of ICDS API 360 is that it is platform-independent. Accordingly, it can be implemented on numerous hardware platforms including those that are Intel-based, Macintosh-based and Sun Microsystems-based.

In one embodiment, a portion of API 360's subroutines and a set of prefabricated modules can be marketed together as a Software Development Kit (hereinafter "SDK"). This will allow ISPs, corporations and/or end-users to customize the type of internet content delivery/caching which they require. In addition, because modules 310-326 may be dynamically linked, they may be loaded and unloaded without having to reboot the hardware platform on which cache 330 is executed.

I. Distributed Content Processing

As illustrated, ICDS node 300 includes a plurality of network protocol modules 310-319 which interface with API 360. These modules provide caching support on ICDS node 300 for numerous different Internet protocols including, but not limited to, Web protocols such as the Hypertext Transfer Protocol (hereinafter "HTTP") 310, Usenet news protocols such as the Network News Transport Protocol (hereinafter "NNRP") 312, directory access protocols such as the Lightweight Directory Access Protocol (hereinafter "LDAP") 314, data streaming protocols such as the Real Time Streaming Protocol (hereinafter "RTSP") 316, and protocols used to perform Wide Area Load Balancing (hereinafter "WALB") 318. Because the underlying architecture of the present ICDS system includes an open API, new protocol modules (e.g., module 319) can be seamlessly added to the system as needed.

One embodiment of the ICDS system includes a plurality of standardized service definitions through which individual service modules 320-326 may be configured to interface

with the ICDS API 360. These service modules provide the underlying functionality of ICDS node 300 and may include a data services module 320, an access services module 322, a transaction services module 323, a commercial services module 324, a directory services module 325, and a resource services module 326. The functionality of each of these modules will be described in more detail below.

In one embodiment of the ICDS system, the ICDS API includes a distributed relational database engine 361. As a result, a plurality of ICDS nodes 410-440 can be distributed across ISP 400's internal network and still maintain a coherent, up-to-date storage of Internet content. For example, if a particular data object is updated at two nodes simultaneously, the underlying distributed database system may be configured to resolve any conflicts between the two modifications using a predefined set of distributed database algorithms. Accordingly, the present system provides built in caching support for dynamically changing Internet content (e.g., Web pages which are modified on a regular basis). Such a result was not attainable with the same level of efficiency in prior art caching systems such as proxy server 210 of **Figure 2** (which are executed on, e.g., standard flat file systems such as UNIX or NFS file servers).

Data Services

Data services modules such as module 320 running on each ICDS node 410-440 provide support for data replication and distribution across ISP 400's internal network 480. This includes caching support for any data protocol included in the set of protocol modules 310-319 shown in **Figure 3** as well as for any future protocol which may be added as a module to the ICDS API 360. Because the ICDS API 360 provides a set of standardized service definitions for data services module 320, an ISP using a plurality of ICDS nodes 300 as illustrated in **Figure 4** can replicate data across its network without an extensive knowledge of distributed database technology. In other words, the ISP can configure its plurality of nodes by invoking the standardized service definitions associated with data services module 320 and leave the distributed database functionality to the distributed database engine 361.

Generally, three different types of data replication may be implemented by the present system: dynamic replication, database replication (or "actual" replication), and index replication. Using dynamic replication, if client 472 requests content from internal ICDS server 460 or from a server across network 490, the content will be delivered to client 472 and replicated in ICDS node 430. If client 473 (or any other client) subsequently requests the

same content, it will be transmitted directly from ICDS node 430 rather than from its original source (i.e., a second request to server 460 or a server across network 490 will not be required). Accordingly, bandwidth across ISP 400's internal network and across Internet link 405 is conserved.

The dynamic replication mechanism just described works well for replicating static content but not for replicating dynamically changing content. For example, if the replicated content is a magazine article then caching a copy locally works well because it is static information – i.e., there is no chance that the local copy will become stale (out of date). However, if the replicated content is a Web page which contains continually changing information such as a page containing stock market quotes, then dynamic replication may not be appropriate. No built in mechanism is available for proxy cache server 210 to store an up-to-date copy of the information locally.

The present ICDS system, however, may use database replication to maintain up-to-date content at each ICDS node 410-440. Because the present system includes a distributed database engine 361, when a particular piece of content is changed at one node (e.g., ICDS server 460) a store procedure may be defined to update all copies of the information across the network. This may be in the form of a relational database query. Thus, the present system may be configured to use dynamic replication for static content but to use database replication for time-sensitive, dynamically changing content.

The third type of database replication is known as index replication. Using index replication a master index of content is replicated at one or more ICDS nodes 410-440 across the network 480. Once again, this implementation is simplified by the fact that the underlying ICDS node engine is a distributed database engine. Certain types of information distributions are particularly suitable for using index replication. For example, news overview information (i.e., the list of news articles in a particular newsgroup) is particularly suited to index replication. Instead of replicating each individual article, only the news overview information needs to be replicated at various nodes 410-440 across the network 480. When a client 473 wants to view a particular article, only then will the article be retrieved and cached locally (e.g., on ICDS node 430).

ICDS node 430 is capable of caching and delivering various types of Internet data using any of the foregoing replication techniques. While prior art proxy servers such as proxy server 210 may only be used for caching Web pages, ICDS node 430 is capable of caching various other types of internet content (e.g., news content) as a result of the protocol modules 310-319 interfacing with ICDS API 360. Moreover, as stated above, ICDS node 430 (in conjunction with nodes 410, 420 and 440) may be configured to cache dynamic as well as static Web-based content using various distributed database algorithms.

One specific example of a data service provided by one embodiment of the present system is Wide Area Load Balancing (hereinafter "WALB") using layer 7 switching as described in the co-pending U.S. Patent Application entitled "WIDE AREA LOAD BALANCING" (Serial No. _____), which is assigned to the assignee of the present application and which is incorporated herein by reference. The present system may also perform dynamic protocol selection, dynamic query resolution, and/or heuristic adaptation for replicating content across a network as set forth in the co-pending U.S. Patent Application entitled Dynamic Protocol Selection and "QUERY RESOLUTION FOR CACHE SERVERS" (Serial No. ____), which is assigned to the assignee of the present application and which is incorporated herein by reference. Finally, the present system also may include network news (e.g., Usenet news) services set forth in the co-pending U.S. Patent Applications entitled "HYBRID NEWS SERVER" (Serial No. ___), and "SELF-MODERATED VIRTUAL COMMUNITIES" (Serial No. ___), each assigned to the assignee of the present application and each incorporated herein by reference.

Access Services

As stated above, prior art proxy server cache systems such as proxy server 210 are only capable of caching static, publicly available Web pages. A substantial amount of Web-based and non-Web-based content, however, requires some level of authentication before a user will be permitted to download it. Thus, client 472 (in Figure 4) may pay a service fee to obtain access to content on a particular web site (e.g., from server 460 or from another server over network 490). As a result, when he attempts to access content on the site he will initially be prompted to enter a user name and password. Once the user transmits this information to the Web server, he will then be permitted to download Web server content as per his service agreement.

A problem that arises, however, is that prior art cache systems such as proxy server 210 are not permitted to cache the requested content. This is because proxy server 210 has no way of authenticating subsequent users who may attempt to download the content. Thus, documents which require authentication are simply uncacheable using current network cache systems.

The present ICDS system, however, includes user authentication support embedded in access services module 322. Thus, when client 473, for example, attempts to access a Web page or other information which requires authentication, ICDS node will determine whether the requested content is stored locally. If it is, then ICDS node 430 may communicate with the authentication server (e.g., server 460 or any server that is capable of authenticating client 473's request) to determine whether client 473 should be granted access to the content. This may be accomplished using standardized authentication service definitions embedded in access services module 322. Using these definitions, ICDS node 430 will not only know what authentication server to use, it will also know what authentication *protocol* to use when it communicates to the authentication server. As a result of providing local access services module 322 for authentication, network information which requires authentication can now be cached locally in ICDS node 430, thereby conserving additional bandwidth across network link 405 and/or ISP network 480.

One particular embodiment of the present system replicates Remote Authentication Dial In User Service (hereinafter "RADIUS") information across network 480. RADIUS is an application-level protocol used by numerous ISP's to provide user authentication and profile services. This is achieved by setting up a central RADIUS server with a database of users, which provides both authentication services (i.e., verification of user name and password) and profile services detailing the type of service provided to the user (for example, SLIP, PPP, telnet, rlogin).

Users connect to one or more Network Access Servers (hereinafter "NASs") which operate as a RADIUS clients and communicate with the central RADIUS server. The NAS client passes the necessary user information to the central RADIUS server, and then acts on the response which is returned. RADIUS servers receive user connection requests, authenticate users, and then return all configuration information necessary for the client to deliver service to the user.

One problem associated with the RADIUS protocol is that it does not provide any built in facilities for replication of RADIUS information. Accordingly, on large ISP's such as America Online ("AOL"), which may have tens of millions of users, RADIUS servers are hard hit, potentially handling thousands of logon requests a minute. This may create severe performance/bandwidth problems during high traffic periods. In response, some ISP's have taken a brute-force approach to distributing RADIUS information by simply copying the information to additional servers across the network without any built in mechanism to keep the RADIUS data coherent and up-to-date.

One embodiment of the present ICDS system provides an efficient, dynamic mechanism for distributing RADIUS information. Specifically, a RADIUS module is configured to interface with ICDS API 360 in this embodiment (similar to the way in which protocol modules 310-319 interface with the ICDS API 360). RADIUS information can them be seamlessly distributed across the system using distributed database engine 361. For example, the RADIUS module in conjunction with access services module 322 on ICDS node 430 may maintain radius information for local users. [Exactly how will this work? I assume that access services module will be used but there will be a separate RADIUS protocol module to support the protocol??] Thus, when client 472 first logs in to the system, ICDS node 430 may communicate with a second ICDS node (e.g., central ICDS server 460) which contains the necessary RADIUS authentication and user profile information. Client 472 will input a user name and password and will then be permitted access to the network as per his service agreement with ISP 400.

Unlike previous RADIUS systems, however, ICDS node 430 in the present embodiment may locally cache client 472's RADIUS information so that the next time client 472 attempts to login to the network, the information will be readily available (i.e., no access to a second ICDS node will be necessary). ICDS node 430 may be configured to save client 472's RADIUS information locally for a predetermined period of time. For example, the information may be deleted if client 472 has not logged in to local ICDS node 430 for over a month.

Thus, if client 472 represents a user who frequently travels across the country and logs in to ISP 400's network 480 from various different ICDS nodes, the present system provides a quick, effective mechanism for dynamically replicating client 472's user information into

those geographical locations from which he most commonly accesses ISP 400. This reduces the load which would otherwise be borne by a central RADIUS server and also improves client 472's user experience significantly (i.e., by providing him with a quick login).

Database replication can also be used to update RADIUS information distributed across multiple ICDS nodes 410-440. This may be done using known store procedures defined in relational database 361. For example, if client 472 cancels his service agreement with ISP 400, he should not be able to continually log in to local ICDS node 430 using the RADIUS information which has been cached locally. Thus, under the present ICDS system, ISP 400 may simply issue a relational database query such as [let's add another update query here using database terminology as an example] to immediately update ICDS node 430's radius information.

One of ordinary skill in the art will readily recognize from the preceding discussion that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention. Throughout the foregoing description, specific embodiments of the ICDS system were described using the RADIUS protocol in order to provide a thorough understanding of the operation of the ICDS system. It will be appreciated by one having ordinary skill in the art, however, that the present invention may be practiced without such specific details. For example, the ICDS system may also distribute authentication and user profile information in the form of the Lightweight Directory Access Protocol ("LDAP"). In other instances, well known software and hardware configurations/techniques have not been described in detail in order to avoid obscuring the subject matter of the present invention.

Access services module 322 may also provide local encryption/decryption and watermarking of internet content. Audio or video content delivery systems, for example, commonly use encryption of content to protect the rights of the underlying copyright holder. When a user requests a particular piece of content some delivery systems encrypt the content using a unique client encryption key. Only a client who possesses the encryption key (presumably the client who paid for the content) will be permitted to play the content back. Other systems provide for the "watermarking" of content (rather than encrypting) so that the rightful owner of the content may be identified. This simply entails embedding a unique "tag"

which identifies the source of the content and/or the owner of the content (i.e., the one who paid for it).

Prior art caching systems such as proxy server 210 are not capable of dealing with encrypted or watermarked content because the encryption/watermarking functionality was not provided locally (i.e., proxy server 210 was not "smart" enough). In one embodiment of the present ICDS system, however, access services module 322 of ICDS nodes 410-440 includes a local encryption module and/or a local watermarking module. For example, if client 473 requests specific content such as a copyrighted music content stored on a music server (e.g., ICDS server 460), the initial request for the content will be made from ICDS node 430 on behalf of client 473. ICDS node 430 will retrieve the requested content and cache it locally. If the requested content requires encryption, ICDS node 430 will use its local encryption module to encrypt the requested content using a unique user encryption key for client 473.

If a second client – e.g., client 472 – requests the same content, the copy stored locally on ICDS module 430 can be used. ICDS module 430 will simply encrypt the content using a *different* encryption key for user 472. Thus, frequently requested multimedia content (which, as is known in the art, can occupy a substantial amount of storage space) may be cached locally at ICDS node 430 notwithstanding the fact that the content requires both user authentication and encryption.

The same functionality may be provided for watermarking of content. ICDS node will use a watermarking module (which may comprise a component of access services module 322) to individually watermark multimedia information requested by individual clients, thereby protecting the rights of the copyright holder of the underlying multimedia content. This information can then be regularly communicated back to a central database repository.

As is known in the art, multimedia files can be extremely large and, accordingly, take substantially more time to communicate across a network than do, for example, generic Web pages. As such, the ability to locally cache multimedia files significantly reduces traffic across network 480, and also significantly improves the user experience for local users when downloading multimedia information.

Transaction Services

In addition to replicating data services and access services information across a network, the present ICDS system also provides for the replication of transaction services. Transaction services includes maintaining information on client payments for use of ISP 400's services as well as information relating to the client's online access profile (i.e., recording of the times when the user is online).

When a client logs in to an ISP today, the client's online information is maintained on a single central server. The central server maintains records of when and for how long the client logged in to the network and may also include information about what the client did while he was online. As was the case with maintaining a central RADIUS server, maintaining a central transaction server for all users of a large ISP is inefficient and cumbersome. The present system solves the performance and bandwidth problems associated with such a configuration by storing transaction information locally via transaction module 323 and algorithms build around distributed database engine 361.

Thus, if client 472 only logs on to ISP 400's network 480 via ICDS node 430, all of his transaction and billing information will be stored locally. The information may then be communicated across network 480 to a central billing server at predetermined periods of time (e.g., once a month). [We didn't go into great detail on transaction services and the rest; please add information as you feel appropriate]

Commercial Services

Commercial services module 324 provides a significantly improved local caching capability for add rotation and accounting. An add rotation system operating in conjunction with a typical proxy cache server will now be described with respect to **Figure 2**. When client 120 downloads a web page from Web server 142 the Web page may contain an ad tag or an add tag may automatically be inserted. The add tag will identify add server 170 and will indicate that an add should be inserted into the requested Web page from add server 170. Add server will then identify a specific add to insert into the downloaded Web page from add content server. The Web page plus the inserted add will then be forwarded to proxy server 210 and on to client 120.

Add server 170 will keep an accounting of how many different users have downloaded Web pages with adds inserted as described above. However, one problem with accounting on this system is that proxy server 210 requests Web pages *on behalf of* its clients. Accordingly, once the requested Web pages has been cached locally in Web cache 220, add server will only receive requests from proxy server 210 for any subsequent requests for the Web page. This will result in an inaccurate accounting of how many unique clients actually requested the Web page (and how many adds were viewed by unique users).

Because one embodiment of the present system provides built in caching support for ad rotation services, an accurate accounting of the number of hits that a particular ad receives may be maintained. More specifically, one embodiment of the present ICDS system solves this problem by providing a commercial services module that monitors and records how often individual clients request Web pages containing particular adds from add content server 171. This information than then be communicated to a central server (e.g., ICDS server 460) at predetermined intervals for generating add rotation usage reports.

Directory Services

Directory services provide the ability to cache locally a directory of information across network 480 or 490. That is, the question here is not whether the particular information is available but where exactly over networks 480 or 490 it is located. It should be noted that there may be some overlap between the directory services concept and the index replication concept described above with respect to data services. [I'm still not 100% sure what this is – please elaborate with an example]

II. Content Routing

The term "content routing" refers to the ability to select among various techniques/protocols for maintaining a coherent set of content across a network. The selection of a particular technique may be based on several routing variables including, but not limited to, the type of content involved (i.e., FTP, HTTP . . . etc), the size of the content involved (i.e., small files such as HTTP vs. large files such as audio/video streaming), the location of the content on network 480 and/or network 490, the importance of a particular piece of content (i.e., how important it is that the content be kept up-to-date across the entire network), the particular user requesting the content and the terms of his subscription agreement (i.e., some users may be willing to pay more to be insured that they receive only the most up-to-date

content without having to wait), the frequency with which the content is accessed (e.g., 5%-10% of content on the Internet represents 90% of all the *requested* content), and the underlying costs and bandwidth constraints associated with maintaining up-to-date, coherent content across a particular network (e.g., network 480).

Three content routing techniques which may be selected (based on one or more of the foregoing variables) to maintain coherent content across the plurality of nodes illustrated in **Figure 4** are content revalidation, content notification, and content synchronization.

Content Revalidation

When content validation is selected, the original content source will be checked only when the content is requested locally. For example, client 473 may request an installation program for a new Web browser (e.g., the latest version of Microsoft's™ Internet Explorer™). The file may then be transmitted form ICDS server 460 to client 473 and a copy of the file cached locally on ICDS node 430. Consequently, if client 472 requests the same program, for example, two weeks later, ICDS node 430 may be configured to check ICDS server 460 to ensure that it contains the most recent copy of the file before passing it on to client 472 (i.e., ICDS node 430 "revalidates" the copy it has locally).

ICDS node 430 may also be configured to revalidate a piece of content only if has been stored locally for a predetermined amount of time (e.g., 1 week). The particular length of time selected may be based on one or more of the variables discussed above. Moreover, in one embodiment, the age/revision of a particular piece of content is determined based on tags (e.g., HTML metatags) inserted in the particular content/file.

Revalidation may work more efficiently with certain types of content than with others. For example, revalidation may be an appropriate mechanism for maintaining up-to-date copies of larger files which do not change very frequently (i.e., such as the program installation files described above). However, revalidation may not work as efficiently for caching smaller and/or continually changing files (e.g., small HTML files) because the step of revalidating may be just as time consuming as making a direct request to ICDS server 460 for the file itself. If the file in question is relatively small and/or is changing on a minute-by-minute basis (e.g., an HTML file containing stock quotes) then one or more other content routing techniques may be more appropriate.

Of course, other routing variables may influence the decision on which technique to use, including the issue of how strong the data transmission connection is between ICDS node 430 and ICDS server 460 (i.e., how reliable it is, how much bandwidth is available . . . etc) and the necessity that the underlying information cached locally (at ICDS node 430) be accurate. The important thing to remember is that ICDS node 430 – because of its underlying open API architecture – may be configured based on the unique preferences of a particular client.

Content Notification

Content notification is a mechanism wherein the central repository for a particular piece of content maintains a list of nodes, or "subscribers," which cache a copy of the content locally. For example, in **Figure 4**, a plurality of agents may run on ICDS server 460 which maintain a list of content subscribers (e.g., ICDS node 430, ICDS node 420 . . . etc) for specific types of content (e.g., HTML, data streaming files, FTP files . . . etc). In one embodiment of the system, a different agent may be executed for each protocol supported by ICDS server 460 and/or ICDS nodes 410-440.

When a particular piece of content is modified on ICDS server 460, a notification of the modification may be sent to all subscriber nodes (i.e., nodes which subscribe to that particular content). Upon receiving the notification, the subscriber node – e.g., ICDS node 430 – may then invalidate the copy of the content which it is storing locally. Accordingly, the next time the content is requested by a client (e.g., client 472), ICDS node 430 will retrieve the up-to-date copy of the content from ICDS server 460. The new copy will then be maintained locally on ICDS node 430 until ICDS node 430 receives a second notification from an agent running on ICDS server 460 indicating that a new copy exists.

Alternatively, each time content is modified on ICDS server 460 the modified content may be sent to all subscriber nodes along with the notification. In this manner a local, up-to-date copy of the content is always ensured. In one embodiment of the system, notification and/or transmittal of the updated content by the various system agents is done after a predetermined period of time has elapsed (e.g., update twice a day). The time period may be selected based on the importance of having an up-to-date copy across all nodes on the network 480, 490.

As was the case with content revalidation, the different varieties of content notification may work more efficiently in some situations than in others. Accordingly, content notification may be selected as a protocol (or not selected) based on one or more of the routing variables recited at the beginning of this section (i.e., the "content routing" section). For example, content notification may be an appropriate technique for content which is frequently requested at the various nodes across networks 480 and 490 (e.g., for the 5-10% of the content which is requested 90% of the time), but may be a less practical technique for larger amount of content which is requested infrequently. As another example, large files which change frequently may not be well suited for content notification (i.e., particularly the type of content notification where the actual file is sent to all subscribers along with the notification) due to bandwidth constraints across networks 480 and/or 490 (i.e., the continuous transmission of large, frequently changing files may create too much additional network traffic).

Content Synchronization

Content synchronization is a technique for maintaining an exact copy of a particular type of content on all nodes on which it is stored. Using content synchronization, as soon as a particular piece of content is modified at, for example, ICDS node 430, it will immediately be updated at all other nodes across networks 480 and/or 490. If the same piece of data was concurrently modified at one of its other storage locations (e.g., ICDS node 410) then the changes may be backed off in order to maintain data coherency. Alternatively, an attempt may be made to reconcile the two separate modifications if it is possible to do so (using, e.g., various data coherency techniques).

Once again, as with content notification and content revalidation, content synchronization is more suitable for some situations than it is for others. For example, content synchronization is particularly useful for information which can be modified from several different network nodes (by contrast, the typical content notification paradigm assumes that the content is modified at one central node). Moreover, content synchronization may be useful for maintaining content across a network which it is particularly important to keep current. For example, if network 480 is an automatic teller machine (hereinafter "ATM") network, then when a user withdraws cash from a first node (e.g., ICDS node 440), his account will be instantly updated on all nodes (e.g., ICDS node 410, 420, 460, and 430) to reflect the withdrawal. Accordingly, the user would not be able to go to a different node in a different part of the country and withdraw more than what he actually has in his account.

As another example, a user's account status on a network (i.e., whether he is a current subscriber and/or what his network privileges are) may be maintained using content synchronization. If, for example, a user of network 480 were arrested for breaking the law over network 480 (e.g., distributing child pornography), it would be important to disable his user account on all network nodes on which this information might be cached. Accordingly, using content synchronization, once his account was disabled at one node on network 480 this change would automatically be reflected across all nodes on the network.

As previously stated, the choice of which content routing technique to use for a particular type of content may be based on any of the variables set forth above. In one embodiment, the frequency with which content is accessed across the networks 480 and/or 490 may be an important factor in deciding which protocol to use. For example, the top 1% accessed content may be selected for content synchronization, the top 2%-10% accessed content may be selected for content notification, and the remaining content across networks 480, 490 may be selected for content revaildation.

III. Content Delivery Medium Selection

In addition to the content routing flexibility provided by the content delivery system as set forth above, one embodiment of the system allows content delivery nodes such as ICDS node 430 to select from a plurality of different transmission media. For example, ICDS node 430 may receive content from ICDS server 460 via a plurality of communication media, including, but not limited to, satellite transmission, wireless RF transmission, and terrestrial transmission (e.g., fiber).

Moreover, as with the selection of a particular content routing technique, the selection of a particular transmission medium may be based on any of the variables set forth above (see, e.g., routing variables listed under counter routing heading; page 24, line 18 through page 25, line 9). Moreover, the choice of a particular transmission medium may be dynamically adjustable based on performance of that medium. For example, ICDS node 430 may be configured to receive all of its content over terrestrial network 480 as long as network 480 is transmitting content at or above a threshold bandwidth. When transmissions over network 480 dip below the threshold bandwidth, ICDS node 480 may then begin receiving certain content via satellite broadcast or wireless communication.

In addition, a transmission medium may be selected for transmitting specific content based on how frequently that content is accessed. For example, the top 10% frequently accessed content may be continually pushed out to ICDS node 430 via satellite broadcast while the remaining content may be retrieved by (i.e., "pulled" to) ICDS node 430 over network 480 upon request by clients (e.g., client 473). Accordingly, those employing ICDS nodes such as node 430 can run a cost-benefit analysis to determine the most cost effective way to implement their system by taking in to consideration, for example, the needs of their users, the importance of the content involved and the expense of maintaining multiple transmission connections into ICDS node 430 (e.g., the cost associated with maintaining an ongoing satellite connection).

In one embodiment of the system, tags (e.g., HTML metatags) may be inserted into particular types of content to identify a specific transmission path/medium for delivering that content to ICDS node 480. The tags in this embodiment may identify to various nodes (and/or routers) across networks 480 and/or 490 how the particular content should be routed across the networks (e.g., from node 410 to node 420 via terrestrial network 480; from node 420 to node 430 via wireless transmission).

One of ordinary skill in the art will readily recognize from the preceding discussion that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention. Throughout this detailed description, numerous specific details are set forth such as specific network protocols (i.e., RADIUS) and networks (i.e., the Internet) in order to provide a thorough understanding of the present invention. It will be appreciated by one having ordinary skill in the art, however, that the present invention may be practiced without such specific details. In other instances, well known software and hardware configurations/techniques have not been described in detail in order to avoid obscuring the subject matter of the present invention. The invention should, therefore, be measured in terms of the claims which follow.

CLAIMS

What is claimed is:

1. A network content delivery system configured to:

select a first content routing technique for processing a first set of network content; and

select a second content routing technique for processing a second set of network content, wherein said first and second content routing techniques are selected based on one or more content routing variables.

2. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content revalidation technique.

3. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content notification technique.

4. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content synchronization technique.

5. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the frequency with which said network content is accessed by users.

6. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the size of said network content.

7. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the frequency with which said network content is modified.

8. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the type of network content (e.g., HTML, Usenet News).

9. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is identity of the user requesting said network content.

10. The network content delivery system as claimed in Claim 2 wherein said content revalidation technique is selected based on the size of said network content.

11. The network content delivery system as claimed in Claim 2 wherein said content revalidation technique is selected based on the frequency with which said network content is accessed.

12. The network content delivery system as claimed in Claim 3 wherein said content notification technique is selected based on the size of said network content.

13. The network content delivery system as claimed in Claim 3 wherein said content notification technique is selected based on the frequency with which said network content is accessed.

14. The network content delivery system as claimed in Claim 4 wherein said content synchronization technique is selected based on the size of said network content.

15. The network content delivery system as claimed in Claim 4 wherein said content synchronization technique is selected based on the frequency with which said network content is accessed.

16. The network content delivery system as claimed in Claim 1 including the additional step of selecting a first transmission medium for a first group of network content based on one or more of said content routing variables.

17. The network content delivery system as claimed in Claim 16 including the additional step of selecting a second transmission medium for a second group of network content based on one or more of said content routing variables.

18. The network content delivery system as claimed in Claim 1 including an application programming interface for interfacing with a plurality of network protocol and service modules.

19. A content delivery system comprising:

a network node for storing network content;

a first transmission medium communicatively coupled to said network node for transmitting a first set of network content to said network node; and

a second transmission medium communicatively coupled to said network node for transmitting a second set of network content to said network node,

wherein said first and second sets of network content are selected based on one or more routing variables.

20. The content delivery system as claimed in Claim 19 wherein said first transmission medium is a satellite transmission.

21. The content delivery system as claimed in Claim 19 wherein said first transmission medium is a wireless radio frequency transmission.

22. The content delivery system as claimed in Claim 19 wherein said first transmission medium is terrestrial-based transmission.

23. The content delivery system as claimed in Claim 19 wherein said network node monitors transmission bandwidth of said first transmission medium and reallocates content

from said first set to said second set if said first transmission medium drops below a predetermined threshold value.

24. The content delivery system as claimed in Claim 23 wherein said first transmission medium is terrestrial and said second transmission medium is non-terrestrial.

25. The content delivery system as claimed in Claim 19 wherein content is included in said first set based on the frequency with which said content is accessed.

26. The network content delivery system as claimed in Claim 19 including an application programming interface for interfacing with a plurality of network protocol and service modules.

27. An article of manufacture including a sequence of instructions stored on a computer-readable media which, when executed by a network node, cause the network node to perform the acts of:

establishing a plurality of groups of network content to be cached on said network node based on one or more content routing variables;

selecting a first content routing technique for maintaining data coherency in a first group of said plurality; and

selecting a second content routing technique for maintaining data coherency in a second group of said plurality.

28. The article of manufacture as claimed in claim 27 wherein said first content routing technique is content revalidation.

29. The article of manufacture as claimed in Claim 28 wherein said second content routing technique is content notification.

30. The article of manufacture as claimed in Claim 28 wherein said second content routing technique is content synchronization.

31. The article of manufacture as claimed in Claim 28 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

32. The article of manufacture as claimed in Claim 29 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

33. The article of manufacture as claimed in Claim 30 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

34. A network node comprising:

an application programming interface ("API"), said API including a distributed relational database engine;

a plurality of protocol modules for interfacing with said API, said protocol modules configured to allow said system to communicate over a network using a plurality of network protocols;

a cache memory for caching data communicated to said cache memory using said plurality of protocol modules; and

a data services module for maintaining coherency between said data stored in said cache memory and data stored at other nodes across said network.

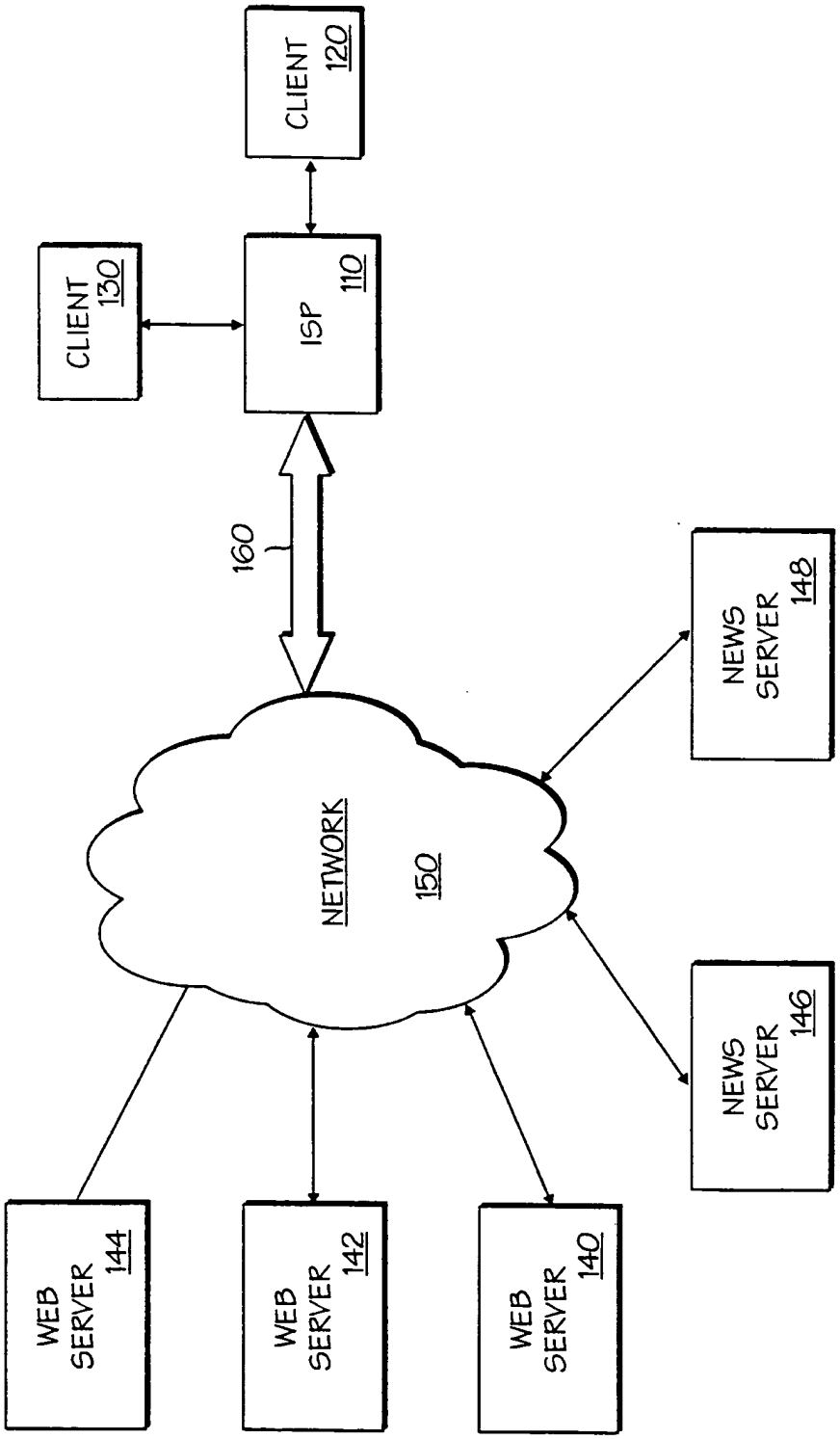

CLIENT 120
CLIENT 130
ISP 110
160
NEWS SERVER 148
NETWORK 150
NEWS SERVER 146
WEB SERVER 144
WEB SERVER 142
WEB SERVER 140

FIG. 1

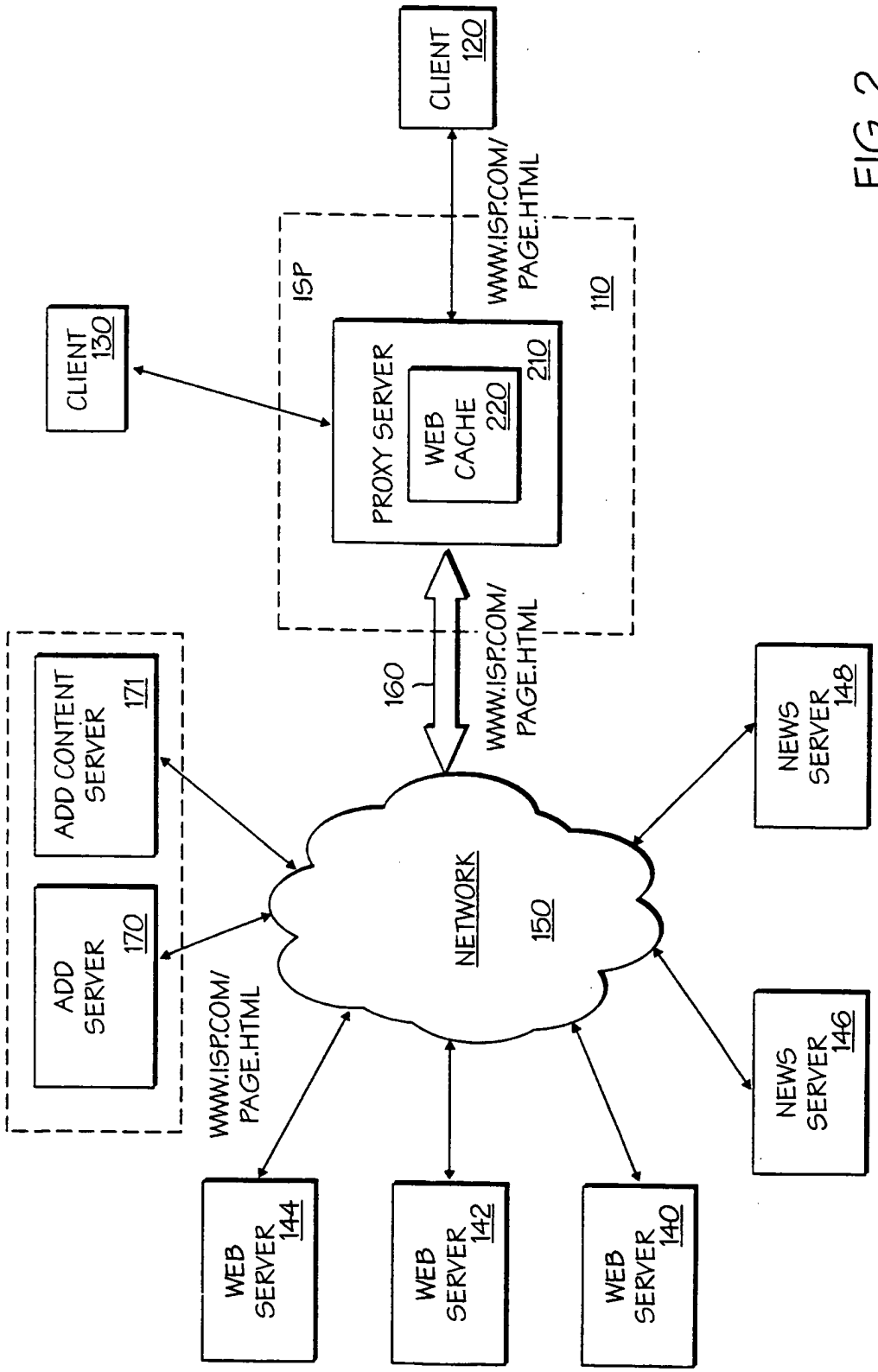

CLIENT 120
WWW.ISP.COM/ PAGE.HTML
ISP
110
CLIENT 130
PROXY SERVER 210
WEB CACHE 220
160
WWW.ISP.COM/ PAGE.HTML
ADD CONTENT SERVER 171
ADD SERVER 170
WWW.ISP.COM/ PAGE.HTML
NETWORK 150
NEWS SERVER 148
NEWS SERVER 146
WEB SERVER 144
WEB SERVER 142
WEB SERVER 140

FIG. 2

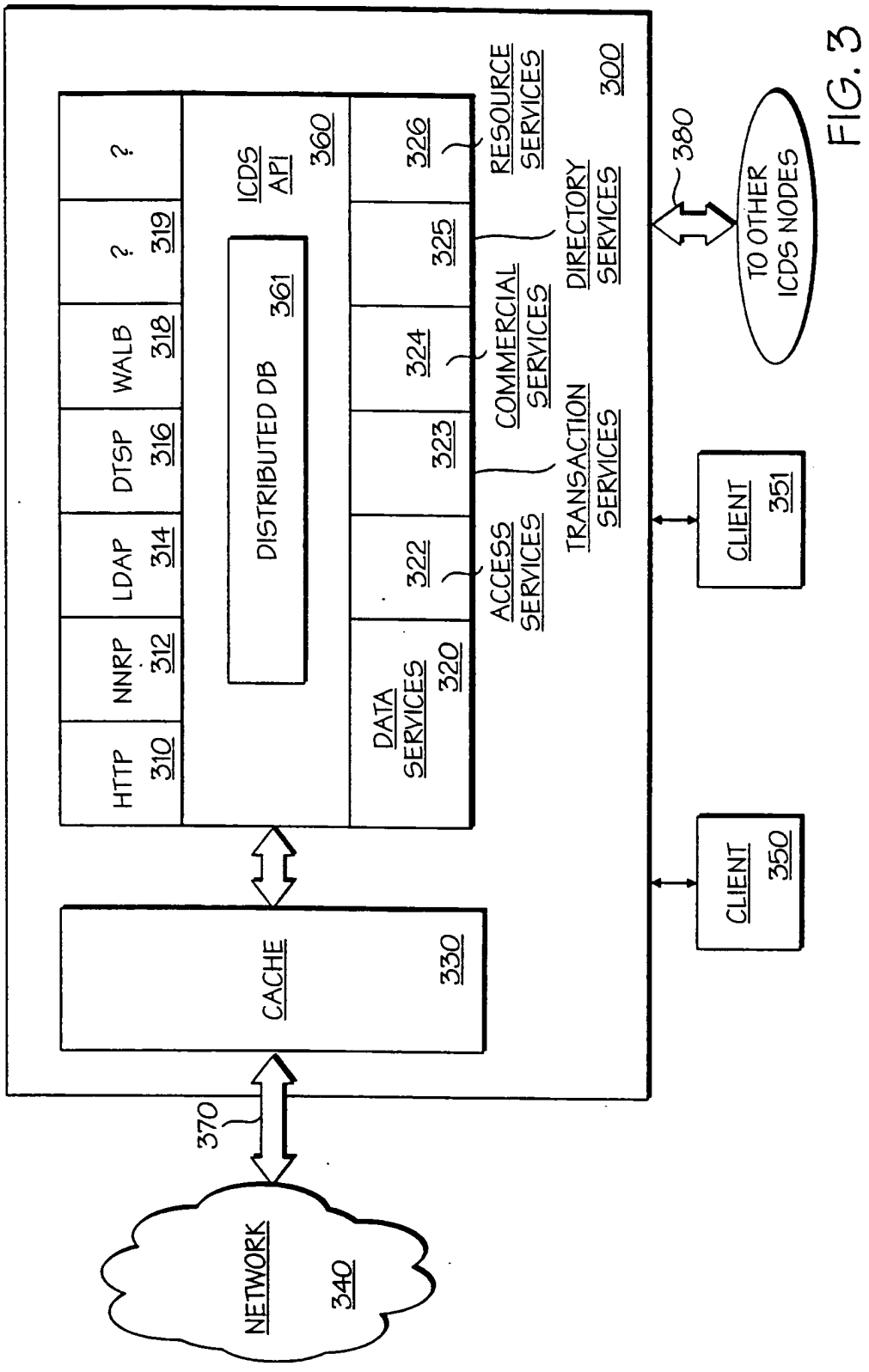

HTTP 310
NNRP 312
LDAP 314
DTSP 316
WALB 318
? 319
?
DISTRIBUTED DB 361
ICDS API 360
DATA SERVICES 320
322
323
324
325
326
ACCESS SERVICES
TRANSACTION SERVICES
COMMERCIAL SERVICES
DIRECTORY SERVICES
RESOURCE SERVICES
300
CACHE 330
370
NETWORK 340
380
TO OTHER ICDS NODES
CLIENT 351
CLIENT 350

FIG. 3

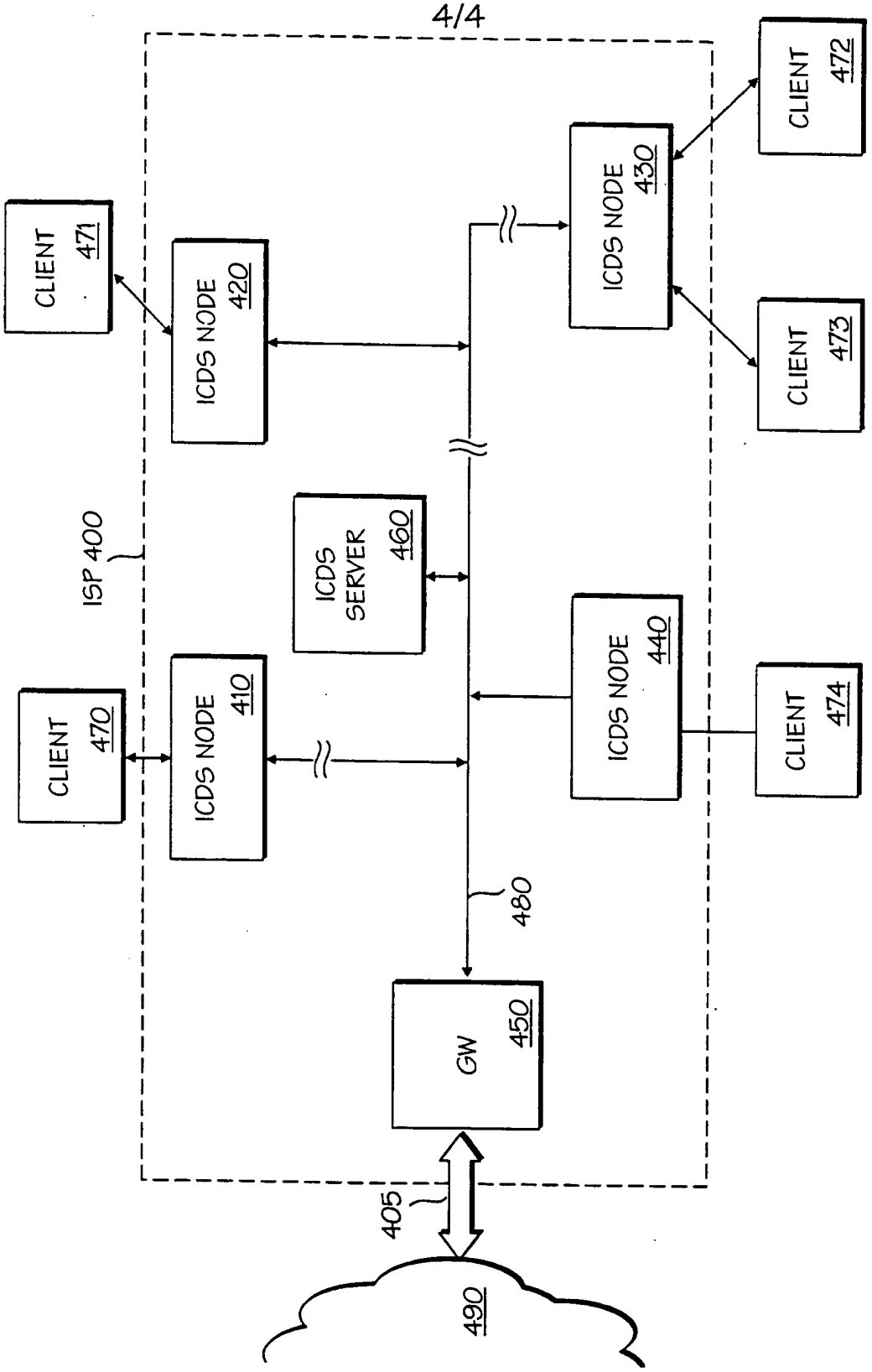

ISP 400
CLIENT 470
ICDS NODE 410
CLIENT 471
ICDS NODE 420
ICDS SERVER 460
ICDS NODE 430
CLIENT 472
CLIENT 473
ICDS NODE 440
CLIENT 474
480
GW 450
405
490

FIG. 4

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 May 2001 (25.05.2001)

PCT

(10) International Publication Number
WO 01/37209 A1

(51) International Patent Classification[7]: G06K 9/36, 9/46

(21) International Application Number: PCT/US00/30825

(22) International Filing Date:
8 November 2000 (08.11.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/438,666 12 November 1999 (12.11.1999) US

(71) Applicant: TERALOGIC, INC. [US/US]; 1240 Villa Street, Mountain View, CA 94041 (US).

(72) Inventors: CASTOR, Jon, S.; 2160 Stockbridge Avenue, Woodside, CA 94062 (US). CHUI, Charles, K.; 340 Olive Street, Menlo Park, CA 94025 (US).

(74) Agents: WILLIAMS, Gary, S. et al.; Pennie & Edmonds LLP, 1155 Avenue of the Americas, New York, NY 10036 (US).

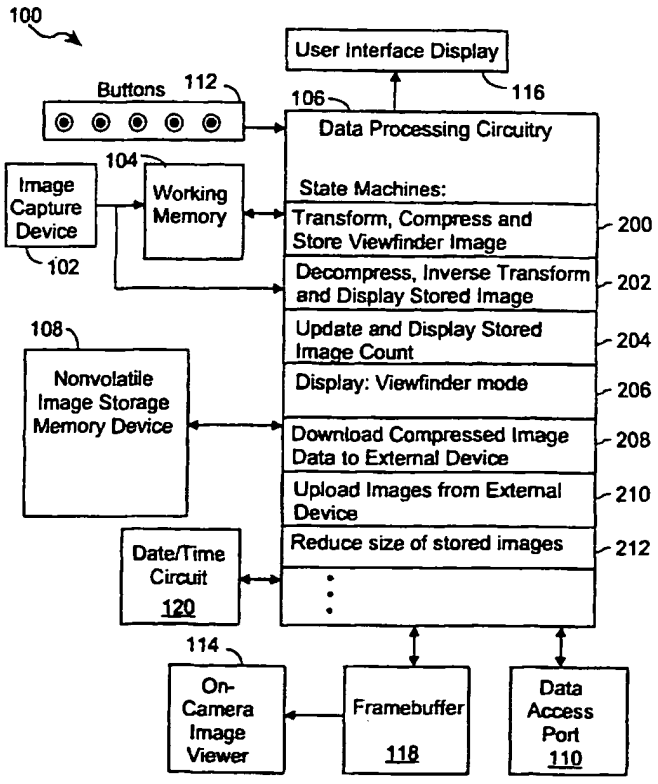
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

[Continued on next page]

(54) Title: PICTURE AND VIDEO STORAGE MANAGEMENT SYSTEM AND METHOD



(57) Abstract: An image processing system (100) stores image files in a memory device (108) at a number of incremental quality levels. Each image file has an associated image quality (that is fidelity or resolution) level corresponding to a quality level at which the corresponding image has been encoded. The images are initially encoded by applying a predefined transform, such as a DCT transform or wavelet-like transform (200), to image data received from an image capture mechanism (102) and then applying a data compression method to the transform data (200). The image is regenerated by successively applying a data decompression method and an inverse transform to an image file (202). Image file size reduction circuitry (212) and one or more state machines are used to lower the quality level of a specified one of the image files, including circuitry for extracting a subset of the data in the specified image file and forming a lower quality version of the specified image file that occupies less space in the memory device than was previously occupied by the specified image data structure. As a result, the amount of space occupied by image files in the memory device can be reduced so as to make room for the storage of additional image files or to allow more rapid transmission in a restricted bandwidth environment.

WO 01/37209 A1

— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PICTURE AND VIDEO STORAGE MANAGEMENT SYSTEM AND METHOD

The present invention relates generally to the storage of still and video images in an image processing device or system, such as a digital camera or digital video camera or a computer based image storage system, and particularly to a system and method for storing images at different image quality levels and for enabling users to dynamically compress high quality images into lower quality images in order to free up storage for the storage of additional images.

BACKGROUND OF THE INVENTION

Digital cameras typically include either permanent internal storage, and/or removable storage, for storing images. For instance, many digital cameras use removable flash memory cards for storing images. Each flash memory card has a fixed amount of memory, and when the memory card is full it can be removed from the camera and replaced by another flash memory card. In addition, digital cameras typically have a built-in viewer that enables the user to review the images stored on the flash memory card (and/or in the internal memory) and to delete user specified ones of the stored images. Deleting stored images obviously creates room for storing additional images.

When a digital camera user is "in the field" he/she generally has a limited amount of image storage on hand. If all the available image storage is full, the user has the choice of either not taking any additional pictures, or of deleting pictures from the image storage devices on hand to make room for new images. While this is actually one level better than the situation with film cameras, in which the user is simply out of luck when all the available film has been used, it is the premise of the present invention that the current image storage limitations of digital cameras are caused, in part, by failure to fully exploit the advantages of having images stored in digital format.

Similar storage vs. image quality considerations also apply to digitally encoded video frames. In particular, for any given amount of storage space, such as in a digital video camera, the goal

is to retain the best image quality for the amount of storage required for a given number of video frames. Current devices allow the user to select image quality prior to capturing a digital video image, but do not enable the user to effectively manage the storage space in the video camera with respect to video sequences already taken, other than by deletion.

It is an object of the present invention to provide a digital camera or digital video camera, or other constrained storage device, that can store images at a plurality of image quality (i.e., fidelity or resolution) levels and furthermore can reduce images initially stored at a first image quality level to a lower image quality level so as to reduce the amount of storage occupied by those images.

It is also an object of the present invention to provide space efficient and computationally efficient image and video handling mechanisms for other applications, including network connected image and video libraries, Internet web browsers executed by client computers coupled to server computers, cable television set top boxes having video storage capabilities, and so on.

SUMMARY OF THE INVENTION

In summary, the present invention is an image processing device or system, such as a digital camera or digital video camera or a computer based image storage system, that can store images at a number of different image quality levels. The image processing device includes a memory device and image management logic. The memory device stores image files that each represent a respective image, each image file having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded. The image management logic includes data processing circuitry and state machines for storing and processing image data received from an image capture mechanism. More specifically, the image management logic includes image processing circuitry and one or more state machines for applying a predefined transform, such as a wavelet-like transform, to image data received from the image capture mechanism to generate transform image data and for applying a data compression method to the transform image data so as to generate an image file having an associated image quality level.

The image management logic also includes image reconstruction circuitry and one or more state machines for successively applying a data decompression method and an inverse transform to a specified one of the image files so as to generate a reconstructed image suitable for display on an image viewer.

Further, the image management logic includes image file size reduction circuitry and one or more state machines for reducing the size of an image file while minimizing the reduction in image quality level. This circuitry extracts a subset of the data in the specified image file and forms a lower quality version of the specified image file that occupies less space in the memory device than was previously occupied by the specified image data structure. As a result, the amount of space occupied by image files in the memory device can be reduced so as to make room for the storage of additional image files or to allow more rapid transmission in a restricted bandwidth environment.

In a preferred embodiment, the image transform data in an image file is organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes. To generate a lower quality image file, the image size reduction circuitry extracts a portion of the image file that excludes the image transform data for at least one bit plane and replaces the image file with an image file containing the extracted portion. Further, the image data is also organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers. The image size reduction circuitry can also generate a lower quality image file by extracting a portion of the image file that excludes the image transform data for at least one transform layer and replaces the image file with an image file containing the extracted portion.

BRIEF DESCRIPTION OF THE DRAWINGS

Additional objects and features of the invention will be more readily apparent from the following detailed description and appended claims when taken in conjunction with the drawings, in which:

Fig. 1 is a block diagram of a digital camera in accordance with an embodiment of the present invention.

Fig. 2 depicts an image data array divided into a set of smaller analysis arrays for purposes of encoding and data compression.

Fig. 3 schematically depicts the process of transforming a raw image data array into a transform image array and compressing the transform image array into a compressed image file.

Figs. 4A and 4B depict image storage data structures.

Fig. 5 is a conceptual flow chart depicting changes in the state of a digital camera as various operations are performed.

Fig. 6 is a conceptual flow chart depicting changes in the state of a video image sequence as the video image sequence is encoded and compressed.

Figs. 7, 8 and 9 show data structures used in one particular embodiment for video image sequence encoding and compression.

Fig. 10 is a conceptual diagram of an Internet server and client devices that utilize the image or and video image compressing and management features of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

In some image processing systems, an image can be stored at a number of discrete resolution levels, typically with each resolution level differing from its "neighbors" by a resolution factor of four. In other words, if the highest resolution representation (at resolution level 1) of the image contains X amount of information, the second resolution level representation contains (for example) X/4 amount of information, the third resolution level representation contains X/16 amount of information, and so on. Thus, an image's "resolution" typically means the

amount of image information, and in the context of digital image processing systems is often expressed in terms of the number of distinct pixel elements. The number of resolution levels and the particular amount of information reduction from one level to the next may vary considerably from one system to another. Further, the present invention would be equally applicable to systems having a continuous range of resolution levels.

Another concept concerning image quality is "fidelity." The fidelity of an image can be compromised even if its resolution, in terms of the number of pixels in the image, is unchanged. An image's fidelity can be reduced by reducing the number of bits used to represent the image data. For instance, if the transform coefficients used to represent an image are represented at full fidelity using 12 bits, and then the number of bits used to represent transform coefficients is reduced to 11, the fidelity of the resulting image will be reduced. In other words, the quality of the image reconstructed from the lower fidelity data will be a little less sharp than an image reconstructed from higher fidelity data.

In this document, the terms "image quality" and "quality level" will be used in the general sense of image quality, encompassing both image "resolution" and image "fidelity." Thus, a reduction in an image's "image quality" from a top image quality level to a next highest image quality level might be accomplished either by reducing the image's resolution or by reducing its fidelity, or both. In other embodiments the terms "image quality" and "quality level" may be used to refer to other aspects of image quality as well.

Digital Camera Architecture

Referring to Fig. 1, there is shown an embodiment of a digital camera system 100 in accordance with the present invention. The digital camera system 100 includes an image capture device 102, such as a CCD or CMOS sensor array or any other mechanism suitable for capturing an image as an array of digitally encoded information. Thus the image capture device is assumed to include analog to digital conversion (ADC) circuitry for converting analog image information into digital values.

A working memory 104, typically random access memory, receives digitally encoded image information from the image capture device 102. More generally, it is used to store a digitally

encoded image while the image is being transformed and compressed and otherwise processed by the camera's data processing circuitry 106. Memory 104 may be integrated on the same integrated circuit as other devices, or may be implemented using separate circuit(s).

The data processing circuitry 106 in one embodiment consists of hardwired logic and a set of state machines for performing a set of predefined image processing operations. In alternate embodiments the data processing circuitry 106 could be implemented in part or entirely using a fast general purpose microprocessor and a set of software procedures. However, at least using the technology available in 1999, it would be difficult to process and store full resolution images (e.g., full color images having 1280 x 840 pixels) fast enough to enable the camera to be able to take, say, twenty pictures per second, which is a requirement for some commercial cameras, as well as digital video cameras. In the future, general purpose microprocessors or general purpose image data microprocessors (e.g., single instruction multiple data (SIMD) processors) may be able to provide the fast image processing needed by digital cameras, in which case the data processing circuit 106 could be implemented using such a general purpose microprocessor or perhaps a hybrid processor system.

Each image, after it has been processed by the data processing circuitry 106, is typically stored as an "image file" in a nonvolatile memory storage device 108, typically implemented using "flash" (i.e., EEPROM) memory technology. The nonvolatile memory storage device 108 is preferably implemented as a removable memory card. This allows the camera's user to remove one memory card, plug in another, and then take additional pictures. However, in some implementations, the nonvolatile memory storage device 108 may not be removable, in which case the camera will typically have a data access port 110 to enable the camera to transfer image files to and from other devices, such as general purpose, desktop computers, computer systems and devices used to warehouse libraries of images, computer systems and devices used to store and distribute image files, and so on. Digital cameras with removable nonvolatile memory 108 may also include a data access port 110.

While the amount of storage in the nonvolatile image memory 108 will vary from one implementation to the next, such devices will typically have sufficient capacity to store 10 to 50 high quality images. Once the nonvolatile image memory 108 is full, if the file size reduction methodology of the present invention is not used, the only way the camera can be

used to take additional pictures is either by deleting images from the nonvolatile image memory 108 or by replacing the nonvolatile image memory 108 with another one. If neither of these options are feasible (e.g., because the user has filled all the memory cards he/she has on hand with images that he/she does not wish to delete), then no further pictures can be taken until a new memory device 108 is inserted or the stored images are transferred to an external device (if available).

The digital camera 100 includes a set of buttons 112 for giving commands to the camera. In addition to the image capture button, there will typically be several other buttons to enable the use to select the quality level of the next picture to be taken, to scroll through the images in memory for viewing on the camera's image viewer 114, to delete images from the nonvolatile image memory 108, and to invoke all the camera's other functions. Such other functions might include enabling the use of a flash light source, and transferring image files to and from a computer. In accordance with the present invention, the user selectable functions, selected by using the buttons 112, further include reducing the size of one or more of the image files stored in the nonvolatile image memory 108 so as to make room for the storage of additional images. The buttons in one embodiment are electromechanical contact switches, but in other embodiments at least some of the buttons may be implemented as touch screen buttons on a user interface display 116, or on the image viewer 114.

The user interface display 116 is typically implemented either (A) as an LCD display device separate from the image viewer 114, or (B) as images displayed on the image viewer 114. Menus, user prompts, and information about the images stored in the nonvolatile image memory 108 may be displayed on the user interface display 116, regardless of how that display is implemented.

After an image has been captured, processed and stored in nonvolatile image memory 108, the associated image file may be retrieved from the memory 108 for viewing on the image viewer. More specifically, the image file is converted from its transformed, compressed form back into a data array suitable for storage in a framebuffer 118. The image data in the framebuffer is displayed on the image viewer 114. A date/time circuit 120 is used to keep track of the current date and time, and each stored image is typically date stamped with the date and time that the image was taken.

Image Data Structures

Referring to Fig. 2, in one embodiment the nonvolatile image memory 108 stores a directory 130 that lists all the image files 132 stored in the memory 108. Preferably, the directory 130 contains information for each stored image file 132, such as the date and time the image was taken, the quality level of the image and the file's location in the memory 108.

To understand the image data structure stored in each image file, it is helpful to first understand how an image file is encoded. Referring to Fig. 3, a raw image data array 140, obtained from the digital camera's image capture mechanism 102 (Fig. 1), is treated as a set of non-overlapping "analysis arrays" 142 of a fixed size, such as 32 x 32, or 64 x 64 (or more generally $2^n \times 2^n$, for some integer value of n). A sufficient number of subarrays are used to cover the entire data array that is to be encoded, even if some of the subarrays overhang the edges of the data array. The overhanging portions of the subarrays are filled with zero data values during the data encoding process. In a preferred embodiment, the origin of the data array is the top left corner, the first coordinate used to identify data array positions is the "Y" axis or vertical coordinate, and the second coordinate used is the "X" axis or horizontal coordinate. Thus, a position of 0,64 indicates a pixel at the top vertical position of the array, 64 pixel positions over to the right from the array origin, while a position of 32,0 indicates a pixel on the left edge of the array, 32 pixel positions vertically down from the array origin.

An appropriate transform is applied to each of the analysis arrays 142, and then the resulting transform coefficients for each analysis array are quantized (e.g., divided by an appropriate value to generate integer valued, quantized transform coefficients) so as to generate a transformed image array 144. In one embodiment the transform applied to the raw image data is a wavelet-like transform. In other embodiments a DCT transform could be used (which is the type of transform used in current JPEG image encoding systems), or other types of wavelet or wavelet-like transforms could be used.

In this document, the terms "wavelet" and "wavelet-like" are used interchangeably. Wavelet-like transforms generally have spatial frequence characteristics similar to those of conventional wavelet transforms, and are losslessly reversible, but have shorter filters that are more computationally efficient.

In one embodiment the transformed image array 144 is generated by successive applications of a wavelet-like decomposition transform. A first application of the wavelet-like decomposition transform to an initial two dimensional array of "raw" image data generates four sets of coefficients, labeled LL, HL1, LH1 and HH1. Each succeeding application of the wavelet-like decomposition transform is applied only to the LL set of coefficients generated by the previous wavelet transformation step and generates four new sets of coefficients, labeled LL, HLx, LHx and HHx, where x represents the wavelet transform "layer" or iteration. After the last wavelet-like decomposition transform iteration only one LL set remains. The total number of coefficients generated is equal to the number of data samples in the original data array. The different sets of coefficients generated by each transform iteration are sometimes called layers. The number of wavelet transform layers generated for an image is typically a function of the resolution of the initial image. Performing five to seven wavelet transformation layers is fairly typical, but more or less may be used depending on such considerations as the size of the analysis arrays, the subject matter of the image, the data processing resources available for image compression, and the like.

For the purposes of explaining the operation of the image encoding and decoding operations of the present invention, the specific type of image transform used and the specific type of data quantization used to transform a raw image file 140 into a transformed image array 142 are not relevant and therefore are not further described herein. However, a preferred embodiment of the wavelet transform and data quantization methods are described in U.S. Patent No. 5,909518, "System and Method for Performing Wavelet and Inverse Wavelet Like Transformations of Digital Data," which is hereby incorporated by reference as background information.

Each transformed image array 144 is compressed and encoded using a sparse data encoding technique. In one embodiment, the method of compressing and encoding the analysis arrays is the method described in detail in U.S. patent application 08/858,035, filed May 16, 1997, entitled "System and Method for Scalable Coding of Sparse Data Sets," now U.S. Patent No. 5,949,911, which is hereby incorporated by reference as background information. The encoded image data for all the analysis arrays of the image are combined and stored as an image file 132.

Referring to Fig. 4A, the image file 132 includes header data 160 and a sequence of data structures 162, each representing one analysis array. The header data 160 indicates the size of the image file and the image file's quality level. The header data also includes a list of analysis array size values indicating the length of each of the analysis array data structures 162, thereby enabling fast indexing into the image data. Storing size values for the analysis arrays enables the camera's data processing circuitry 106 (Fig. 1) to locate the beginning of any analysis array data structure 162 without having to decode the contents of the earlier analysis arrays in the image file 132.

As shown in Fig. 4B, the encoded data 162 representing any one analysis array is stored in "bit layer order". For each analysis array, the encoding procedure determines the most significant non-zero bit in the data to be encoded, which is herein called the y^{th} bit. The value of y is determined by computing the maximum number of bits required to encode the absolute value of any data value in the analysis array. In particular, y is equal to $int(\log_2 V) + 1$, where V is the largest absolute value of any element in the analysis array, and "int()" represents the integer portion of a specified value.

The encoded data 162 representing one analysis array includes (A) header data 170 indicating the maximum number of bits required to encode the absolute value of any data value in the analysis array, and (B) a sequence of data structures 172, each representing one bit plane of the elements in the analysis array. The x^{th} bit plane of the analysis array is the x^{th} bit of the absolute value of each of the elements in the analysis array. A sparse data encoding technique is used so that it takes very little data to represent a bit plane that contains mostly zero values. Typically, higher frequency portions of the transformed, quantized image data will contain more zero values than non-zero values, and further most of the non-zero values will have relatively small absolute value. Therefore, the higher level bit planes of many analysis arrays will be populated with very few non-zero bit values.

In an alternate embodiment, the data structure shown in Fig. 4A is modified slightly. In particular, to facilitate fast extraction of lower-resolution image data from an image file, the boundaries of the analysis arrays are adjusted, if necessary, so as to coincide precisely with the boundaries between the wavelet transform regions shown in Fig. 3 (e.g., the boundary between HL2 and HL1). If the size of the initial image array is not equal to an integer number of

analysis arrays (i.e., if either the height or width of the image array is not an integer multiple of 2^n, where the size of each analysis array is 2^n x 2^n for an integer value of n), at least some of the boundaries between wavelet transform regions will fall in the middle of the analysis regions. For example, for a 800 x 600 pixel image, the LL region might have a size of 50 x 38. If the wavelet transform coefficients are encoded in units of analysis regions of size 32 x 32, the LL region will be encoded in four analysis regions, three of which would normally contain data for neighboring wavelet transform regions. In this alternate embodiment, each analysis array that overlaps a border between wavelet transform regions is replaced by two or four analysis regions (depending on whether the analysis array overlaps one or two region boundaries), with zero values being stored in the appropriate locations so that each analysis array contains data from only one wavelet transform region. The analysis arrays are still stored in "origin sorted order" in the image file 132, with the "origin" now being defined as the coordinate of the coefficient closest to the upper left corner of the analysis array that has not been overwritten with zero values.

In another alternate embodiment, a different transform than the wavelet-like transform could be used, but the resulting image data would still be stored in bit plane order. For instance, a DCT transform could be used.

In some embodiments of the present invention, the raw image array received from the digital camera's image capture mechanism may first be divided into "analysis arrays" and then transformed and quantized. Further, the analysis arrays may each be a thin horizontal strip of the image array. That is, each analysis array may extend the full width of the image array, but have a height of only a few (e.g., 4 to 16) image elements. In yet another embodiment, the image array might not be divided into analysis arrays at all.

Generally, in all embodiments described above, the compressed encoded image data is stored in bit plane order. The reason that bit plane ordered storage is favored is that it makes gradual fidelity reduction very easy: to reduce the fidelity of an image file by a minimum amount, the data for the lowest level bit plane in the file is discarded and the remaining image data is retained, resulting in a smaller file with one bit plane less fidelity.

However, in yet other alternate embodiments, each image file could be organized in "quality level support order" with the data for each analysis array being arranged so that each successive data structure 172 stores the image information needed to increase image quality by one predefined image quality level. Thus, the information in some data structures 172 might represent two, three or more bit planes of information. In this embodiment, an image can be reduced by one quality level by deleting the last data structure 172 from every analysis array data structure 172 in the image file.

Digital Camera State Machines

For the purposes of this explanation, it will be assumed that the digital camera 100 has four predefined image quality levels: High, Very Good +, Very Good -, and Good. It will be further assumed that image files stored at High quality typically occupy about twice as much space as image files stored at Good quality. In other embodiments, more or fewer image quality levels could be used, and the ratio of image file sizes from highest to lowest quality could be larger or smaller than 2:1. For instance, if the camera is capable of taking very high resolution images, such as 2000 x 2000 pixels or even 4000 x 4000 pixels, and at very high fidelity, then it would make sense to provide a large number of quality levels with a ratio of image file sizes from highest to lowest quality of perhaps as high as 64:1.

It is noted that an image file's quality cannot be increased once it has been lowered, unless the original image file or an alternate source thereof remains available, because the information needed to restore the image's quality has been lost.

Referring back to Fig. 1, the digital camera 100 preferably includes data processing circuitry 106 for performing a predefined set of primitive operations, such as performing the multiply and addition operations required to apply a transform to a certain amount of image data, as well as a set of state machines 200-212 for controlling the data processing circuitry so as to perform a set of predefined image handling operations. In one embodiment, the state machines in the digital camera are as follows.

- One or more state machines 200 for transforming, compressing and storing an image received from the camera's image capture mechanism. This image is sometimes called the "viewfinder" image, since the image being processed is generally the one seen on the camera's

image viewer 114. This set of state machines 200 are the ones that initially generate each image file stored in the nonvolatile image memory 108. Prior to taking the picture, the user specifies the quality level of the image to be stored, using the camera's buttons 112. It should noted that in most digital cameras the viewfinder is capable of displaying only a very small and low fidelity version of the captured image, and thus the image displayed in the camera's viewfinder is typically a much lower quality rendition of the captured image than the quality of the "viewfinder" image stored in the image file.

- One or more state machines 202 for decompressing, inverse transforming and displaying a stored image file on the camera's image viewer. The reconstructed image generated by decompressing, inverse transforming and dequantizing the image data is stored in camera's framebuffer 118 so that it can be viewed on the image viewer 114.
- One or more state machines 204 for updating and displaying a count of the number of images stored in the nonvolatile image memory 108. The image count is preferably displayed on the user interface display 116. This set of state machines 204 will also typically indicate what percentage of the nonvolatile image memory 108 remains unoccupied by image files, or some other indication of the camera's ability to store additional images. If the camera does not have a separate interface display 116, this memory status information may be shown on the image viewer 114, for instance superimposed on the image shown in the image viewer 114 or shown in a region of the viewer 114 separate from the main viewer image.
- One or more state machines 206 for implementing a "viewfinder" mode for the camera in which the image currently "seen" by the image capture mechanism 102 is displayed on the image viewer 114 to that the user can see the image that would be stored if the image capture button is pressed. These state machines transfer the image received from the image capture device 102, possibly after appropriate remedial processing steps are performed to improve the raw image data, to the camera's framebuffer 118.
- One or more state machines 208 for downloading images from the nonvolatile image memory 108 to an external device, such as a general purpose computer.
- One or more state machines 210 for uploading images from an external device, such as a general purpose computer, into the nonvolatile image memory 108. This enables the camera to be used as an image viewing device, and also as a mechanism for transferring image files on memory cards.
- One or more state machines 212 for reducing the size of image files in the nonvolatile image memory 108. This will be described in more detail next.

In the context of the present invention, an image file's quality level can be reduced in one of two ways: 1) by deleting from the image file all the analysis arrays associated with one or more transform layers, or 2) by deleting from the image file one or more bit planes of data. In either method, the state machines 212 extract the data structures of the image file that correspond to the new, lower image quality level selected by the user, and then replaces the original image file with one that stores the extracted data structures. Alternately, the original image file is updated by deleting a portion of its contents, thereby freeing some of the memory previously occupied by the file. A feature of the present invention is that the image quality level of an image file can be lowered without having to reconstruct the image and then re-encode it, which would be costly in terms of the computational resources used. Rather, the data structures within the image file are pre-arranged so that the image data in the file does not need to be read, analyzed or reconstructed. The image quality level of an image file is lowered simply by keeping an easily determined subset of the data in the image file and deleting the remainder of the data in the image file, or equivalently by extracting and storing in a new image file a determined subset of the data in the image file and deleting the original image file.

For the purposes of this document, it should be noted that the term "deleting" when applied to a data structure in an image file does not necessarily mean that the information in the data structure is replaced with null values. Rather, what this means is that the image file is replaced with another image file that does not contain the "deleted" data structure. Thus, various data structures in an image file may be deleted simply by copying all the other data structures in the image file into a new image file, and updating all required bookkeeping information in the image file and the image directory for the modified file. The "deleted" data structures may actually remain in memory unchanged until they are overwritten with new information. Alternately, data in an image file may in some implementations be deleted solely by updating the bookkeeping information for the file, without moving any of the image data.

In one embodiment of the present invention, the digital camera lowers the image quality of an image from High quality to "Very Good +" by deleting the two lowest bit planes of the image. Similarly, lowering the image's quality to "Very Good -" is accomplished by deleting two more bit planes of the image, and then lowering the image's quality to Good is accomplished by deleting yet another two bit planes of the image. More generally, each quality level

transition is represented by deleting a certain percentage of the bit planes of the highest quality level representation of the image.

In an alternate embodiment, the transition from High quality to the next highest quality level is accomplished by deleting the analysis arrays for a first transform layer (e.g., the analysis arrays for the HL1, HH1 and LH1 regions of the transformed image in Fig. 3). Subsequent quality level transitions to lower quality levels are accomplished by deleting appropriate numbers of bit planes.

In one embodiment of the present invention the digital camera provides two image file size reduction modes. In a first size reduction mode, the user selects one image (or a specified group of images), uses the camera's buttons to indicate what lower quality level the image is to be stored at, and then the state machine 212 generates a smaller image file and stores the resulting image file in the camera's nonvolatile image memory 108. In the second size reduction mode, the user commands the camera to reduce the size of all image files that are currently stored at quality level A to quality level B. For instance, in this second size reduction mode the user might command the camera to convert all "High" quality image files to "Very Good +" image quality files. This latter size reduction mode is particularly useful for "clearing space" in memory 108 to enable additional pictures to be stored in the memory 108.

In another embodiment, the camera or other device may include one or more automatic image file size reduction modes. For instance, in one such mode the camera could be set to record all pictures at a particular quality level. When the camera's memory is sufficiently filed with images files so that there is insufficient room to store one more image at the current quality level setting, the camera automatically reduces the size of enough of the stored image files so as to create room for one more image at the current quality level. In some embodiments, the quality level setting of the device for future images might be automatically reduced to match the quality level of the highest quality image stored in the camera's memory. In this way, the camera takes and stores the maximum quality images for the space available, and this maximization will occur flexibly and "on-the-fly."

Camera Operation and
Image File Size Reduction

Referring to Fig. 5, the status of a digital camera is represented by status information displayed on the camera's user interface display 116. For example, before the camera's image memory 108 is filled, the camera might indicate to the user that it is currently storing twenty-one pictures at High quality and has enough memory to store three more pictures. The indication of how many more pictures can be stored in the camera's image memory 108 (Fig. 1) depends on the camera's current picture quality setting, which determines the quality of the next picture to be taken.

After the camera has stored three more pictures, the camera's image memory 108 is full (i.e., it has insufficient room to store another picture at the camera's current picture quality setting), and the camera indicates to the user that it is currently storing twenty-four pictures at High quality and has enough memory to store zero more pictures. For the purposes of this example, we will assume that the user wants to take at least ten more pictures, despite the fact that he/she has no more memory cards. To make this possible, the user utilizes the image size reduction feature of the camera.

In this example, the user commands the camera to reduce all "High" quality image files down one quality level to the "Very Good +" quality level. The camera accomplishes this by running the size reduction state machine 212 and then updating the status information displayed on the camera's user interface display 116. In this example, the twenty-four images are now shown to be stored in image files having the "Very Good +" quality level, and the camera has room for seven new images at the High quality image level.

In this example, the user next commands the camera to perform a second size reduction so as to compress all "Very Good +" quality image files down one quality level to the "Very Good -" quality level. The camera accomplishes this by running the size reduction state machine 212 and then updating the status information displayed on the camera's user interface display 116. In this example, the twenty-four images are now shown to be stored in image files having the "Very Good -" quality level, and the camera has room for twelve new images at the High quality image level.

Alternately, if the user had, before capturing the images, switched the quality level for new images to "Very Good +" quality, a single image size reduction step might have been sufficient to create room for at least ten additional pictures.

In another example, the digital camera may be configured to have an automatic image file size reduction mode that is activated only when the camera's memory is full and the user nevertheless presses the image capture button on the camera. In this mode of operation, the camera's image processing circuitry reduces the size of previously stored image files as little as possible so as to make room for an additional image file. If the user continues to take more pictures in this mode, the quality of the stored images will eventually degrade to some user defined or predefined setting for the lowest allowed quality level, at which point the camera will not store any additional image files until the permitted quality level is lowered further or at least some of the previously stored image files are transferred to another device or otherwise deleted.

The image management system and method of the present invention can also be implemented in computer systems and computer controlled systems, using a data processor that executes procedures for carrying of the image processing steps discussed above. The present invention can also be implemented as a computer program product (e.g., a CD-ROM or data signal conveyed on a carrier signal) containing image processing procedures suitable for use by a computer system.

Video Image Management System

Referring to Fig. 6, there is shown a conceptual data flow diagram for a video image management system for storing video images at a plurality of image quality levels. The basic structure of the video image management system is the same as shown in Fig. 1. However, when the camera is a digital video camera, successive images F_i are automatically generated at a predefined rate, such as eight, sixteen, twenty-four or thirty frames per second. In a preferred embodiment, the sequence of video images is processed N frames at a time, where N is an integer greater than three, and is preferably equal to four, eight or sixteen; generally N will be determined by the availability of memory and computational resources in the particular system in which the invention is being implemented. That is, each set of N (e.g., sixteen) successive

images (i.e., frames) are processed as a set, as follows. For each set of sixteen frames F_{16x} to F_{16x+15}, all the frames except the first one are replaced with differential frames. Thus, when N=16, fifteen differential frames F_{i+1}-F_i are generated. Then, following the data processing method shown in Fig. 3 and discussed above, the first frame and the fifteen differential frames are each divided into analysis arrays, a wavelet-like or other transform is applied to the analysis arrays, and then the resulting transform coefficients are encoded.

In alternate embodiments, other methodologies could be used for initially transforming and encoding each set of success frames. For instance, a frame by frame decision might be made, based on a measurement of frame similarity or dissimilarity, as to whether or not to replace the frame with a differential frame before applying the transform and encoding steps.

In all embodiments, the image file (or files) representing the set of frames is stored so as to facilitate the generation of smaller image files with minimal computational resources. In particular, the data in the image file(s) is preferably stored using distinct data structures for each bit plane (see Fig. 4B). Furthermore, as explained above with reference to Fig. 4A, the analysis arrays may be adjusted prior to the transform step so that the boundaries between analysis arrays correspond to the boundaries between transform layer coefficients. By so arranging the data stored in the video image files, the generation of smaller, lower quality level video image files is made much easier.

Continuing to refer to Fig. 6, after the video image files for a video frame sequence have been generated at a particular initial quality level, the user of the device (or the device operating in a particular automatic mode) may decide to reduce the size of the video image files while retaining as much image quality as possible. By way of example, in a first video image file size reduction step, the HH1 transform coefficients for the last eight frames of each sixteen frame sequence are deleted. In a second reduction step, the HH1 transform coefficients are deleted for all frames other than the first frame of each sixteen frame sequence. In a third reduction step, the Z (e.g., four) least significant bit planes of the video image files are deleted. In a fourth reduction step, the HL1 and LH1 coefficients are deleted for the last eight frames of each sixteen frame sequence. In a fifth reduction step, the HL1 and LH1 coefficients are deleted for all frames other than the first frame of each sixteen frame sequence. These reduction steps are only examples of the type of file size reduction steps that could be

performed. For instance, in other embodiments, bit planes might be deleted in earlier reduction steps and transform layers (or portions of transform layers) deleted only in later reduction steps. In general, each video image size reduction causes a corresponding decrease in image quality.

Referring to Figs. 7, 8 and 9, in another embodiment, video image sequences are compressed and encoded by performing a time domain, one dimensional wavelet transformation on a set of video frames. In particular, the video frames are divided into groups of N frames, where N is an integer greater than 3, and for every x,y pixel position in the video image, a one dimensional K level wavelet transform is performed on the pixels for a sequence of N+1 frames. For instance, the K level wavelet transform is performed on the 1,1 pixels for the last frame of the previous group and the current group of N frames, as well as the 1, 2 pixels, the 1,3 pixels and so on.

In order to avoid artifacts from the separate encoding of each group of N frames, the frame immediately preceding the current group is included in K level wavelet transform. The wavelet transform uses a short transform filter that extends only one position to the left (backwards in time) of the position for which a coefficient is being generated and extends in the right hand (forward in time) direction only to the right hand edge of the set of N frames.

Furthermore, as shown in Fig. 8, the "right edge" coefficients are saved for each of the first through K-1 level wavelet transforms for use when processing the next group of N frames. In a preferred embodiment, only the rightmost edge coefficient is saved for each of the first through K-1 level transforms; in other embodiments two or more right edge coefficients may be saved and used when processing the next block of N frames. When the second level transform is performed on a block of N frames, the saved layer 1 right edge coefficients for the previous set of N frames are used (i.e., included in the computation of the leftmost computed coefficient(s) for layer 2). By saving the rightmost edge coefficients for each of the 1 through K-1 layers, artifacts that would caused (during regeneration of the video image sequence) by the discontinuities between the last frame of one block and the first frame of the next block are avoided, resulting in a smoother and more visually pleasing reconstructed video image sequence. The wavelet-like transformation and data compression of a video sequence is shown in pseudocode form in Table 1.

Table 1
Pseudocode for Wavelet-Like Transform and
Compression of One Block of Video Frames

```
Repeat for each block of video frames:
{
For each row y (of the images)
        {
        For each column x (of the images)
                {
                Save rightmost edge value for use when processing next block of video frames;

                Apply level 1 wavelet-like transform to time-ordered sequence of pixel values
                at position x,y, including saved edge value from prior block to generate level 1
                L and H coefficients;

                Save rightmost edge L coefficient for use when processing next block of video
                frames;

                Apply level 2 wavelet-like transform to level 1 L coefficients for position x,y,
                including saved level 1 edge value from prior block to generate level 2 L and H
                coefficients;

                Save rightmost edge level 2 L coefficient for use when processing next block of
                video frames;

                ...

                Apply level k-1 wavelet-like transform to level k-2 L coefficients for position
                x,y, including saved level k-2 edge value from prior block to generate level k-1
                L and H coefficients;

                Save rightmost edge level k-1 L coefficient for use when processing next block
                of video frames;

                Apply level k wavelet-like transform to level k-1 L coefficients for position x,y,
                including saved level k-1 edge value from prior block to generate level k L and
                H coefficients;
                }
        }
Quantize coefficients
Encode coefficients
Store coefficients in image data structure(s), creating image file for current block of video
frames
}
```

A more detailed explanation of saving edge coefficient from one block of image data for use while performing a wavelet or wavelet like transforms on a neighboring block of image data is provided in U.S. patent application serial no. 09/358,876, filed 07-22-99, "Memory Saving Wavelet-Like Image Transform System and Method for Digital Camera And Other Memory Conservative Applications," which is hereby incorporated by reference as background information.

Once the wavelet-like transform of each block of video data has been completed, all other aspects of processing the transformed video data are as described above. That is, the transformed data is quantized, stored in image data structures and subject to reductions in image quality, using the same techniques as those applied to still images and video image sequences as described above.

The video image management system and method of the present invention can also be implemented in computer systems and computer controlled systems, using a data processor that executes procedures for carrying of the video frame processing steps discussed above. The present invention can also be implemented as a computer program product (e.g., a CD-ROM or data signal conveyed on a carrier signal) containing image and/or video frame processing procedures suitable for use by a computer system.

Alternate Embodiments

The state machines of the embodiments described above can be replaced by software procedures that are executed by a general purpose (programmable) data processor or a programmable image data processor, especially if speed of operation is not a concern.

Numerous other aspects of the described embodiments may change over time as technology improvements are used to upgrade various parts of the digital camera. For instance, the memory technology used to store image files might change from flash memory to another type of memory, or a camera might respond to voice commands, enabling the use of fewer buttons.

Referring to Fig. 10, the present invention can also be used in a variety of image processing systems other than digital cameras and digital video cameras, including cable television set top

boxes, computer systems and devices used to warehouse libraries of images, computer systems and devices used to store and distribute image files, and so on. For example, an Internet server 300 can store images and/or video sequences in the wavelet transform compressed data structures of the present invention. Copies of those compressed data structures are transferred to the memory 306 of client computers or other client devices 302, using HTTP or any other suitable protocol via the Internet 304 or other communications network. When appropriate, an image or video sequence is reduced in size so as to fit in the memory available in the client computer or other device (client device). Furthermore, once an image or video sequence has been stored in the memory 306 of a client device, the techniques of the present invention can be used to manage the storage of the image, for instance through gradual reduction of image quality so as to make room for the storage of additional images or video sequences. In the embodiment shown in Fig. 10, the memory 306 of the client computer will have stored therein:

- an operating system 310;
- a browser or other image viewer application 312 for viewing documents and images;
- image files 314;
- image transform procedures 316, such as wavelet or wavelet-like transform procedures for converting a raw image array into wavelet transform coefficients, procedures for compressing and encoding the wavelet transform coefficients, as well as other transform procedures for handling images received in other image formats, such JPEG transform procedures for converting JPEG files into reconstructed image data that is then used as the raw image data by a wavelet or wavelet-like transform procedure;
- an image compression, quality reduction procedure 318 for implementing the image data structure size and quality reduction features of the present invention; and
- image reconstruction procedures 320 for decompressing and reverse transforming image files so as to generate reconstructed image data arrays that are suitable for viewing on the monitor of the client workstation, or for printing or other use.

The client workstation memory 306 will typically include both high speed random access memory and slower non-volatile memory such as a hard disk and/or read-only memory. The client workstation's central processing unit(s) 308 execute operating system procedures and image handling procedures, as well as other applications, thereby performing image processing functions similar to those performed by dedicated circuitry in other embodiments of the present invention.

As indicated above, when the present invention is used in conjunction with, or as part of, a browser application, for management of image storage, some images may be initially received in formats other than "raw" image arrays. For instance, some images may be initially received as JPEG files, or in other proprietary or industry standard formats. To make full use of the capabilities of the present invention, such images are preferably decoded so as to generate reconstructed "raw" image arrays, and then those raw image arrays are wavelet or wavelet-like transformed so as to put the images in a form that enables use of the image quality level management features of the present invention.

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those skilled in the art without departing from the true spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. Image processing apparatus, for use in conjunction with an image capture mechanism, the image processing apparatus comprising:

a memory device for storing a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

image management logic, including data processing circuitry and state machines for storing and processing image data received from the image capture mechanism, the data processing circuitry and state machines including:

image processing circuitry for applying a predefined transform to image data received from the image capture mechanism to generate transform image data and for applying a data compression method to the transform image data so as to generate a new image data structure having an associated image quality level selected from the predefined range of image quality levels; the new image data structure being stored in the memory device;

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures stored in the memory device, and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

2. The image processing apparatus of claim 1, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction circuitry and one or more state machines includes logic for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

3. The image processing apparatus of claim 1, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

4. Image processing apparatus, for use in conjunction with an image capture mechanism, the image processing apparatus comprising:

a memory device for storing a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

image management logic for storing and processing image data received from the image capture mechanism, including:

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including instructions for storing and processing image data received from the image capture mechanism, the instructions including:

an initial image processing procedure for applying a predefined transform to image data received from the image capture mechanism to generate transform image data and for applying a data compression procedure to the transform image data so as to generate an

image data structure having an associated image quality level selected from the predefined range of image quality levels;

an image size reduction procedure for lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in the first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

5. The image processing apparatus of claim 4, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction instructions include instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

6. The image processing apparatus of claim 4, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction instructions include instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at

least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

7. Image processing apparatus, comprising:

image management logic, including:

image processing circuitry for applying a predefined transform to an array of image data so as to generate transform image data and for applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

the image management logic further including:

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures stored in the memory device, and forming a reduced size version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure and that has a lower associated image quality level than the image quality level associated with the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by the image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

8. The image processing apparatus of claim 7, further including a communications interface for receiving the image data from another apparatus.

9. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

10. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

11. Image processing apparatus, comprising:

a communications interface for receiving an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

image management logic, including:

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures to form a reduced size image data structure that occupies less space in the memory device than was previously occupied by the first specified

image data structure and that has a lower associated image quality level than the quality level associated with the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures and the reduced size image data structure so as to generate a reconstructed image suitable for display on a display device;

wherein the amount of space occupied by the image data structure in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

12. The image processing apparatus of claim 11, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of the first specified image data structure that excludes the image transform data for at least one bit plane and for replacing the first specified image data structure with an image data structure containing the extracted portion.

13. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

14. Image processing apparatus, the image processing apparatus comprising:

a communications interface for receiving an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a

highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including instructions for storing and processing image data, the instructions including:

an image size reduction procedure for lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in the first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

15. The image processing apparatus of claim 14, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction instructions include instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

16. The image processing apparatus of claim 14, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is

stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction instructions include instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

17. A computer program product, for use in conjunction with a computer system having a memory in which image data structures can be stored, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

an image handling procedure, including instructions for storing in the memory of the computer system a plurality of image data structures,

an image size reduction procedure for accessing image data structures in the memory of the computer system, each of the image data structures containing image transform data, lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in a first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression procedure and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

18. The computer program product of claim 17, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction procedure includes instructions for extracting a portion of the first specified image data structure that excludes the image transform data for at least one bit plane and for replacing the first specified image data structure with an image data structure containing the extracted portion.

19. The computer program product of claim 17, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction procedure includes instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

20. The computer program product of claim 17, wherein the image handling procedure includes one or more image processing procedures for applying a predefined transform to raw image data to generate transform image data and for applying a data compression procedure to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels.

21. A method of processing images, comprising:

storing in a memory device a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

reducing the size of a specified one of the image data structures stored in the nonvolatile memory device, including extracting a subset of the data in the specified image data structure and forming a lower quality version of the specified image data structure that

occupies less space in the nonvolatile memory device than was previously occupied by the specified image data structure; and

successively applying a data decompression method and an inverse transform to a specified one of the image data structures stored in the nonvolatile memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the nonvolatile memory device can be reduced so as to make room for the storage of additional image data structures in the nonvolatile memory device.

22. The method of claim 21, wherein the method is performed by a digital camera and the method includes applying a predefined transform to image data received from an image capture mechanism in the digital camera to generate transform image data, applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels, and storing the image data structure in the memory device.

23. The method of claim 21, wherein the method includes applying a predefined transform to raw image data to generate transform image data, applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels, and storing the image data structure in the memory device.

24. The method of claim 21, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the size reduction step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure in the nonvolatile memory device with an image data structure containing the extracted portion.

25. The method of claim 21, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the size reduction step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and replacing the first specified image data structure with an image data structure containing the extracted portion.

26. Video image processing apparatus, comprising:

a memory device for storing a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels; and

image management logic including:

image size reduction circuitry for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

27. The video image processing apparatus of claim 26, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of an image data structure that excludes the image transform data for at

least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

28. The video image processing apparatus of claim 26, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

29. The video image processing apparatus of claim 26, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

30. The video image processing apparatus of claim 29, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

31. The video image processing apparatus of claim 26, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

32. Video image processing apparatus, comprising:

image management logic, including:

image processing circuitry for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate a set of image data structures having an associated

image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the set of image data structures;

the image management logic further including:

image size reduction circuitry for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

33. The video image processing apparatus of claim 32, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

34. The video image processing apparatus of claim 32, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the

image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

35. The video image processing apparatus of claim 32, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

36. The video image processing apparatus of claim 35, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

37. The video image processing apparatus of claim 32, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

38. Video image processing apparatus, comprising:

a memory device for storing a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

39. The video image processing apparatus of claim 38, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

40. The video image processing apparatus of claim 38, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

41. The video image processing apparatus of claim 38, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

42. The video image processing apparatus of claim 41, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

43. The video image processing apparatus of claim 38, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

44. Video image processing apparatus, comprising:

a memory device for storing image data structures;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including:

at least one image processing procedure for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate a set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels, and for storing the set of image data structures in the memory device;

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

45. The video image processing apparatus of claim 44, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

46. The video image processing apparatus of claim 44, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

47. The video image processing apparatus of claim 44, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

48. The video image processing apparatus of claim 47, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

49. The video image processing apparatus of claim 44, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

50. A computer program product, for use in conjunction with a computer system having a memory in which image data structures can be stored, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

an image handling procedure, including instructions for storing in the memory of the computer system a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a data processor coupled to the memory device;

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

51. The computer program product of claim 50, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit

plane and for replacing the image data structure with an image data structure containing the extracted portion.

52. The computer program product of claim 50, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

53. The computer program product of claim 50, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

54. The computer program product of claim 53, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

55. The computer program product of claim 50, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

56. The computer program product of claim 50, including:

at least one image processing procedure for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate the set of image data structures stored in the memory.

57. The computer program product of claim 56, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

58. The computer program product of claim 56, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

59. The computer program product of claim 56, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

60. The computer program product of claim 59, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

61. The computer program product of claim 56, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

62. A method of processing video images, comprising:

storing in a memory device a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

successively applying a data decompression method and an inverse transform to the specified set of image data structures so as to generate a reconstructed sequence of video images suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

63. The method of claim 62, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the extracting step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane, and the forming step includes replacing the image data structure with an image data structure containing the extracted portion.

64. The method of claim 62, wherein

the of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the extracting step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer, and the forming step includes replacing the first specified image data structure with an image data structure containing the extracted portion.

65. The method of claim 62, after performing the extracting and forming steps, applying the predefined transform to a sequence of additional video images to generate transform image data and applying the data compression method to the transform image data so as to generate an additional set of image data structures having an associated image quality, and storing the additional set of image data structures in the memory device.

66. The method of claim 62, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

67. The method of claim 66, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

68. The method of claim 62, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

69. The method of claim 62, including applying a predefined transform to a sequence of video images to generate transform image data and applying a data compression method to the transform image data so as to generate the set of image data structures stored in the memory device.

70. The method of claim 69, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the extracting step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane, and the forming step includes replacing the image data structure with an image data structure containing the extracted portion.

71. The method of claim 69, wherein

the of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the extracting step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer, and the forming step includes replacing the first specified image data structure with an image data structure containing the extracted portion.

72. The method of claim 69, after performing the extracting and forming steps, applying the predefined transform to a sequence of additional video images to generate transform image data and applying the data compression method to the transform image data so as to generate an additional set of image data structures having an associated image quality, and storing the additional set of image data structures in the memory device.

73. The method of claim 69, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

74. The method of claim 73, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

75. The method of claim 69, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

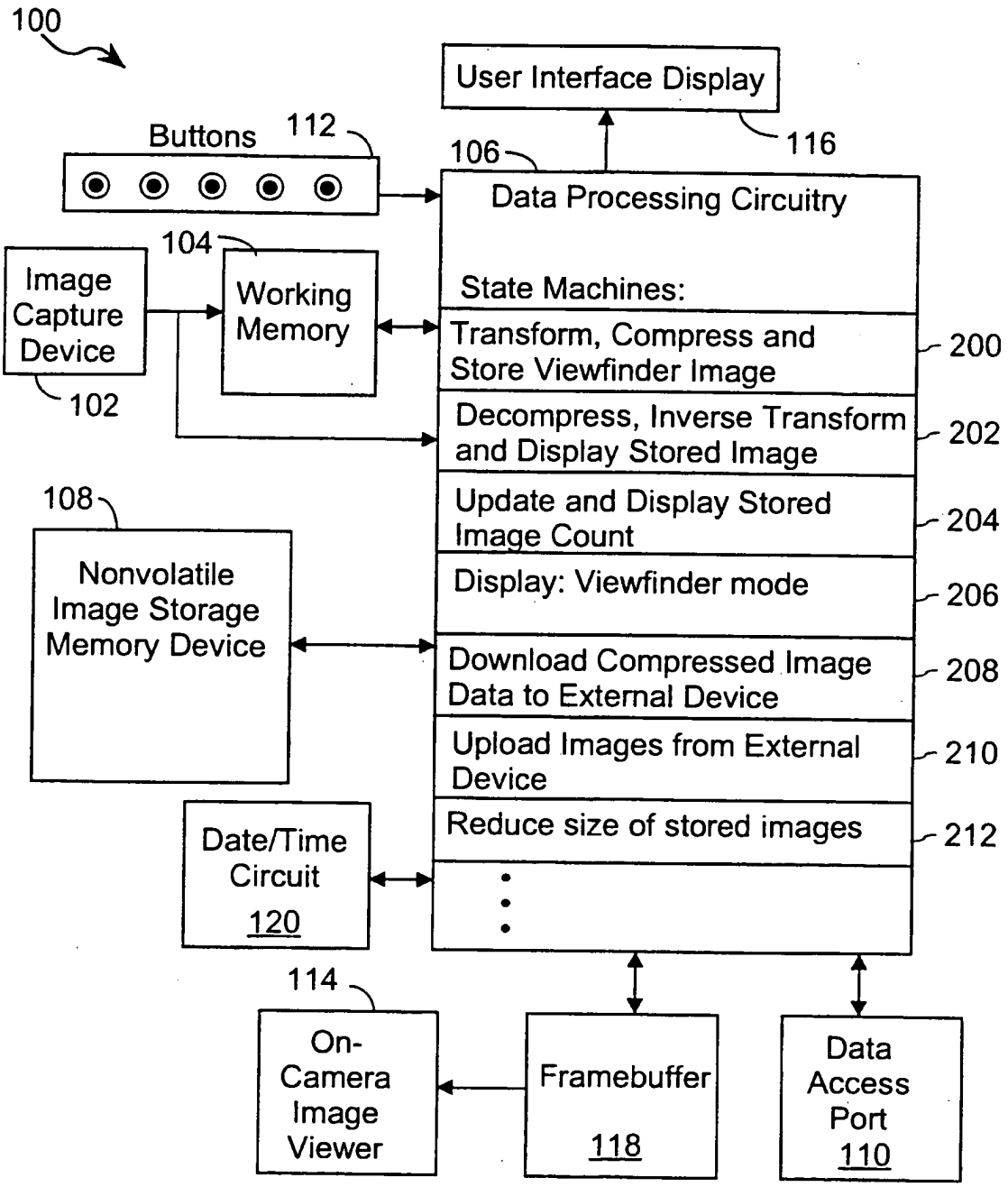

100
User Interface Display
116
Buttons
112
106
Data Processing Circuitry
State Machines:
104
Image Capture Device
Working Memory
102
Transform, Compress and Store Viewfinder Image
200
Decompress, Inverse Transform and Display Stored Image
202
Update and Display Stored Image Count
204
108
Nonvolatile Image Storage Memory Device
Display: Viewfinder mode
206
Download Compressed Image Data to External Device
208
Upload Images from External Device
210
Reduce size of stored images
212
Date/Time Circuit
120
114
On-Camera Image Viewer
Framebuffer
118
Data Access Port
110

FIG. 1

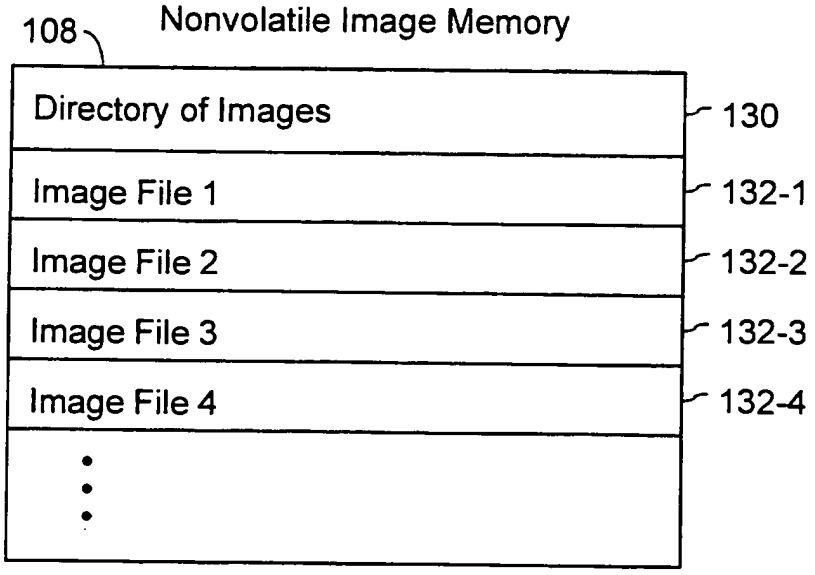

108
Nonvolatile Image Memory
Directory of Images
130
Image File 1
132-1
Image File 2
132-2
Image File 3
132-3
Image File 4
132-4

FIG. 2

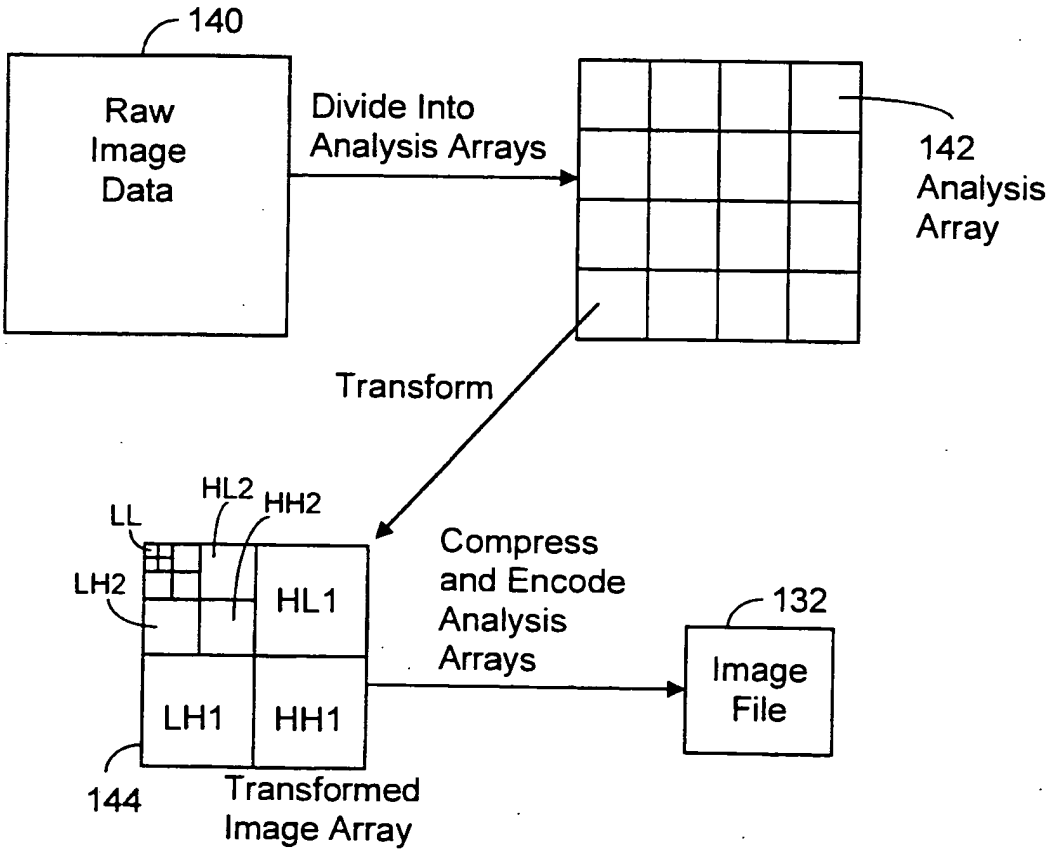

140
Raw Image Data
Divide Into Analysis Arrays
142 Analysis Array
Transform
LL
HL2
HH2
LH2
HL1
LH1
HH1
144
Transformed Image Array
Compress and Encode Analysis Arrays
132
Image File

FIG. 3

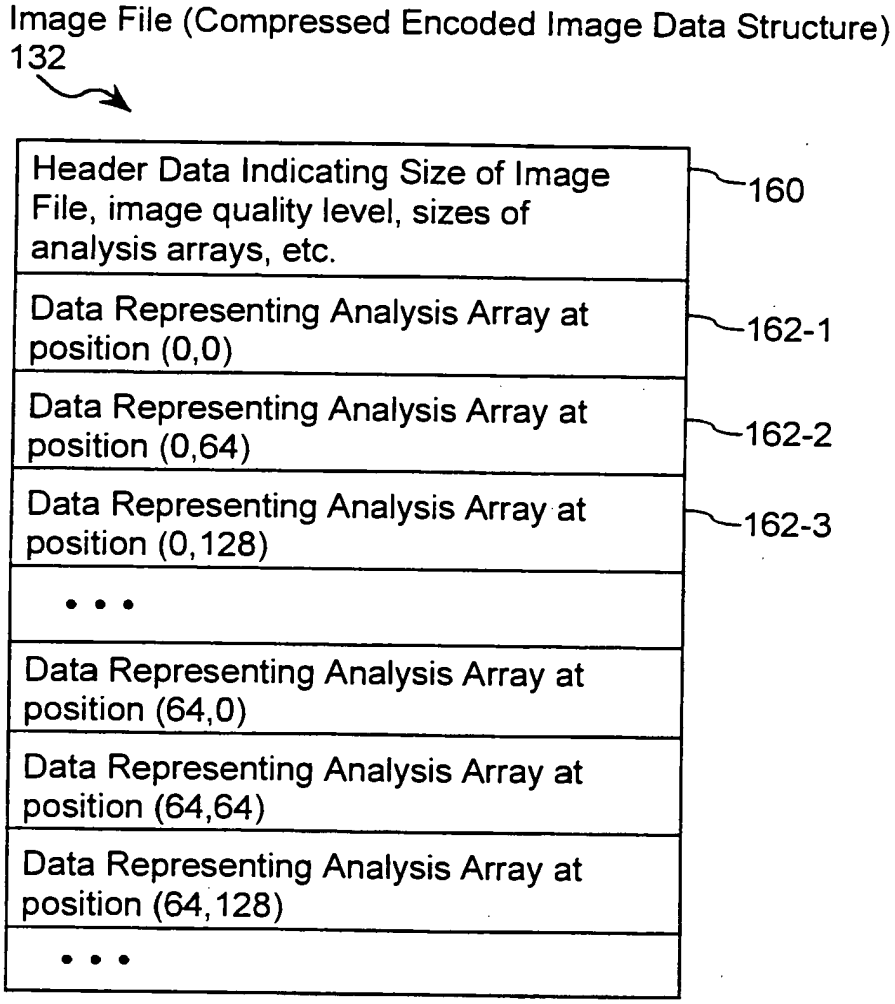

Image File (Compressed Encoded Image Data Structure)
132
Header Data Indicating Size of Image File, image quality level, sizes of analysis arrays, etc.
160
Data Representing Analysis Array at position (0,0)
162-1
Data Representing Analysis Array at position (0,64)
162-2
Data Representing Analysis Array at position (0,128)
162-3
• • •
Data Representing Analysis Array at position (64,0)
Data Representing Analysis Array at position (64,64)
Data Representing Analysis Array at position (64,128)
• • •

FIG. 4A

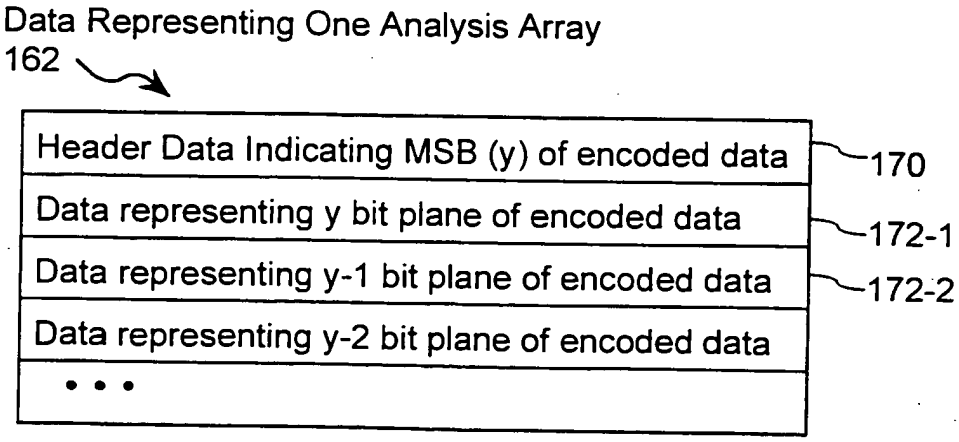

Data Representing One Analysis Array
162
Header Data Indicating MSB (y) of encoded data
170
Data representing y bit plane of encoded data
172-1
Data representing y-1 bit plane of encoded data
172-2
Data representing y-2 bit plane of encoded data
• • •

FIG. 4B

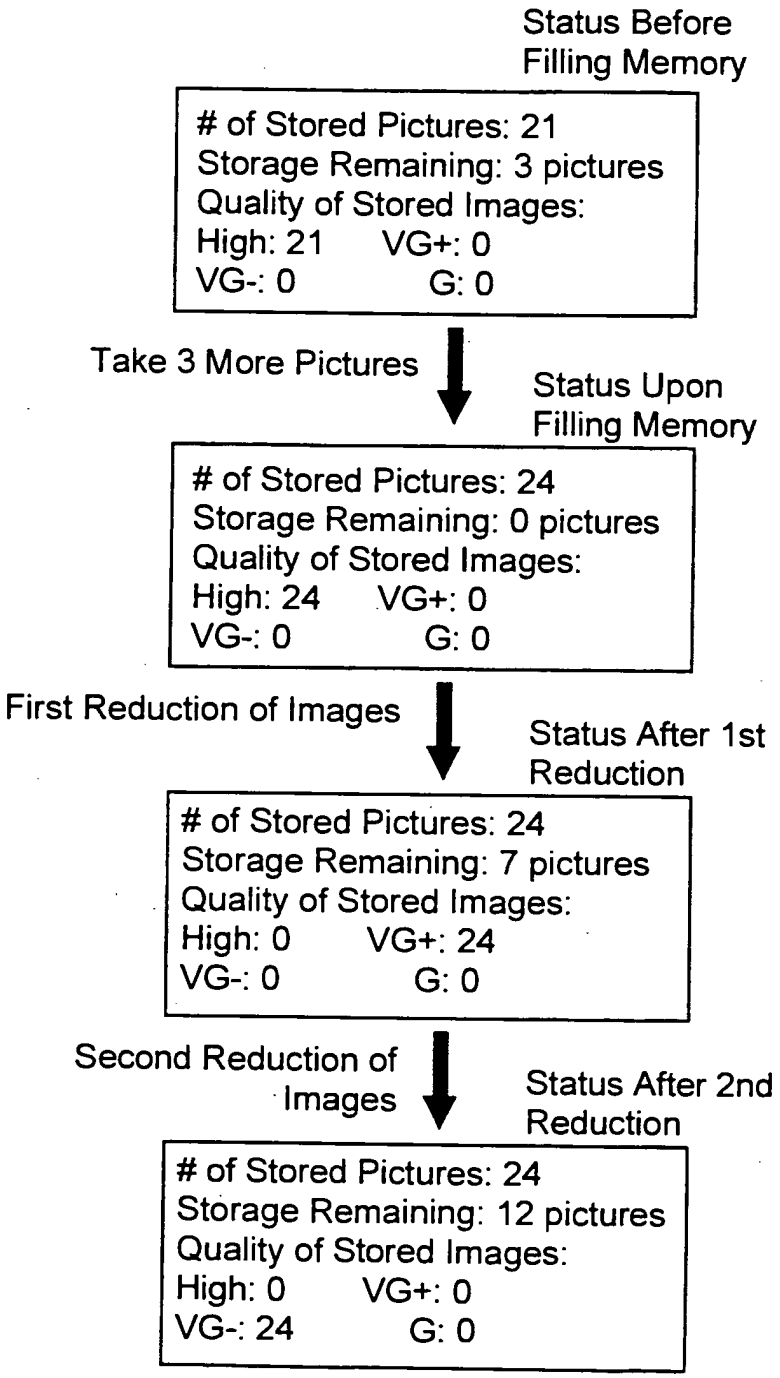

Status Before Filling Memory
# of Stored Pictures: 21
Storage Remaining: 3 pictures
Quality of Stored Images:
High: 21 VG+: 0
VG-: 0 G: 0
Take 3 More Pictures
Status Upon Filling Memory
# of Stored Pictures: 24
Storage Remaining: 0 pictures
Quality of Stored Images:
High: 24 VG+: 0
VG-: 0 G: 0
First Reduction of Images
Status After 1st Reduction
# of Stored Pictures: 24
Storage Remaining: 7 pictures
Quality of Stored Images:
High: 0 VG+: 24
VG-: 0 G: 0
Second Reduction of Images
Status After 2nd Reduction
# of Stored Pictures: 24
Storage Remaining: 12 pictures
Quality of Stored Images:
High: 0 VG+: 0
VG-: 24 G: 0

FIG. 5

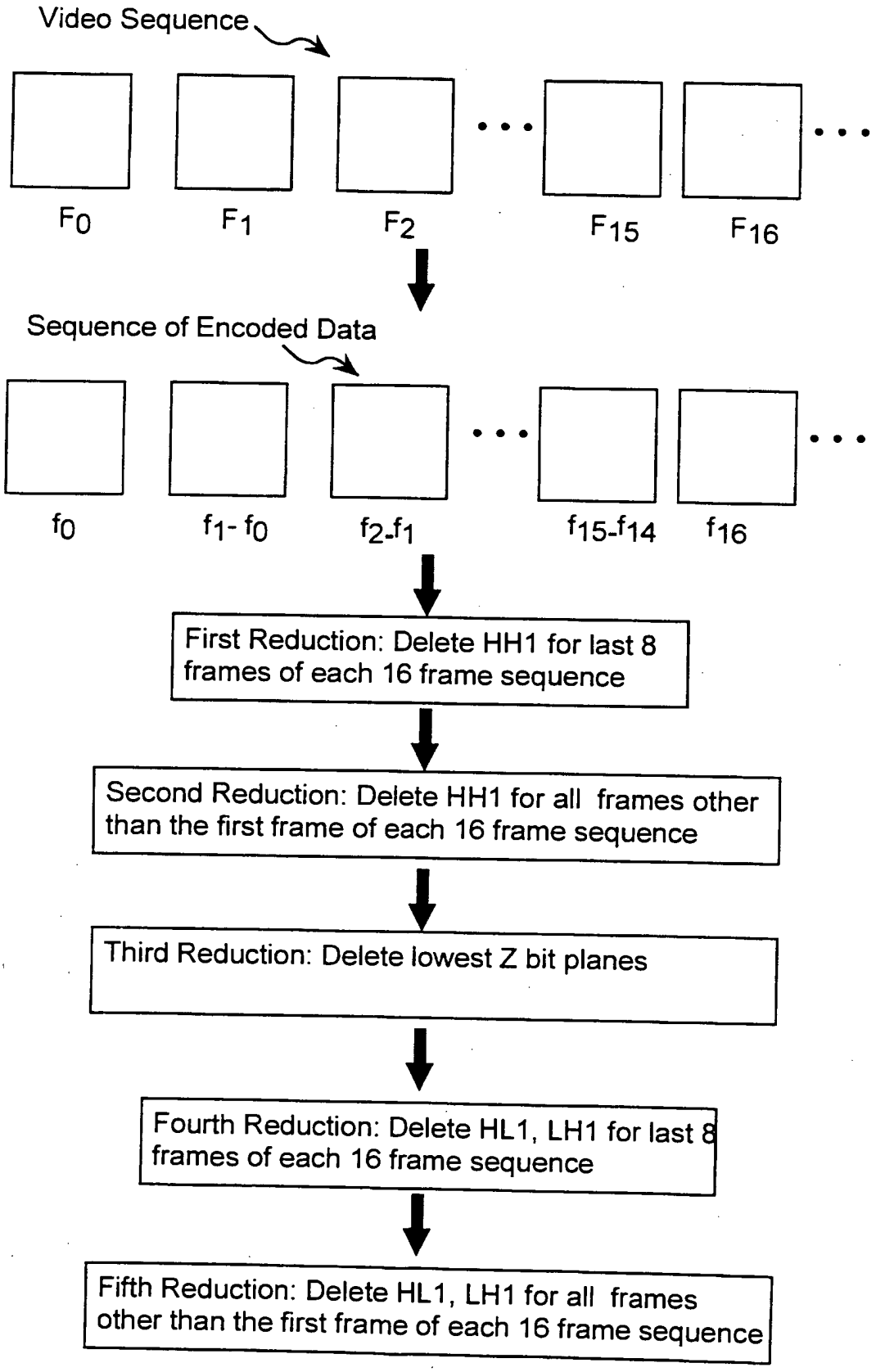

Video Sequence
F0
F1
F2
F15
F16
Sequence of Encoded Data
f0
f1- f0
f2-f1
f15-f14
f16
First Reduction: Delete HH1 for last 8 frames of each 16 frame sequence
Second Reduction: Delete HH1 for all frames other than the first frame of each 16 frame sequence
Third Reduction: Delete lowest Z bit planes
Fourth Reduction: Delete HL1, LH1 for last 8 frames of each 16 frame sequence
Fifth Reduction: Delete HL1, LH1 for all frames other than the first frame of each 16 frame sequence

FIG. 6

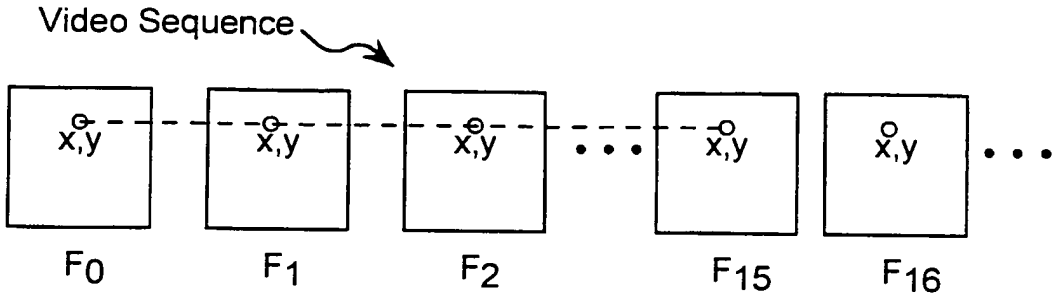


FIG. 7

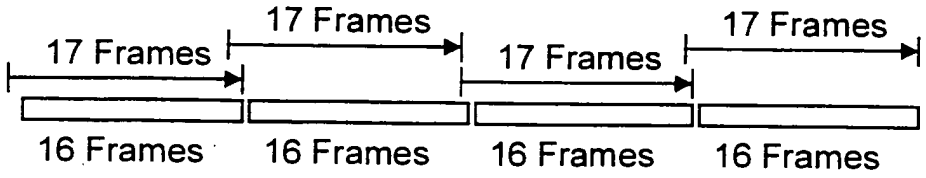


FIG. 8

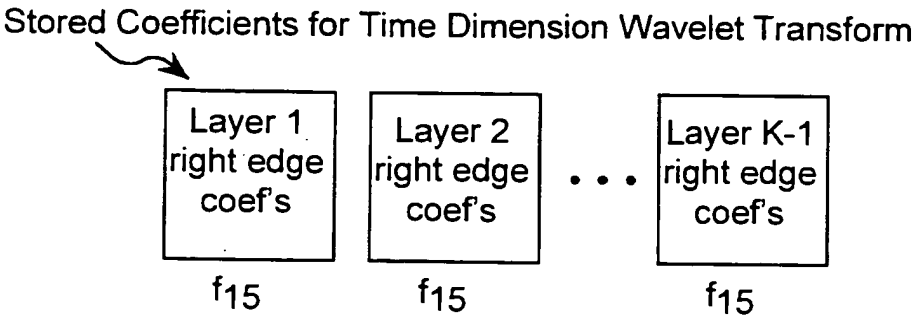


FIG. 9

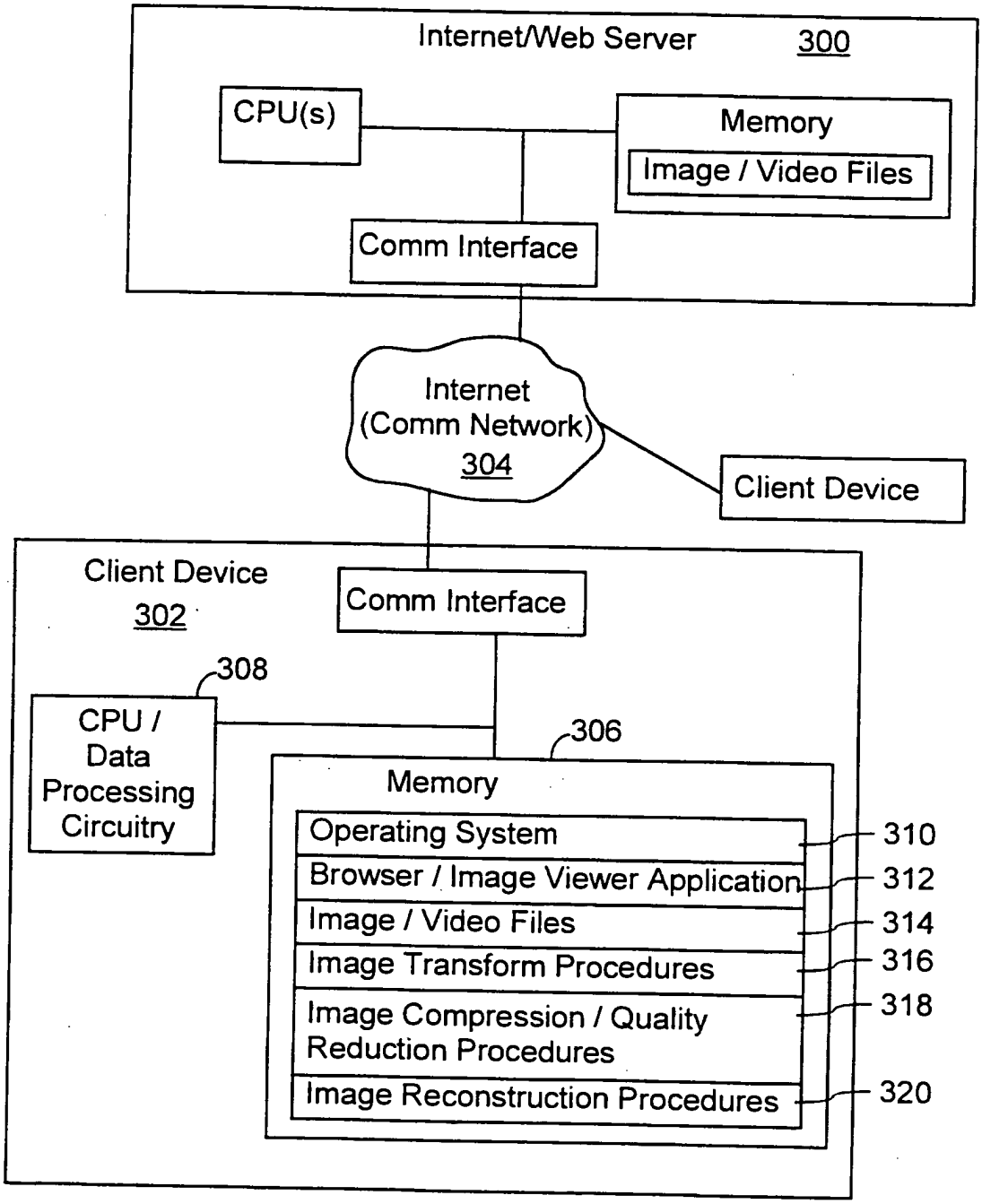

Internet/Web Server
300
CPU(s)
Memory
Image / Video Files
Comm Interface
Internet
(Comm Network)
304
Client Device
Client Device
302
Comm Interface
308
CPU /
Data
Processing
Circuitry
306
Memory
Operating System
Browser / Image Viewer Application
Image / Video Files
Image Transform Procedures
Image Compression / Quality
Reduction Procedures
Image Reconstruction Procedures
310
312
314
316
318
320

FIG. 10

INTERNATIONAL SEARCH REPORT

Internal application No.
PCT/US00/30825

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06K 9/36, 9/46
US CL : 382/232

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : Please See Continuation Sheet

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WEST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,867,602 A (ZANDI ET AL.) 02 February 1999 (02.02.99).	1-75
A	US 5,881,176 A (KEITH ET AL.) 09 March 1999 (09.03.1999).	1-75
A	US 5,966,465 A (KEITH ET AL.) 12 October, 1999 (12.10.1999).	1-75

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
04 January 2001 (04.01.2001)	30 MAR 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer Jose L. Couso Telephone No. (703)305-8576

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

Inter ional application No.
PCT/US00/30825

Continuation of B. FIELDS SEARCHED Item 1: 382/232, 233, 234, 240, 244, 246, 247, 248, 260, 263, 264, 276; 341/51, 63, 65, 67, 107; 364/724.011, 724.04, 724.05, 724.13, 724.14, 725.01, 725.02.

Petitioner Apple Inc. - Exhibit 1004, p. 5845

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 257 585**
A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **87112158.8**

(51) Int. Cl.⁴: **H04L 9/00**

(22) Date of filing: **21.08.87**

(30) Priority: **22.08.86 JP 197610/86**
22.08.86 JP 197611/86

(43) Date of publication of application:
02.03.88 Bulletin 88/09

(84) Designated Contracting States:
BE DE FR GB

(71) Applicant: **NEC CORPORATION**
33-1, Shiba 5-chome, Minato-ku
Tokyo 108(JP)

(72) Inventor: **Okamoto, Eiji c/o NEC Corporation**
33-1, Shiba 5-chome
Minato-ku Tokyo(JP)

(74) Representative: **Vossius & Partner**
Siebertstrasse 4 P.O. Box 86 07 67
D-8000 München 86(DE)

(54) Key distribution method.

(57) The invention relates to a method of distributing a key for enciphering un unenciphered or plaintext message and for deciphering the enciphered message.

The method comprises the following steps: generating a first random number in a first system (101); generating first key distribution information in the first system (101) by applying a predetermined first transformation to the first random number on the basis of first secret information known only by the first system (101); transmitting the first key distribution information to a second system (102) via a communication channel (103); receiving the first key distribution information in the second system (102); generating a second random number in the second system (102); generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of second secret information known only by the second system (102); transmitting the second key distribution information to the first system (101) via the channel (103); receiving the second key distribution information in the first system (101); and generating an enciphering key in the first system (101) by applying a predetermined second transformation to the second key distribution information on the basis of the first random number and identification information of the second system (102) which is not secret.

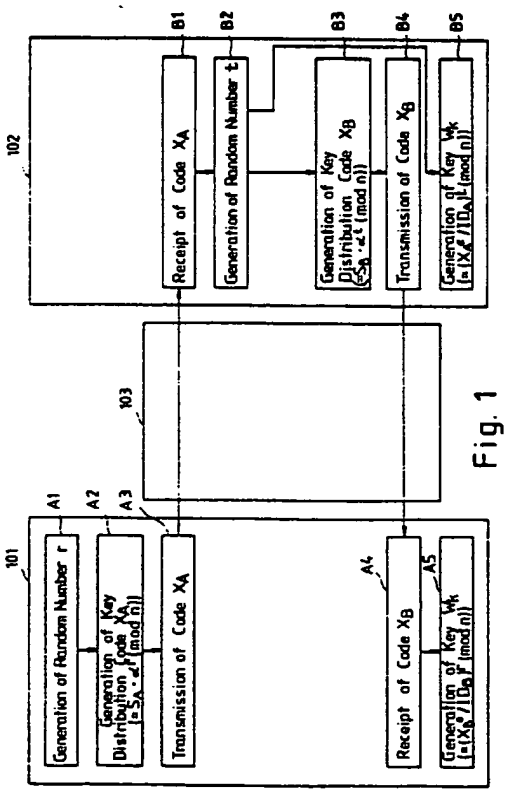


Fig. 1

EP 0 257 585 A2

Petitioner Apple Inc. - Exhibit 1004, p. 5846

KEY DISTRIBUTION METHOD

BACKGROUND OF THE INVENTION

The invention relates to a method of distributing a key for enciphering an unenciphered or plaintext message and for deciphering the enciphered message.

A public key distribution method used in a public key cryptosystem as a well-known key distribution method is disclosed in a paper entitled "New Directions in Cryptography" by W. Diffie and M.E. Hellman, published in the IEEE Transactions on Information Theory, Vol. IT-22, No. 6, pp. 644 to 654, November issue, 1976. The key distribution method disclosed in the paper memorizes public information for each of conversers. In the system, before a converser A sends an enciphered message to a converser B, the converser A prepares an enciphering key (which represents a number obtained by calculating $Y_B{}^{X_A}$ (mod p)) generated from public information Y_B of the converser B and secret information X_A which is kept secret by the converser A. The number p is a large prime number of about 256 bits in binary representation, which is publicly known. a (mod b) means a remainder of division of the number a by the number b. The converser B also prepares the key wk in accordance to $Y_A{}^{X_B}$ (mod p) in a similar manner. Y_A and Y_B are selected so as to be equal to α^{X_A} (mod p) and α^{X_B} (mod p), respectively. As a result, $Y_B{}^{X_A}$ (mod p) becomes equal to $Y_A{}^{X_B}$ (mod p). It is known that even if Y_A, a and p are known, it is infeasible for anybody except the converser A to obtain X_A which satisfies $Y_A = \alpha^{X_A} \pmod{p}$.

The prior art key distribution system of the type described, however, has disadvantages in that since the system needs a large amount of public information corresponding to respective conversers, the amount of the public information increases as the number of conversers increases. Further, strict control of such information becomes necessary to prevent the information from being tampered.

SUMMARY OF THE INVENTION

An object of the invention is, therefore, to provide a key distribution method free from the above-mentioned disadvantages of the prior art system.

According to an aspect of the invention, there is provided a method which comprises the following steps: generating a first random number in a first system; generating first key distribution information in the first system by applying a predetermined first transformation to the first random number on the basis of first secret information known only by the first system; transmitting the first key distribution information to a second system via a communication channel; receiving the first key distribution information in the second system; generating a second random number in the second system; generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of second secret information known only by the second system; transmitting the second key distribution information to the first system via the channel; receiving the second key distribution information in the first system; and generating an enciphering key in the first system by applying a predetermined second transformation to the second key distribution information on the basis of the first random number and identification information of the second system which is not secret.

According to another aspect of the invention, there is provided a method which comprises the following steps: generating a first random number in the first system; generating first key distribution information by applying a predetermined first transformation to the first random number on the basis of public information in the first system and generating first identification information by applying a predetermined second transformation to the first random number on the basis of first secret information known only by the first system; transmitting the first key distribution information and the first identification information to a second system via a communication channel; receiving the first key distribution information and the first identification information in the second system; examining whether or not the result obtained by applying a predetermined third transformation to the first key distribution information on the basis of the first identification information satisfies a first predetermined condition, and, if it does not satisfy, suspending key distribution processing; generating a second random number if said condition is satisfied in the preceding step; generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of the public information, and generating second identification information by applying the predetermined second transformation to the second random number on the basis of second secret information known only by the second system; transmitting the second key distribution information and the second identification information to the first system via the communication channel; and exam-

Petitioner Apple Inc. - Exhibit 1004, p. 5847

ining whether or not the result obtained by applying a third predetermined transformation to the second key distribution information on the basis of the second identification information in the first system satisfies a predetermined second condition, and if the result does not satisfy the second condition, suspending the key distribution processing, or if it satisfies the second condition, generating an enciphering key by applying a fourth predetermined transformation to the first random number on the basis of the second key distribution information.

BRIEF DESCRIPTION OF THE DRAWINGS

Other features and advantages of the invention will become more apparent from the following detailed description when taken in conjunction with the accompanying drawings in which:

FIG. 1 is a block diagram of a first embodiment of the invention;

FIG. 2 is a block diagram of a second embodiment of the invention; and

FIG. 3 is a block diagram of an example of systems 101, 102, 201 and 202.

In the drawings, the same reference numerals represent the same structural elements.

PREFERRED EMBODIMENTS

Referring now to FIG. 1, a first embodiment of the invention comprises a first system 101, a second system 102 and an insecure communication channel 103 such as a telephone line which transmits communication signals between the systems 101 and 102. It is assumed herein that the systems 101 and 102 are used by users or conversers A and B, respectively. The user A has or knows a secret integer number S_A and public integer numbers e, c, α and n which are not necessarily secret while the user B has or knows a secret integer number $S_B$and the public integer numbers. These integer numbers are designated and distributed in advance by a reliable person or organization. The method to designate the integer numbers will be described later.

An operation of the embodiment will next be described on a case in which the user A starts communication. The system 101 of the user A generates a random number γ (Step A1 in FIG. 1) and sends a first key distribution code X_A representative of a number obtained by computing $S_A \cdot \alpha^{\gamma} \pmod{n}$ (Step A2) to the system 102 of the user B (step A3). Next, when the system 102 receives the code X_A(Step B1), it generates a random number t (Step B2), calculates $(X_A^e/ID_A)^t \pmod{n}$ (Step B5), and keeps the resulting number as a enciphering key wk for enciphering a message into storage means (not shown). The identification code ID_A represents herein a number obtained by considering as a numeric value a code obtained by encoding the address, the name and so on of the user A. The encoding is, for instance, performed on the basis of the American National Standard Code for Information Interchange. Then, the system 102 transmits to the system 101 of the user A a second key distribution code X_B representative of a number obtained by calculating $S_B \cdot \alpha^t \pmod{n}$ (Steps B3 and B4).

The system 101, on the other hand, receives the code X_B (Step A4), calculates $(X_B^e/ID_B)^{\gamma} \pmod{n}$ (Step A5), and keeps the resulting number as the key wk for enciphering a message. The identification code ID_B represents the numbers obtained by considering as a numeric value a code obtained by encoding the name, address, and so on of the user B.

Subsequently, communication between the users A and B will be conducted by transmitting messages enciphered with the enciphering key wk via the channel 103.

The integer numbers S_A, S_B, e, c, α and n are determined as follows. n is assumed to be a product of two sufficiently large prime numbers p and q. For instance, p and q may be 2^{256} or so. e and c are prime numbers which are equal to or less than n, while α is a positive integer number which is equal to or less than n. Further, d is defined as an integer number which satisfies $e.d \pmod{(p-1)\cdot(q-1)} = 1$. S_A and S_B are defined as numbers obtainable from $ID_A^d \pmod{n}$ and $ID_B^d \pmod{n}$, respectively.

If S_A, S_B, e, c, α, and n are defined as above, ID_A and ID_B become equal to $S_A^e \pmod{n}$ and $S_B^e \pmod{n}$, respectively. This can be proved from a paper entitled "A Method for Obtaining Digital Signatures and Publick-Key Cryptosystems" by R.L. Rivest et al., published in the Communication of the ACM, Vol. 21, No. 2, pp. 120 to 126. Since the key obtained by $(X_B^e/ID_B)^r \pmod{n}$ on the side of the user A becomes equal to $\alpha^{ert} \pmod{n}$ and the key obtained by $(X_A^e/ID_A)^t \pmod{n}$ on the side of the user B becomes equal to $\alpha^{ert} \pmod{n}$, they can prepare the same enciphering key. Even if a third party tries to assume the identity of the user A, he cannot prepare the key wk since he cannot find out z which meets $ID_A = Z^e \pmod{n}$.

Referring now to FIG. 2, a second embodiment of the invention comprises a first system 201, a second system 202 and an insecure communication channel 203. It is assumed herein that the systems 201 and 202 are used by users A and B, respectively. The user A has or knows a secret integer number S_A and public integer numbers e, c, α, and n, which are not necessarily secret while

Petitioner Apple Inc. - Exhibit 1004, p. 5848

the user B has or knows a secret integer number S_B and the public integer numbers. These integer numbers are designated and distributed by a reliable person or organization in advance. The method to designate the integer numbers will be described later.

An operation of the embodiment will next be described on a case where the user A starts communication. The system 201 of the user A generates a random number r (Step AA1 in FIG. 2) and determines a first key distribution code X_A representative of a number obtained by computing $\alpha^{e \cdot r} \pmod{n}$ as well as a first identification code Y_A indicative of a number obtained by computing $S_A \cdot \alpha^{c \cdot r} \pmod{n}$ (AA2). The system 201 then transmits a first pair of X_A and Y_A to the system 202 of the user B (Step AA3). Thereafter, the system 202 receives the first pair (X_A , Y_A) (Step BB1), calculates $Y_A^e / X_A^c \pmod{n}$, and examines whether or not the number obtained by the calculation is identical to the number indicated by an identification code ID_A obtained by the address, the name and so on of the user A in a similar manner to in the first embodiment (Step BB2). If they are not identical to each other, the system suspends processing of the key distribution (Step BB7). On the other hand, if they are identical to each other, the system 202 generates a random number t (Step BB3) and determines a second key distribution code X_B representative of a number obtained by calculating $\alpha^{e \cdot t} \pmod{n}$ and a second identification code Y_B obtained by calculating $S_B \cdot \alpha^{c \cdot t} \pmod{n}$ (Step BB4). The system 202 then transmits a second pair of X_B and Y_B to the system 201 of the user A (Step BB5). The system 202 calculates $X_A^t \pmod{n}$ and keeps the number thus obtained as a enciphering key wk (Step BB6).

The system 201, on the other hand, receives the second pair (X_B, Y_B) (Step AA4), calculates $Y_B^e / X_B^c \pmod{n}$, and examines whether or not the number thus obtained is identical to the number indicated by an identification code ID_B obtained by the address, the name and so on of the user B in a similar manner to in the first embodiment (Step AA5). If they are not identical to each other, the system suspends the key distribution processing (Step AA7). If they are identical to each other, the system 201 calculates $X_B^r \pmod{n}$, and stores the number thus obtained as a enciphering key wk (Step AA6). Although the codes ID_A and ID_B are widely known, they may be informed by the user A to the user B.

The integer numbers S_A, S_B, e, c, α and n are determined in the same manner as in the first embodiment. As a result, ID_A and ID_B becomes equal to $Y_A^e / X_A^c \pmod{n}$ ($= S_A^e \cdot \alpha^{erc} / \alpha^{erc} \pmod{n}$) and $Y_B^e / X_B^c \pmod{n}$ ($= S_B^e \cdot \alpha^{etc} / \alpha^{etc} \pmod{n}$), respectively. If we presuppose that the above-mentioned reliable person or organization who prepared S_A and S_B do not act illegally, since S_A is possessed only by the user A while S_B is possessed only by the user B, the first pair (x_A, y_A) which satisfies $y_A^e / x_A^c \pmod{n} = ID_A$ can be prepared only by the user A while the second pair (x_B , y_B) which satisfies $y_B^e / x_B^c \pmod{n} = ID_B$ can be prepared only by the user B. It is impossible to find out a number x which satisfies $x^f \pmod{n} = b$ on the basis of f, b and n since finding out X is equivalent to breaking the RSA public key cryptogram system disclosed in the above-mentioned the Communication of the ACM. It is described in the above-referenced IEEE Transactions on Information Theory that the key wk cannot be calculated from the codes x_A or x_B and n. The key distribution may be implemented similarly by making the integer number C variable and sending it from a user to another.

An example of the systems 101, 102, 201 and 202 to be used in the first and second embodiments will next be described referring to FIG. 3.

Referring now to FIG. 3, a system comprises a terminal unit (TMU) 301 such as a personal computer equipped with communication processing functions, a read only memory unit (ROM) 302, a random access memory unit (RAM) 303, a random number generator (RNG) 304, a signal processor (SP) 306, and a common bus 305 which interconnects the TMU 301, the ROM 302, the RAM 303, the RNG 304 and the SP 306.

The RNG 304 may be a key source 25 disclosed in U.S. Patent No. 4,200,700. The SP 306 may be a processor available from CYLINK Corporation under the trade name CY 1024 KEY MANAGEMENT PROCESSOR.

The RNG 304 generates random numbers r or t by a command given from the SP 306. The ROM 407 stores the public integer numbers e , c, α, n and the secret integer number S_A (if the ROM 407 is used in the system 101 or 201) or the secret integer number S_B (if the ROM 407 is used in the system 102 or 202). The numbers S_A and S_B may be stored in the RAM 303 from the TMU 301 everytime users communicates. According to a program stored in the ROM 407, the SP 306 executes the above-mentioned steps A2, A5, AA2, AA5, AA6 and AA7 (if the SP 306 is used in the system 101 or 201), or the steps B3, B5, BB2, BB4, BB6 and BB7 (if the SP 306 is used in the system 102 or 202). The RAM 303 is used to temporarily store calculation results in these steps.

Each of the systems 101, 102, 201 and 202 may be a data processing unit such as a general purpose computer and an IC (integrated circuit) card.

Petitioner Apple Inc. - Exhibit 1004, p. 5849

As described in detail hereinabove, this invention enables users to effectively implement key distribution simply with a secret piece of information and several public pieces of information.

While this invention has thus been described in conjunction with the preferred embodiments thereof, it will now readily be possible for those skilled in the art to put this invention into practice in various other manners.

Claims

1. A key distribution method comprising the following steps:

a) generating a first random number in a first system;

b) generating first key distribution information in said first system by applying a predetermined first transformation to said first random number on the basis of first secret information known only by said first system;

c) transmitting said first key distribution information to a second system via a communication channel;

d) receiving said first key distribution information in said second system;

e) generating a second random number in said second system;

f) generating second key distribution information by applying said predetermined first transformation to said second random number on the basis of second secret information known only by said second system;

g) transmitting said second key distribution information to said first system via said channel;

h) receiving said second key distribution information in said first system; and

i) generating an enciphering key in said first system by applying a predetermined second transformation to said second key distribution information on the basis of said first random number and identification information of said second system which is not secret.

2. A key distribution method as claimed in Claim 1, in which said first system includes first data processing means for executing said steps a), b) and i), and first communication processing means for executing said steps c) and h).

3. A key distribution method as claimed in Claim 1 or 2, in which said second system includes second data processing means for executing said steps e) and f), and second communication processing means for executing said steps d) and g).

4. A key distribution method comprising the following steps:

a) generating a first random number in a first system;

b) generating first key distribution information in said first system by applying a predetermined first transformation to said first random number on the basis of public information and generating first identification information by applying a predetermined second transformation to said first random number on the basis of first secret information known only by said first system;

c) transmitting said first key distribution information and said first identification information to a second system via a communication channel;

d) receiving said first key distribution information and said first identification information in said second system;

e) examining whether or not the result obtained by applying a predetermined third transformation to said first key distribution information on the basis of said first identification information satisfies a predetermined first condition and, if it does not satisfy, suspending key distribution processing;

f) generating a second random number if said first condition is satisfied at said step e);

g) generating second key distribution information by applying said predetermined first transformation to said second random number on the basis of said public information, and generating second identification information by applying said predetermined second transformation to said second random number on the basis of second secret information known only by said second system;

h) transmitting said second key distribution information and said second identification information to said first system via said communication channel; and

i) examining in said first system whether or not the result obtained by applying a predetermined third transformation to said second key distribution information on the basis of said second identification information satisfies a predetermined second condition and, if the result does not satisfy said second condition, suspending said key distribution processing or, if it satisfies said second condition, generating said enciphering key by applying a predetermined fourth transformation to said first random number on the basis of said second key distribution information.

5. A key distribution method as claimed in Claim 4, in which said first system includes first data processing means for executing said steps a), b) and i), and first communication processing means for executing said step c).

6. A key distribution method as claimed in Claim 4 or 5, in which said second system includes second data processing means for executing said steps e), f) and g), and second communication processing means for executing said steps d) and h).

Petitioner Apple Inc. - Exhibit 1004, p. 5850

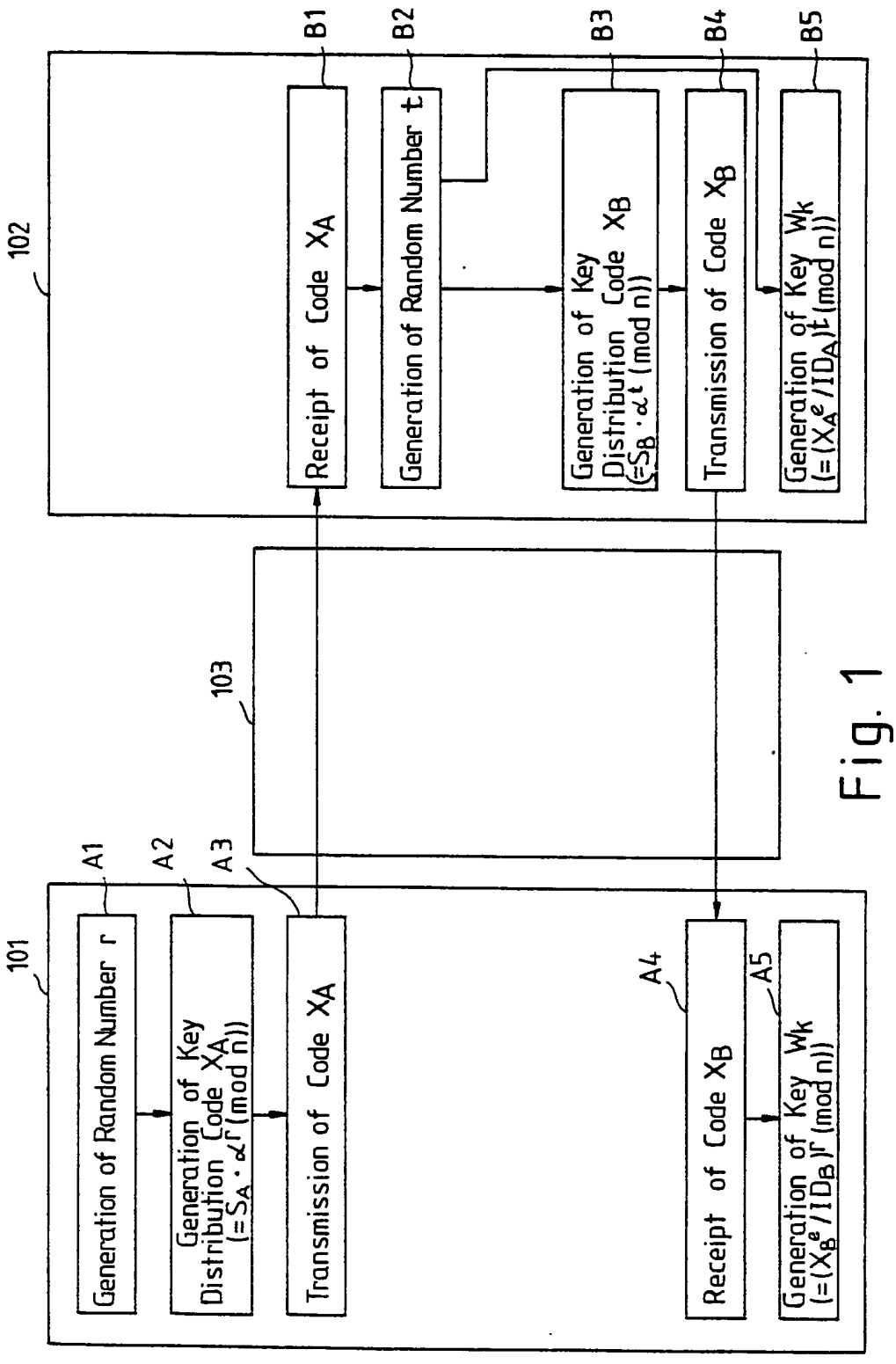

102
B1
B2
B3
B4
B5
Receipt of Code X_A
Generation of Random Number t
Generation of Key Distribution Code X_B ($=S_B \cdot \alpha^t \pmod n$))
Transmission of Code X_B
Generation of Key W_k ($=(X_A^e/ID_A)^t \pmod n$))
103
101
A1
A2
A3
A4
A5
Generation of Random Number r
Generation of Key Distribution Code X_A ($=S_A \cdot \alpha^r \pmod n$))
Transmission of Code X_A
Receipt of Code X_B
Generation of Key W_k ($=(X_B^e/ID_B)^r \pmod n$))

Fig. 1

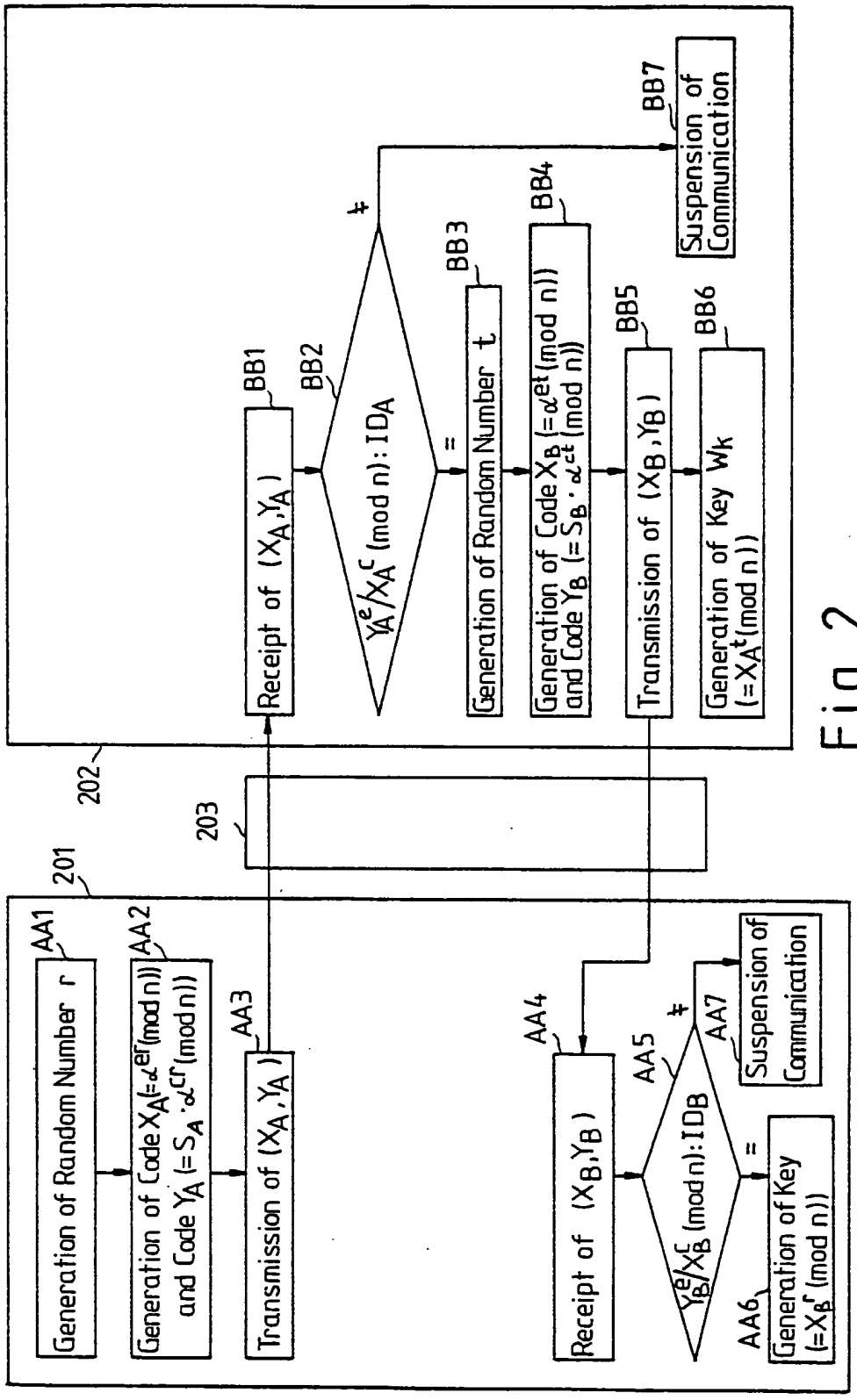

201
202
203
AA1
Generation of Random Number r
AA2
Generation of Code $X_A(=\alpha^{er} \pmod n)$ and Code $Y_A(=S_A \cdot \alpha^{cr} \pmod n)$
AA3
Transmission of (X_A, Y_A)
AA4
Receipt of (X_B, Y_B)
AA5
$Y_B^e/X_B^c \pmod n$: ID_B
AA6
Generation of Key $(=X_B^r \pmod n)$
AA7
Suspension of Communication
BB1
Receipt of (X_A, Y_A)
BB2
$Y_A^e/X_A^c \pmod n$: ID_A
BB3
Generation of Random Number t
BB4
Generation of Code $X_B(=\alpha^{et} \pmod n)$ and Code $Y_B(=S_B \cdot \alpha^{ct} \pmod n)$
BB5
Transmission of (X_B, Y_B)
BB6
Generation of Key W_k $(=X_A^t \pmod n)$
BB7
Suspension of Communication

Fig. 2

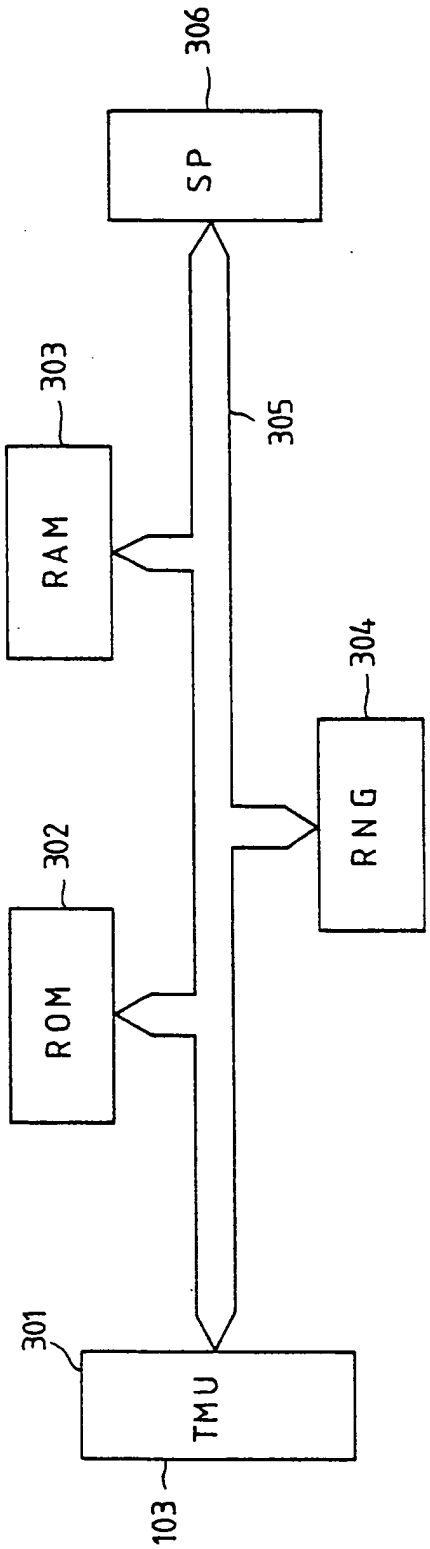


Fig. 3

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 November 2000 (30.11.2000)

PCT

(10) International Publication Number
WO 00/72118 A1

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/US00/10213

(22) International Filing Date: 13 April 2000 (13.04.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/316,717 21 May 1999 (21.05.1999) US

(71) Applicant: COMPAQ COMPUTERS INC. [US/US]; 10435 N. Tautau Avenue, Loc 200-16, Cupertino, CA 95014-3548 (US).

(72) Inventors: GLASSMAN, Steven, C.; 615 Palo Alto Avenue, Mountain View, CA 94041 (US). MANASSE, Mark, S.; 1270 Monterey Boulevard, San Francisco, CA 94127 (US).

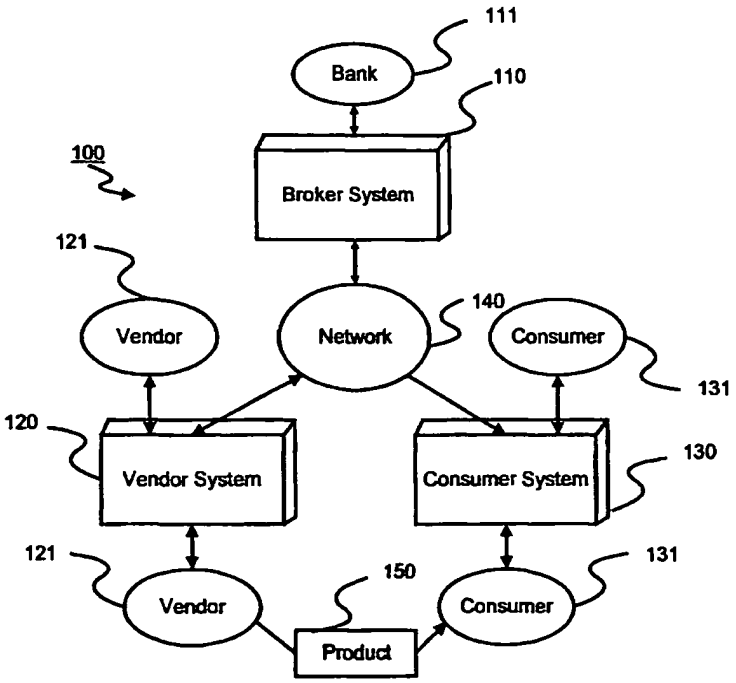
(74) Agents: GRANATELLI, Lawrence; Fenwick & West LLP, Two Palo Alto Square, Palo Alto, CA 94306 et al. (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU,

[Continued on next page]

(54) Title: METHOD AND SYSTEM FOR ENFORCING LICENSES ON AN OPEN NETWORK



(57) Abstract: An electronic commerce system and method enforces a license agreement for content on an open network (140) by restricting the number of consumers (131) that can concurrently access the content. A consumer (131) initially acquires vendor scrip, either from a broker or the vendor (121) itself. The consumer (131) presents the vendor scrip to the vendor (121) along with a request to access the content. In response, the vendor (121) gathers information about the consumer (131) to determine whether the consumer (131) belongs to the class allowed to access the content. The information may be gathered from the scrip or from other sources. If the consumer (131) belongs to the class, then the vendor (121) determines if a license to access the content is available. Generally, a license is available if the number of other consumers (131) having licenses to access the content is less than the maximum specified in the license agreement. If no licenses are available, the vendor (121) provides the consumer (131) with an estimate of when a license will be available. If a license is available, the vendor (121) directs the consumer (131) to obtain license scrip which allows the consumer (131) to access the content. The license scrip expires after a relatively brief period of time. When the consumer (131) uses the license scrip to access the content, the vendor (121) provides the consumer (131) with new license scrip having a later expiration time.

WO 00/72118 A1

MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND SYSTEM FOR ENFORCING LICENSES ON AN OPEN NETWORK

BACKGROUND

FIELD OF THE INVENTION

This invention relates generally to an electronic commerce system and more particularly to a commerce system supporting restricted use of a resource, and even more particularly to a commerce system supporting N-user license agreements.

BACKGROUND OF THE INVENTION

It is common for a library, corporation, or other organization to purchase content that will be made available to members of the organization. Often, the content is subject to a license restriction limiting distribution of the content. For example, a corporation may license or purchase a magazine and then distribute the magazine to interested employees. Typically, the corporation is restricted by the licensing agreement or copyright law from photocopying the magazine. Accordingly, the corporation must either obtain multiple copies of the magazine or circulate the single copy through the organization.

Similarly, the content licensed or purchased by the organization may be in electronic form. For example, the corporation may license a CD-ROM holding an electronic version of the magazine. While the CD-ROM can be loaded onto a server accessible to employees of the corporation via a computer network, the content may be restricted by an N-user license that forbids the corporation from allowing more than N users to simultaneously access the CD-ROM. To implement the restriction, software executing on the server tracks the number of people currently accessing the CD-ROM and blocks usage that exceeds the scope of the license.

Petitioner Apple Inc. - Exhibit 1004, p. 5856

In existing systems, the license control is performed by a combination of a specialized lock server and a client program. The lock server validates users' requests for access to the content and maintains the status of active users. The client program interacts with the lock server to acquire a lock and to provide access to the content.

There are many existing implementations of lock servers. However, they all are subject to one or more of the following undesirable restrictions:

each content source has its own, separate, and proprietary lock server;

the user's system already has the content (protected from direct access) and the client program gets the lock to access the content;

acquiring a lock is a complicated action; and/or

the set of valid users is limited.

For these reasons, existing lock servers are undesirable on an open network.

A lock server providing an N-user license on an open network should also support the following requirements:

an unrestricted set of potential users;

no single administrative domain covers all users;

the users do not need to have a separate user application for each source of content;

access to the content can be easily restricted; and

the content exists on the server and not with the user.

Accordingly, there is a need for a way to provide restricted access to electronic content that works with a wide variety of possible access schemes. Preferably, the solution will allow enforcement of an N-user license for content located on an open network like the Internet.

Petitioner Apple Inc. - Exhibit 1004, p. 5857

SUMMARY OF THE INVENTION

The above needs are met by a method and system for electronic commerce that uses special scrip - called "license scrip" - to provide temporary licenses to consumers accessing content. Scrip is primarily used as a form of electronic currency, however it can be more generally considered as a one-time token representing a general value. When scrip is used as an electronic currency, its value is monetary. When scrip is used as a temporary license, its value is the permission to access specific content. This permission may be unlimited or it may be for only a relatively brief period of time, say a few minutes to a few hours.

Accessing content with license scrip is very much like buying regular content with monetary scrip. Instead of having a price specified in monetary terms. Each page of content has a price (which may be zero) given in terms of license scrip. A consumer obtains license scrip from the vendor, preferably exchanging regular vendor scrip for the license scrip.

The vendor uses the license scrip to enforce an N-user license agreement - granting up to N people simultaneous access to the content. The vendor tracks the number and identity of consumers currently having licenses to access the content (i.e., consumers currently possessing valid license scrip).

A consumer initially lacks the license scrip needed to access the content. Upon receiving an access request from the consumer, the vendor determines whether a license is available. If a license is not available, the vendor tells the consumer to try again later and, optionally, provides the consumer with an estimate of when a license will be available.

If a license is available, then the vendor directs the consumer to obtain license scrip. Normally, the consumer obtains license scrip by requesting it from the vendor, but the consumer may get the license by any acceptable means. After receiving a license scrip

Petitioner Apple Inc. - Exhibit 1004, p. 5858

request. the vendor verifies that the consumer belongs to a class entitled to have a license. For example, if licenses are available to residents of only a certain state, the vendor ensures that the consumer resides in the state before granting the consumer a license.

If a license is available, then the vendor provides the consumer with the license scrip and remembers the granted license. The license scrip is preferably set to expire after a brief time period, but the duration of the license may vary depending upon business or legal concerns. To access content covered by the license, the consumer provides the license scrip when requesting content from the vendor. Each time the consumer accesses the content, the vendor returns replacement license scrip having the same or a later expiration time. Accordingly, the consumer can access the content as long as their license remains valid. When the consumer has not accessed the content for a while, the license scrip expires and the consumer can no longer access the content without obtaining new license scrip.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a top-level block diagram illustrating a computerized system for conducting electronic commerce;

FIGURE 2 is a block diagram illustrating a computer system used in the system of FIG. 1;

FIGURE 3 is a flow diagram illustrating the operations of the system of FIG. 1;

FIGURE 4 is a block diagram illustrating the data fields of a piece of scrip used in the system of FIG. 1;

FIGURE 5 is a diagram illustrating transactions between a consumer and a vendor utilizing license scrip to enforce an N-user license agreement according to the present invention; and

Petitioner Apple Inc. - Exhibit 1004, p. 5859

FIGURE 6 is a flow chart illustrating steps for determining whether to grant a license to a consumer.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A preferred embodiment of the present invention restricts access to electronic content through the use of an electronic commerce system. Accordingly, it is useful to describe the electronic commerce system before detailing how the system is utilized according to the present invention.

FIG. 1 shows a computerized system 100 for conducting electronic commerce. The system 100 includes a broker system 110, a vendor system 120, and a consumer system 130 interconnected by a communications network 140.

For clarity, the system 100 depicted in FIG. 1 shows only single broker, vendor, and consumer systems. In actual practice, any number of broker, vendor, and consumer systems can be interconnected by the network 140. The network 140 can be public or private, such as, for example, the Internet, an organization's intranet, a switched telephone system, a satellite linked network, or another form of network. The broker 111 using the broker system 110 can be a bank, a credit provider, an Internet service provider, a telephone company, or any institution the consumer trusts to sell electronic currency called "scrip."

The vendor system 120 is operated by a vendor 121. The vendor 121 provides products and/or content 150 of any type to consumers and, in one embodiment, provides content which is available by subscription. Each subscription page (i.e., page of data that is available for "purchase") has a price of zero but requires a special type of scrip, called "subscription scrip," before it can be accessed. Since the price of a page is zero, the consumer 131 can "purchase" an unlimited number of pages once the consumer 131 has the

Petitioner Apple Inc. - Exhibit 1004, p. 5860

proper subscription scrip 330. The subscription expires when the subscription scrip 330 expires.

A consumer 131 can use the consumer computer system 130 to electronically acquire the products or content 150 of the vendor 121. As used herein, "consumer" refers to an organization such as a library or corporation, a member of the organization, such as a librarian or an employee, or an individual, such as a person visiting a library or a home computer user. Of course, actions attributed to the organization are usually performed by a member of the organization.

A computer system 200 suitable for use as the broker, vendor, and consumer systems is shown in FIG. 2. The computer system 200 includes a central processing unit (CPU) 210, a memory 220, and an input/output interface 230 connected to each other by a communications bus 240. The CPU 210, at the direction of users 250, e.g. brokers, vendors, and/or consumers, executes software programs, or modules, for manipulating data. The programs and data can be stored in the memory 220 as a database (DB) 221. The DB 221 storing programs and data on the consumer computer system 130 is referred to as a "wallet." In a preferred embodiment of the present invention described herein, many of the operations attributed to the consumer are, in fact, performed automatically by the wallet 221.

The memory 220 can include volatile semiconductor memory as well as persistent storage media, such as disks. The I/O interface 230 is for communicating data with the network 140, the users 250, and other computer system peripheral equipment, such as printers, tapes, etc.

The computer system 200 is scaled in size to function as the broker, vendor, or consumer systems. For example, when scaled as the consumer computer system 130, the computer system 200 can be a small personal computer (PC), fixed or portable. The

Petitioner Apple Inc. - Exhibit 1004, p. 5861

configurations of the computer system 200 suitable for use by the broker 111 and the vendor 121 may include multiple processors and large database equipped with "fail-safe" features. The fail-safe features ensure that the database 221 is securely maintained for long periods of time.

FIG. 3 shows an operation of the electronic commerce system 100. The consumer 131 uses currency to purchase electronic broker scrip 320 generated by the broker 111. Here, purchasing means that upon a validation of the authenticity of the consumer 131 and the consumer's currency 310, the broker system 110 generates signals, in the form of data records. The signals are communicated, via the network 140, to the consumer system 130 for storage in the wallet 221 of the memory 220 of the consumer system 130.

The scrip is stamped by the generator of the scrip to carry information that is verifiable by the originator, and any other system that has an explicit agreement with the originator. In addition, each scrip is uniquely identifiable and valid at only a single recipient. After a single use, the recipient of the scrip can invalidate it, meaning that the signals of the data record are no longer accepted for processing by the recipient computer system.

In one embodiment, the consumer 131 exchanges the broker scrip 320 with the broker 111 for vendor scrip 330. To complete this transaction, the broker system 110 executes licensed software programs which generate scrip 330 for consumers as needed. Alternatively, the broker 111, in a similar transaction 303, exchanges currency 310 for bulk vendor scrip 330 which is then sold to consumers.

In another embodiment, the consumer 131 exchanges currency with the vendor 121 for regular vendor. In this latter embodiment, there is no need for a broker 111. In addition, the vendor scrip may be free, meaning that the consumer 131 does not need to exchange currency for the scrip.

Petitioner Apple Inc. - Exhibit 1004, p. 5862

The consumer 131, in a transaction 304, provides the scrip 330 to the vendor 121. The vendor 121 checks the stamp of the scrip 330 to verify its authenticity, and also checks to make sure the value of the scrip covers the requested content and has not expired. Approval of the transaction results in the delivery of the desired content 150 to the consumer 131. The vendor 121 can also return 304 modified scrip 330 to the consumer 131 as change.

FIG. 4 is a block diagram illustrating the data fields of a single piece of scrip 400. The scrip 400 is logically separated into seven data fields. The Vendor field 410 identifies the vendor for the scrip 400. The Value field 412 gives the value of the scrip 400. The scrip ID field 414 is the unique identifier of the scrip. The Customer ID field 416 is used by the broker 111 and vendor 121 to verify that the consumer has the right to spend the scrip. The Expires field 418 gives the expiration time for the scrip 400. The Props field 420 holds consumer properties, such as the consumer's age, state of residence, employer, etc. Finally, the Stamp field 422 holds a digital stamp and is used to detect tampering with the scrip 400.

The present invention uses "license" scrip, which can be thought of as special purpose scrip having a short period of validity. A consumer with license scrip has a license to view the content covered by the license until the scrip expires.

FIG. 5 is a diagram illustrating transactions between a consumer 510 and a vendor 512 utilizing license scrip to enforce an N-user license agreement according to the present invention. In the transactions of FIG. 5, the vendor 512, for example, can be a library located at a state university. Assume the library purchases a four user license for a CD-ROM and makes the CD-ROM available to other terminals in the library via a local area network and residents of the state via the Internet. To conform with the license, the library must ensure that no more than four consumers are simultaneously accessing the CD-ROM. In this

Petitioner Apple Inc. - Exhibit 1004, p. 5863

example, the library is the vendor 512 and the people who can access the CD-ROM, either in the library or elsewhere, are the consumers 510.

In another example, a newspaper publisher operates a web site. Assume that a corporation purchases a 20 user license allowing up to 20 people from the corporation to simultaneously access content on the web site. To police its license, the publisher tracks the users of its web site and block users who are not licensed or who have exceeded the scope of the applicable license. Accordingly, the newspaper publisher is the vendor 512 and the corporation and its employees are the consumers 510.

Although neither the illustrated transactions nor the above examples directly utilize a broker, there may be circumstances where it is desirable to use a broker 111 to perform one or more of the transactions described below. Those of ordinary skill in the art will understand that certain transactions attributed to the consumer or the vendor can be performed instead by a broker 111. For example, the library and/or newspaper may issue vendor and license scrip directly or rely on a third-party broker for this task.

Turning to FIG. 5, the consumer 510 initially requests 520 content from the vendor 512 without valid license scrip. In response, the vendor 512 checks to determine whether there is an available license (i.e., whether an additional consumer is allowed to view the content under the license). Preferably, the vendor 512 maintains a data structure associated with the licensed content that can be quickly scanned to determine whether a license is available. In one embodiment, this data structure is a simple N-entry array, with each entry holding fields for the expiration time and Customer ID of the consumer 510 having the license. As licenses are granted, the vendor 512 fills in the array until no more entries are available.

Petitioner Apple Inc. - Exhibit 1004, p. 5864

If no licenses are available, then the vendor 512 instructs 522 the consumer 510 to try again later. In one embodiment, the vendor 512 scans the data structure to determine when the first license may become available and provides the consumer 510 with that time as a suggestion of when to try to access the content again. If a license is available, then the vendor 512 instructs the consumer 510 to go and obtain license scrip.

In response, the consumer 510 attempts 524 to obtain license scrip from the vendor 512. The vendor 512 determines whether the consumer 510 is entitled to a license (i.e., entitled to view the content). FIG. 6 is a flow chart 600 illustrating steps for determining whether to grant license scrip to the consumer 510. When the vendor 512 receives the request from the consumer 510, the vender retrieves 610 information about the consumer. The vendor 514 may retrieve this information by asking the consumer 510 to provide it, from the scrip used to request the license scrip, from a "cookie" on the consumer's computer system, or from a table of information shared by the vendor 512 and the consumer 510 or a broker 111. Additionally, the wallet 221 on the consumer's computer system 130 may be configured to automatically provide information about the consumer 510 when requested by a vendor 512. Depending on the needs of the vendor 512 and the license agreement for the content, the information that may be gathered in this manner includes whether the consumer 510 is a member of an organization, the state of residence of the consumer, the consumer's age, or any other information that is relevant to determining whether to provide access to the consumer 510.

The vendor 512 uses this information to determine 612 whether the consumer belongs to a class that has access to the content held by the vendor 512. If the consumer does not belong to a class having access, for example, if the consumer is not a state resident, then the

Petitioner Apple Inc. - Exhibit 1004, p. 5865

vendor denies 614 access to the consumer 510. Preferably, the vendor 512 directs the consumer 510 to a web page explaining why access was denied.

If the consumer 510 belongs to a class having access, the vendor 512 scans the data structure identifying the current licensees of the content and determines 616 whether an additional license is available. Since there may be a delay between the time the consumer 510 is told to buy license scrip and when the wallet 221 tries to buy the scrip, it is possible that the available license may have been acquired by another consumer during that time. If no licenses are available, then the consumer 510 is told to try again later and optionally given a time when a license may be available.

If a license is available, then the vendor 512 grants 618 the license to the consumer 510. The vendor 512 provides 526 the consumer with license scrip that allows the consumer 510 to access the content. The license scrip preferably has a relatively short validity period, say a few minutes to an hour, and allows the consumer 510 full access to the licensed material for the duration of the scrip. The choice of expiration time for the scrip is a business or legal decision. Since the intention of the license scrip is to hold onto one license slot while the consumer 510 is actively using the content, the duration of the license should cover the time that the consumer 510 is expected to be active. In another embodiment, the duration of the scrip is determined, at least in part, by the type of content accessed by the consumer 510. In addition, the vendor 512 preferably records data about the granted license, including the Customer ID of the consumer 510 and the expiration time of the license in the appropriate data structure.

Each time the consumer 510 wishes to access 528 content held by the vendor 512, the consumer provides the license scrip to the vendor. If the scrip is expired or otherwise invalid, then the consumer's request for access is treated as a request without scrip as illustrated by

Petitioner Apple Inc. - Exhibit 1004, p. 5866

transaction 520. If the scrip is valid, then the vendor 512 allows the consumer 510 to access the content. In addition, the vendor 512 provides 530 the consumer 510 with replacement license scrip having an updated expiration time. Typically, the updated expiration time is later than the old expiration time, although it can be the same or earlier. In one embodiment, the vendor 512 grants the consumer 510 less additional time each time the vendor issues new license scrip to ensure that the consumer's license eventually expires and other consumers may eventually access the content. The vendor 512 also updates its data structure to reflect the new expiration date of the consumer's license.

Periodically, the vendor 514 preferably scans the data structure to determine whether any licenses have expired. If so, the entry is purged from the data structure, thereby freeing up a license for another consumer 510. Accordingly, the present invention uses license scrip to enforce an N-user license agreement.

It should be understood that FIG. 5 illustrates only one possible set of transactions. FIG. 3. in combination with FIG. 5, provides insight into other possible transactions. For example, a corporation could purchase an N-user license agreement from a broker 111 to access content on a vendor's system 120. The broker 111 can verify that the corporation is entitled to a license and then issue the license scrip from a special scrip series corresponding to the number of users covered by the license. The vendor 121 knows from the scrip series to restrict access from consumers using that license scrip.

Having described a preferred embodiment of the invention, it will now become apparent to those skilled in the art that other embodiments incorporating its concepts may be provided. It is felt therefore, that this invention should not be limited to the disclosed invention, but should be limited only by the spirit and scope of the appended claims.

Petitioner Apple Inc. - Exhibit 1004, p. 5867

CLAIMS

We claim:

1. A method of restricting simultaneous access to content, comprising the steps of:

receiving a request to access the content from a consumer;
determining whether the consumer is entitled to access the content; and
responsive to a positive determination, providing the consumer with license scrip allowing access to the content.

2. The method of claim 1, wherein the request to access the content is accompanied by license scrip having an expiration time and wherein the providing step provides the consumer with additional license scrip having an updated expiration time.

3. The method of claim 1, wherein the license scrip has an expiration time and further comprising the steps of:

receiving a second request to access the content from the consumer, the second request including the license scrip; and
responsive to the second request, providing the consumer with replacement license scrip having an updated expiration time.

4. The method of claim 1, wherein the step of determining whether the consumer is entitled to access the content comprises the steps of:

determining whether the consumer belongs to a class having access to the content; and
determining whether a license to access the content is available.

5. The method of claim 4, wherein the step of determining whether the consumer belongs to a class having access to the content comprises the step of:

determining information about the consumer from scrip utilized to request access to the content.

Petitioner Apple Inc. - Exhibit 1004, p. 5868

6. The method of claim 4, wherein the step of determining whether a license to access the content is available comprises the steps of:

determining a number of consumers that have licenses to access the content; and

determining a number of allowed licenses;

wherein a license to access the content is available if the number of consumers that have licenses to access the content is less than the number of allowed licenses.

7. The method of claim 4, further comprising the step of:

responsive to a determination that no licenses to access the content are available, providing the consumer with an estimate of when a license will be available.

8. A computer program product having computer-readable instructions embodied thereon for restricting access to content stored on a computer system, the computer-readable instructions comprising instructions for:

receiving a request to access the content stored on the computer system, the request accompanied by scrip;

determining whether the scrip authorizes access to the content;

responsive to a determination that the scrip does not authorize access to the content, determining whether scrip authorizing access to the content is available; and

responsive to a determination that scrip authorizing access to the content is available, providing the scrip.

9. The computer program product of claim 8, further comprising instructions for:

responsive to a determination that the scrip authorizes access to the content, providing replacement scrip having an updated expiration time.

Petitioner Apple Inc. - Exhibit 1004, p. 5869

10. The computer program product of claim 8, wherein the instructions for determining whether the scrip authorizes access to the content further comprise computer instructions for:

determining a type of the scrip accompanying the request; and

responsive to a determination that accompanying scrip is license scrip, determining whether the license scrip has expired, wherein unexpired license scrip authorizes access to the content.

11. The computer program product of claim 8, wherein the instructions for determining whether scrip authorizing access to the content is available comprise instructions for:

determining a maximum number of requesters that can be authorized to access the content;

determining whether a current number of requesters authorized to access the content is less than the maximum number of requesters; and

responsive to a determination that the current number of requesters authorized to access the content is less than the maximum number of requesters, determining that scrip authorizing access to the content is available.

12. The computer program product of claim 8, further comprising instructions for:

responsive to a determination that scrip authorizing access to the content is not available, calculating an estimate of when the scrip authorizing access will be available.

Petitioner Apple Inc. - Exhibit 1004, p. 5870

13. A computer system for limiting a number of users that can access content stored on a server associated with the computer system, the computer system comprising:

a module for receiving a request from a user to access the content stored on the server;

a module for determining the number of users currently having rights to access the content; and

a module for providing the user with license scrip if the number of users currently having rights to access the content is less than a number of users allowed to access the content, the license scrip granting the user the right to access the content.

14. The system of claim 13, wherein the module for determining the number of users currently having access rights to content comprises:

a module for scanning a data structure stored in a memory of the computer system, the data structure having one or more entries indicating the number of users having access rights to the content.

15. The computer system of claim 14, wherein the data structure indicates when users' rights to access the content expire, further comprising:

a module for purging the entries of users whose right to access the content has expired.

16. The system of claim 13, wherein only a privileged class can access the content. further comprising:

a module for determining whether the user is a member of the privileged class.

17. The system of claim 13, wherein the license scrip grants the user the right to access the content until an expiration time.

18. The system of claim 17, further comprising:

a module for receiving a second request from the user to access the content stored on the server accompanied by the license scrip; and

Petitioner Apple Inc. - Exhibit 1004, p. 5871

a module for providing the user with replacement license scrip having a later expiration time.

Petitioner Apple Inc. - Exhibit 1004, p. 5872

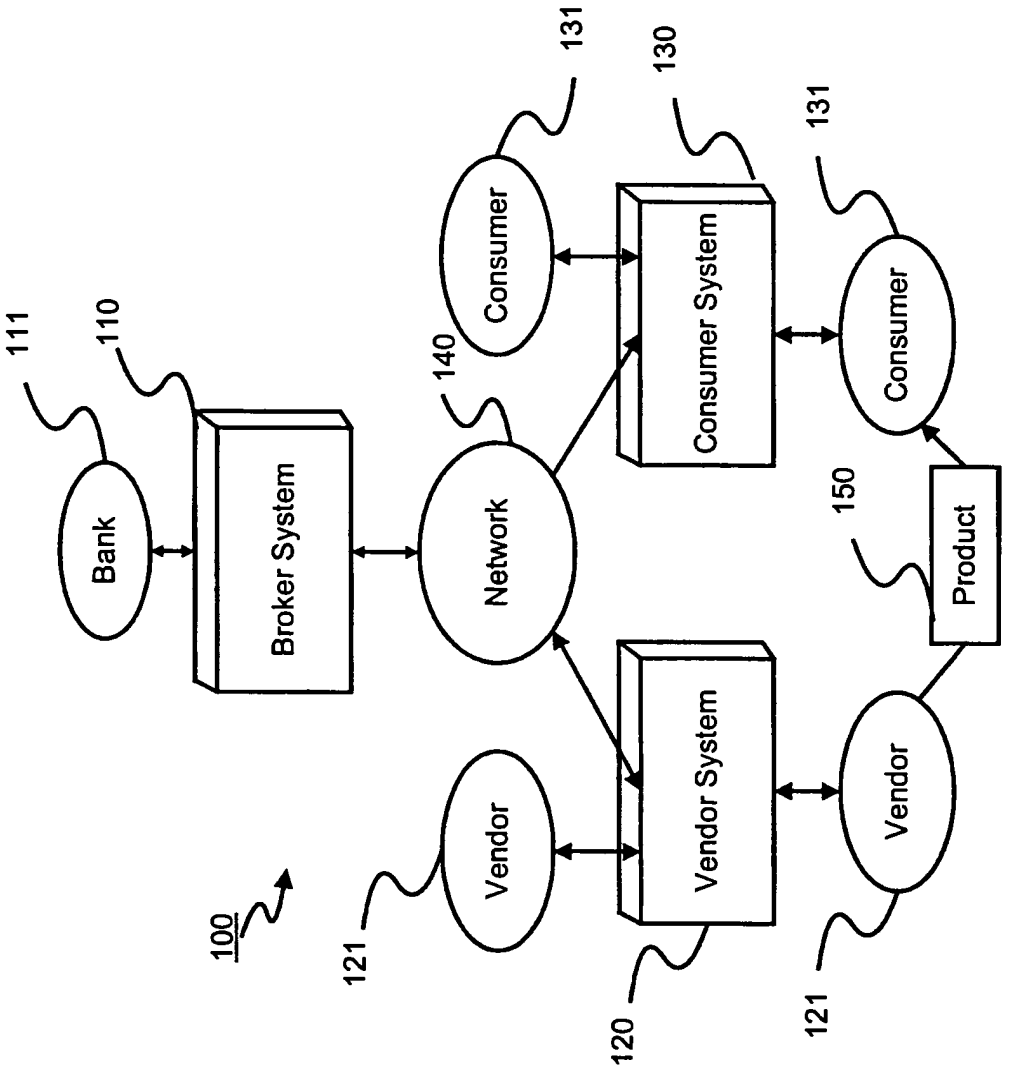


FIG. 1

Petitioner Apple Inc. - Exhibit 1004, p. 5873

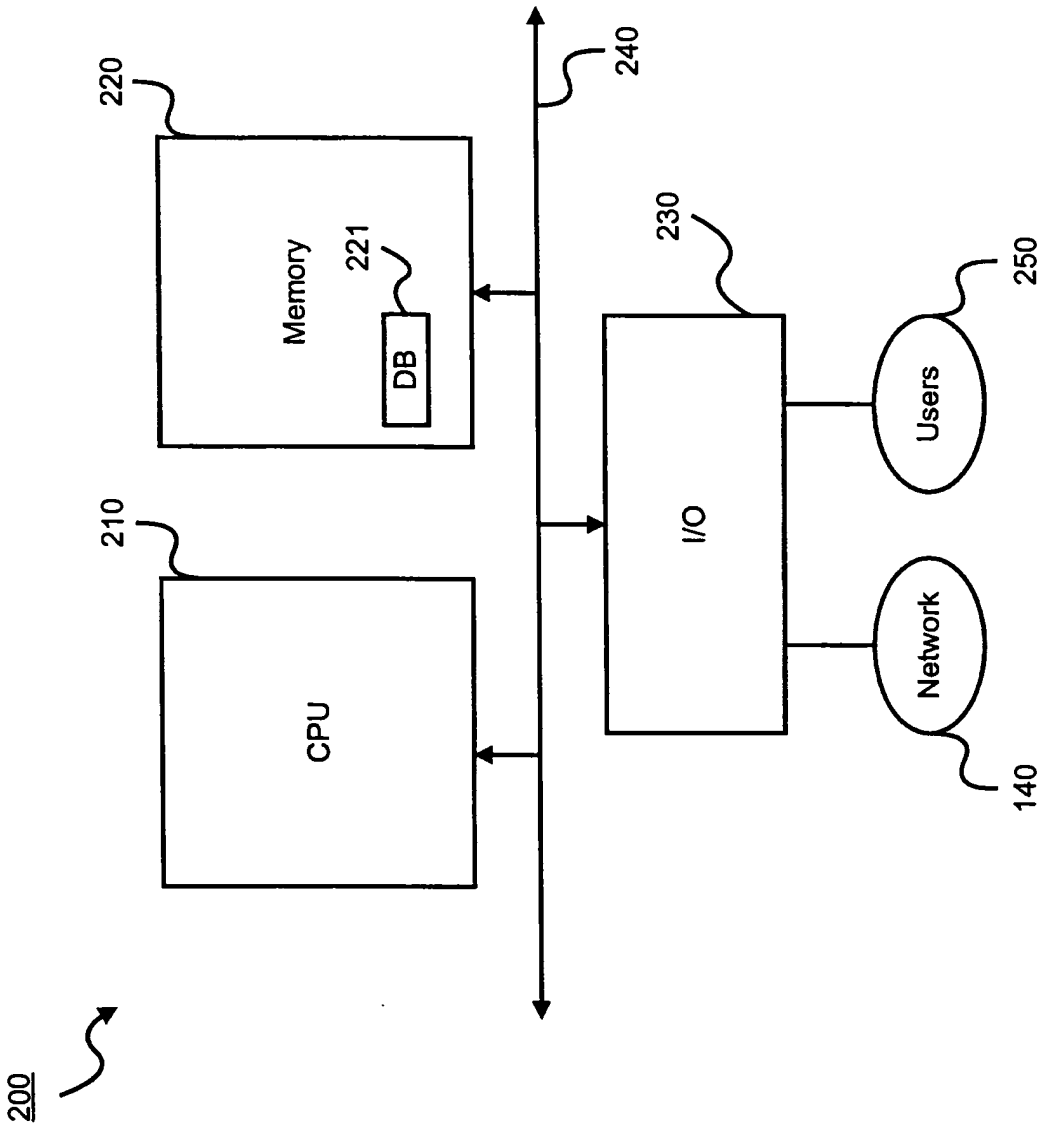


FIG. 2

Petitioner Apple Inc. - Exhibit 1004, p. 5874

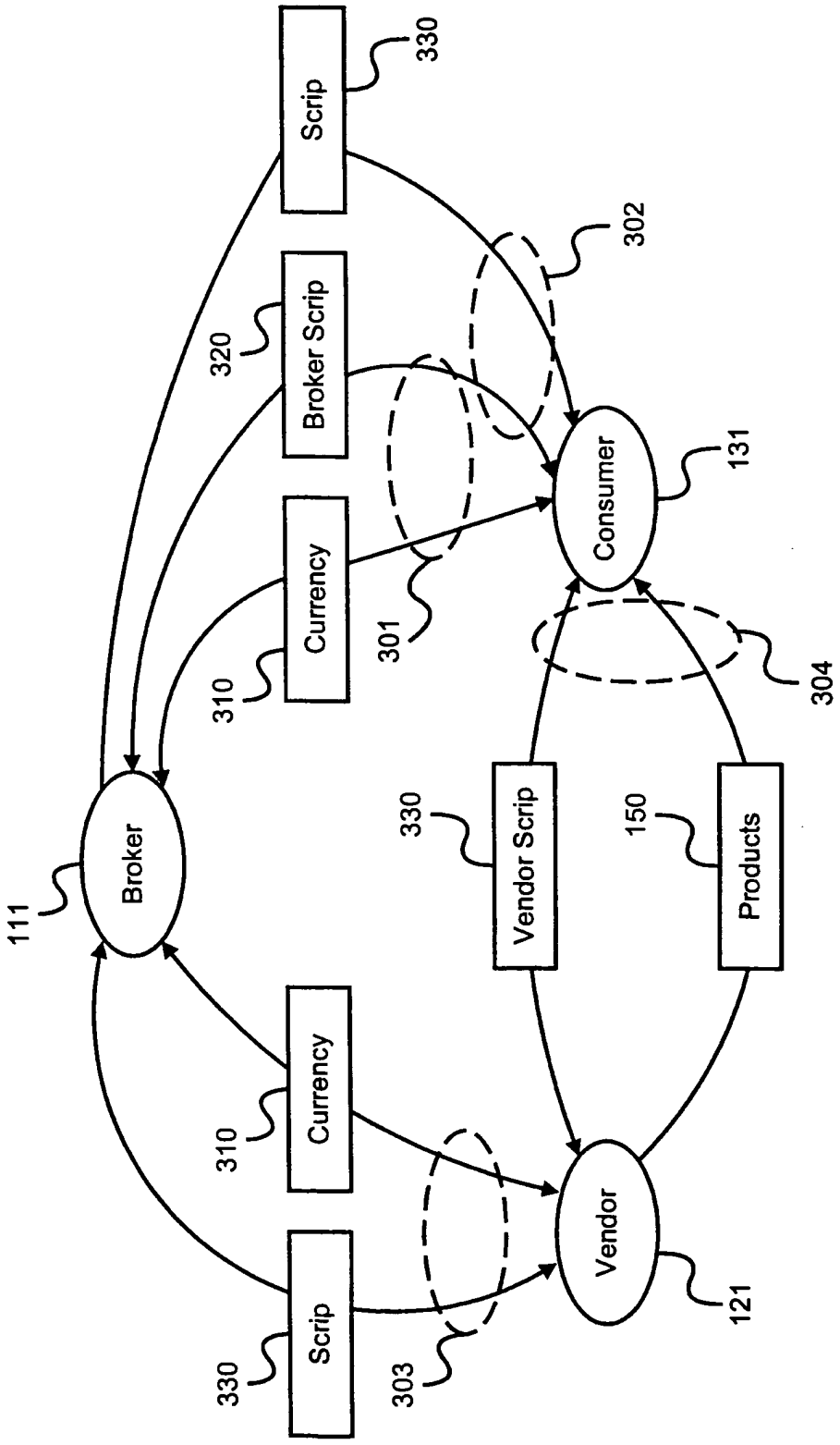

Scrip
330
Broker Scrip
320
Currency
310
Broker
111
Currency
310
Scrip
330
302
301
303
Consumer
131
Vendor Scrip
330
Products
150
304
Vendor
121

FIG. 3

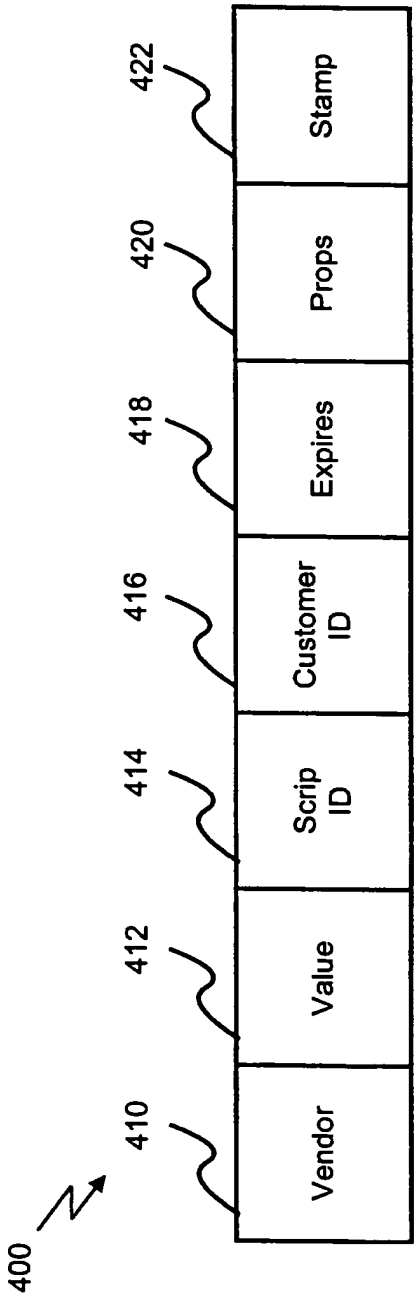


FIG. 4

Petitioner Apple Inc. - Exhibit 1004, p. 5876

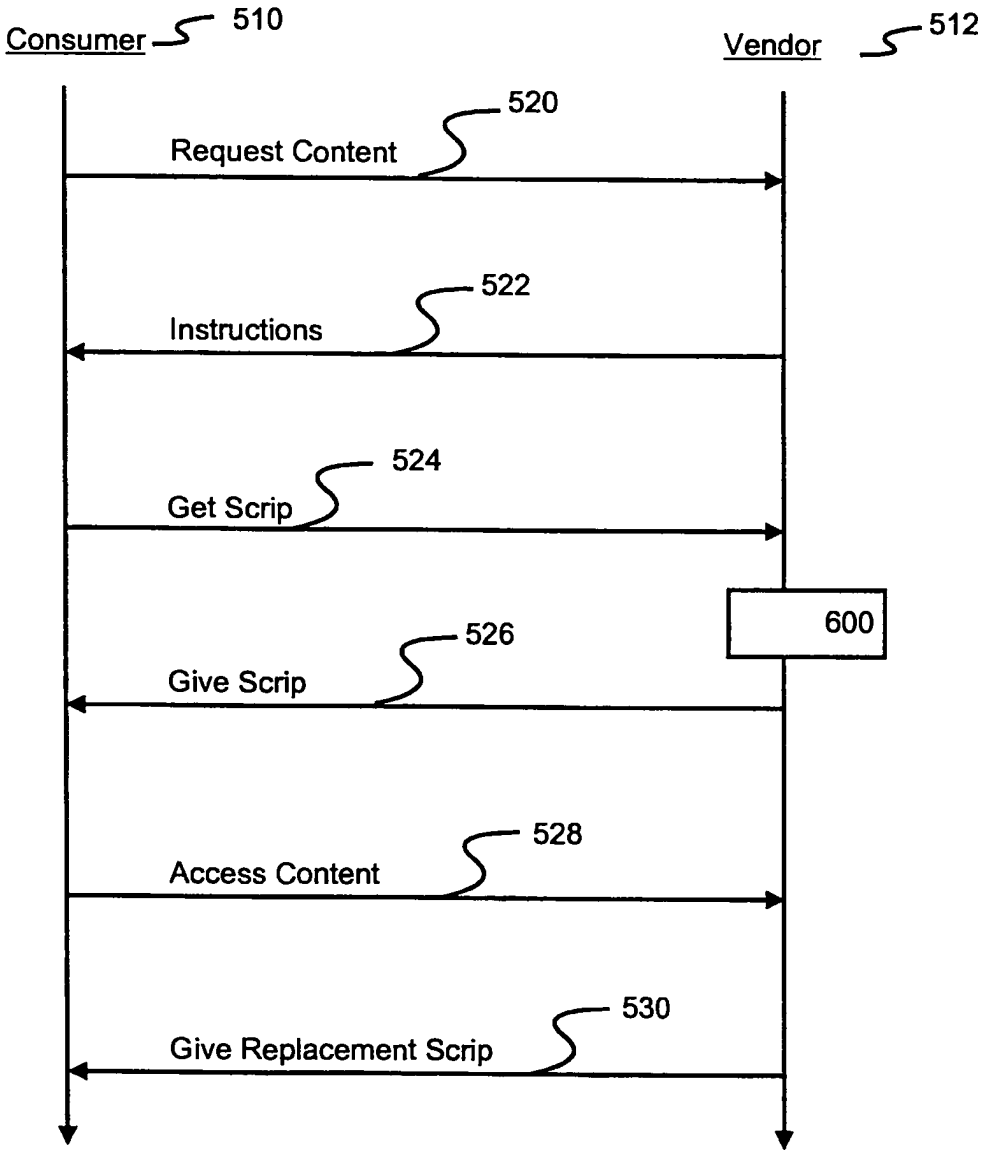
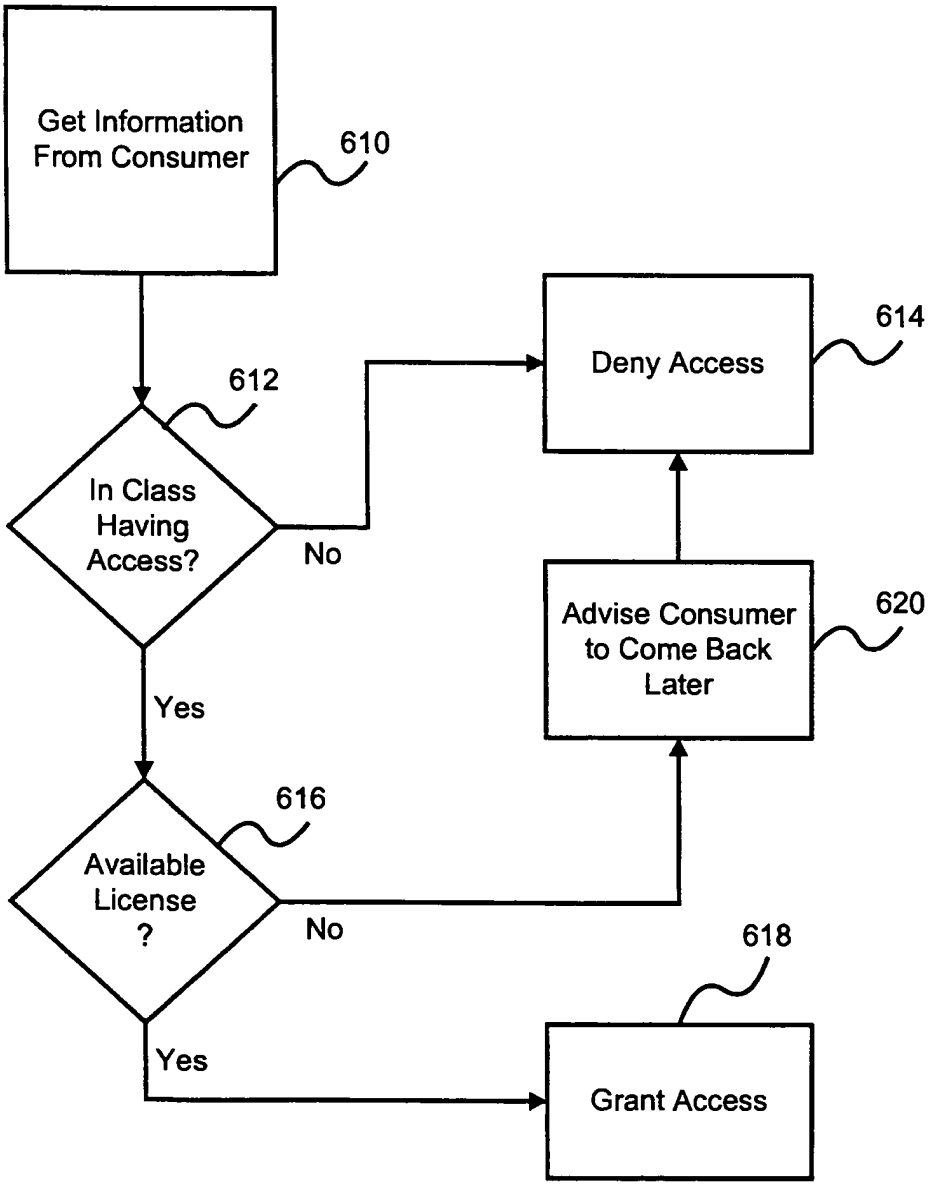


FIG. 5

Petitioner Apple Inc. - Exhibit 1004, p. 5877



Petitioner Apple Inc. - Exhibit 1004, p. 5878

INTERNATIONAL SEARCH REPORT

Interna al Application No
PCT/US 00/10213

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F G07F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 98 58306 A (OYLER SCOTT ;GUTHRIE JOHN (US); TECHWAVE INC (US); KRISHNAN GANAPA) 23 December 1998 (1998-12-23)	1,4-6,8, 10,11, 13,14, 16,17
Y		18
A		2,3,7,9, 12,15
	abstract page 6, line 4 -page 8, line 10 page 10, line 8 -page 16, line 17 page 28, line 7 -page 30, line 23 page 39, line 2 - line 11 figures 1-4	
	-/—	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
2 August 2000	09/08/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Jacobs, P

2

Form PCT/ISA/210 (second sheet) (July 1992)

Petitioner Apple Inc. - Exhibit 1004, p. 5879

INTERNATIONAL SEARCH REPORT

Intern nal Application No
PCT/US 00/10213

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 97 03423 A (DIGITAL EQUIPMENT CORP) 30 January 1997 (1997-01-30)	1-4,8-10
Y		18
A		5,13,16, 17
	page 4, line 15 -page 11, line 33 figures 1-5	
X	US 5 905 860 A (BRINGHURST ADAM L ET AL) 18 May 1999 (1999-05-18)	1,4-6, 13,14, 16,17
A		2,3, 7-12,15, 18
	abstract column 2, line 40 -column 16, line 3	
X	GB 2 316 503 A (ICL PERSONAL SYSTEMS OY) 25 February 1998 (1998-02-25)	1,4-6, 13,14, 16,17
A		2,3, 7-12,15, 18
	abstract page 6 -page 21 figures 1-4 claim 1	

2

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

Petitioner Apple Inc. - Exhibit 1004, p. 5880

INTERNATIONAL SEARCH REPORT

...rmation on patent family members

Intern... ...al Application No
PCT/US 00/10213

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9858306 A	23-12-1998	US 6073124 A	06-06-2000
		AU 8150598 A	04-01-1999
WO 9703423 A	30-01-1997	US 5802497 A	01-09-1998
		BR 9606450 A	30-09-1997
		EP 0796480 A	24-09-1997
		IL 117195 A	20-06-1999
		JP 2984731 B	29-11-1999
		JP 9510814 T	28-10-1997
US 5905860 A	18-05-1999	US 5758069 A	26-05-1998
GB 2316503 A	25-02-1998	NONE	

Petitioner Apple Inc. - Exhibit 1004, p. 5881

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 1/00 — A1

(11) International Publication Number: **WO 99/60461**

(43) International Publication Date: 25 November 1999 (25.11.99)

(21) International Application Number: PCT/GB98/03828

(22) International Filing Date: 18 December 1998 (18.12.98)

(30) Priority Data:
09/080,030 15 May 1998 (15.05.98) US

(71) Applicant: INTERNATIONAL BUSINESS MACHINES CORPORATION [US/US]; New Orchard Road, Armonk, NY 10504 (US).

(71) Applicant *(for MC only)*: IBM UNITED KINGDOM LIMITED [GB/GB]; North Harbour, Portsmouth, P.O. Box 41, Hampshire PO6 3AU (GB).

(72) Inventors: BERSTIS, Viktors; 5194 Cuesta Verde, Austin, TX 78746 (US). HIMMEL, Maria, Azua; 6403 Rain Creek Parkway, Austin, TX 78759 (US).

(74) Agent: BOYCE, Conor; IBM United Kingdom Limited, Intellectual Property Law, Hursley Park, Winchester, Hampshire SO21 2JN (GB).

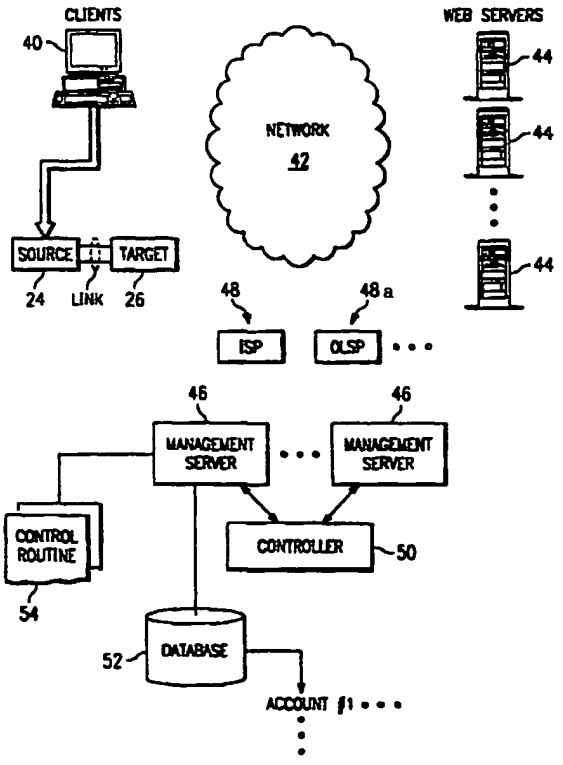
(81) Designated States: CN, CZ, IL, IN, JP, KR, PL, SG, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: ROYALTY COLLECTION METHOD AND SYSTEM FOR USE OF COPYRIGHTED DIGITAL MATERIALS ON THE INTERNET

(57) Abstract

A method, system and computer program product to facilitate royalty collection with respect to online distribution of electronically published material over a computer network. In one embodiment, a method for managing use of a digital file (that includes content subject to copyright protection on behalf of some content provider) begins by establishing a count of a number of permitted copies of the digital file. In response to a given protocol, a copy of the digital file is then selectively transferred from a source to a target. Thus, for example, the source and target may be located on the same computer with the source being a disk storage device and the target being a rendering device (e.g., a printer, a display, a sound card or the like). The method logs an indication each time the digital file is transferred from the source to a target rendering device, and the count is decremented upon each transfer. When the count reaches a given value (e.g., zero), the file is destroyed or otherwise prevented from being transferred from the source device. The indications logged are transferred to a management server to facilitate payment of royalties to the content provider.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

ROYALTY COLLECTION METHOD AND SYSTEM FOR USE OF COPYRIGHTED DIGITAL MATERIALS ON THE INTERNET

BACKGROUND OF THE INVENTION

Technical Field

The present invention relates generally to managing collection of royalties for electronically-published material distributed over a computer network.

Description of the Related Art

The World Wide Web is the Internet's multimedia information retrieval system. In the Web environment, client machines effect transactions to Web servers using the Hypertext Transfer Protocol (HTTP), which is a known application protocol providing users access to files (e.g., text, graphics, images, sound, video, etc.) using a standard page description language known as Hypertext Markup Language (HTML). HTML provides basic document formatting and allows the developer to specify "links" to other servers and files. In the Internet paradigm, a network path to a server is identified by a so-called Uniform Resource Locator (URL) having a special syntax for defining a network connection. Use of an HTML-compatible browser (e.g., Netscape Navigator or Microsoft Internet Explorer) at a client machine involves specification of a link via the URL. In response, the client makes a request to the server (sometimes referred to as a "Web site") identified in the link and, in return, receives in return a document or other object formatted according to HTML.

One of the technical advantages of the World Wide Web is the ease with which digital content (e.g., graphics, sound, video, movies and the like) may be transmitted and distributed to many users. Indeed, copying a digital file is as easy as clicking on a computer mouse. Copyright laws afford a copyright owner the exclusive right to reproduce the copyrighted work in copies, to distribute such copies, and to publicly perform and display the work. Each time a digital file is transferred over the Internet and copied onto a user s memory, the copyright owner's exclusive reproduction right is implicated (and possibly violated). Likewise, transmission of the copyrighted work over the physical wire is tantamount to a distribution. Indeed, in an open system (e.g., a personal computer accessing the World Wide Web through an Internet Service Provider (ISP)), copies of copyrighted materials can undergo unlimited further copying and transmission without the ability of the owner to collect appropriate compensation (e.g., royalties).

Many publishers or other content providers naturally are hesitant to make their copyrighted works available over the Internet due to the ease with which these materials may be copied and widely disseminated without adequate compensation. Presently, Internet commerce remains highly unregulated, and there is no central authority for managing collection and allocation of content provider royalties. Moreover, while publishers and content rights societies and organizations are attempting to address the legal and logistical issues, the art has yet to develop viable technical solutions.

One technique that has been proposed involves wrapping a copyrighted work in a copy protection "environment" to facilitate charging users for use of that information obtained from the Internet or World Wide Web. This approach, called COPINET, links a copyright protection mechanism with a copyright management system, and it is described in *Charging, paying and copyright - information access in open networks*, Bennett et al., 19th International Online Information Meeting Proceedings, Online Information 1995 pp. 13-23 (Learned Information Europe Ltd.). Publishers in such a system can determine an appropriate level of protection while monitoring use and managing the chain of rights. This approach is also said to provide protection for digital material even after delivery to the user workstation. In particular, copyright material is "wrapped" (by encryption) and "unwrapped" as a result of a specific authorization provided by a trusted subsystem. Material thus is only "visible" to the environment and thus any subsequent user actions, such as "save" or "copy", result in the protected material, or material derived from it, remaining in a protected state when outside the environment.

Although the above-described approach provides some advantages, it does not address the problem of managing the collection of royalties and/or the allocating of such payments to content providers. Moreover, it is not an accepting solution in the context of an open PC architecture such as implemented in the public Internet. It also requires the use of a separate trusted subsystem to generate the authorizations for particular content transfers, which is undesirable.

Other known techniques for managing use of content over the Internet typically involve electronic "wallets" or smart cards. Known prior art systems of this type are illustrated, for example, in U.S. Patent Nos. 5,590,197 and 5,613,001. These systems involve complex hardware and encryption schemes, which are expensive and difficult to implement in practice. They are not readily adaptable to provide general royalty payment schemes for Internet content usage.

Thus, there remains a need to provide improved methods and systems for collecting royalties on the Internet as a result of use of copyrighted content.

The present invention solves this important problem.

SUMMARY OF THE INVENTION

An object of this invention is to enable a pair of "certified" devices (e.g., a storage device and a rendering device) to operate within the context of a given security protocol and thereby manage copies of a digital file and associated copy control information.

Still another object of this invention is to enable a copyright proprietor to maintain a degree of control over copyrighted content even after that content has been fetched from a server and downloaded to a client machine, e.g., in a Web client-server environment.

A particular object of the present invention is to manage the number of copies of a digital file that may be made within a Web appliance having a secure disk storage and that is connectable to the Internet using a dialup network connection.

A still further object of this invention is to restrict a number of copies of a digital file that may be made at a given Web client machine connected to the World Wide Web.

It is yet another object of this invention to enable a publisher of an electronic document to control the number of copies of such document that may be made on the Internet by permitted users.

It is a more general object of this invention to manage permissible use of copyrighted content on the Internet and World Wide Web.

It is still another more general object of this invention to manage collection of information to facilitate payment of appropriate compensation to content providers and publishers arising from use of their copyrighted content on the Internet.

Another object of this invention is to manage the charging of users for information obtained from the Internet or World Wide Web.

A still further object of this invention is to facilitate royalty collection as a result of electronically published material distributed

online over a computer network (e.g., the public Internet, an intranet, an extranet or other network).

One embodiment of the invention is a method for managing copies of a digital file, which includes content subject to copyright protection, on behalf of some content provider (e.g., an author, publisher or other). It is assumed that a given usage scheme has been established with respect to the file as defined in copy control information associated with the file. Thus, for example, the copy control information may define a set of payment options including, without limitation, prepayment (for "n" copies), pay-per-copy (as each copy is made), IOU (for copies made offline), or some other payment option. The copy control information may also include other data defining how the file is managed by the scheme including: a count of the number of permitted copies, a count of the number of permitted pay-per-copy versions, copyright management information, payee information, an expiration date (after which copying is no longer permitted), and the like.

The present invention assumes the existence of a pair of devices, a "source" and a "target", that have been or are certified to use the scheme. Typically, the "source" is a storage device while the "target" is a rendering device. An illustrative storage device may be disk storage, system memory, or the like. An illustrative rendering device may be a printer, a display, a sound card or the like. The source and target devices may both be storage devices (e.g., a Web server and a client disk storage). In either case, each of the devices comprising the pair is "certified" (typically upon manufacture) to operate under a given security protocol. Under the protocol, the devices include appropriate circuitry and/or software, as the case may be, to facilitate the establishment of a secure link between the storage and rendering devices. Each device requires the other to validate itself and thus prove that the device can be trusted to manage the content (namely, the digital file) sought to be protected.

When the technique is implemented in an "open" client-server environment, hardware devices (e.g., microcontrollers) preferably are used in the storage and rendering devices to facilitate generation of the secure link. When the technique is implemented in a "closed" Web appliance environment, the secure link may be established and managed using software resident in the control routines associated with the storage and rendering devices. The secure link may be established and managed in software under such conditions because, in the Web appliance environment, it is possible to readily disable the secure link in the event of tampering with the appliance housing or other circuitry. Regardless of the environment, the secure link is first established

between the "certified" storage and rendering devices. Thereafter, the digital file, together with at least part of its copy control information, is transferable between the storage and rendering devices in accordance with the particular usage and payment scheme being utilized. Thus, for example, if a prepayment scheme is implemented and an expiration date (associated therewith) has not occurred, a given number of copies of the file may be transferred between the storage and rendering devices. The prepayment funds are collected at a central location and then redistributed to the copyright proprietor or some third party.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a representative system in which the present invention is implemented;

Figure 2 is a simplified block diagram of a source device and a target device connected by a channel over which a digital file is transferred according to the present invention;

Figure 3 is an illustrative example of a source device connected to a set of target rendering devices in a client computer;

Figure 4 is a block diagram of a representative copyright management system according to the present invention;

Figure 5 is a flowchart of a preferred method of managing a digital file according to the present invention;

Figure 6A is pictorial representation of a data processing system unit connected to a conventional television set to form a "Web" appliance;

Figure 6B is a pictorial representation of a front panel of the data processing system unit;

Figure 6C is a pictorial representation of a rear panel of the data processing system unit;

Figure 6D is a pictorial representation of a remote control unit associated with the data processing system unit; and

Figure 7 is a block diagram of the major components of the data processing system unit.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A representative system in which the present invention is implemented is illustrated in **Figure 1**. A plurality of Internet client machines **10** are connectable to a computer network Internet Service Provider (ISP) **12** via a "resource" such as a dialup telephone network **14**. As is well known, the a dialup telephone network usually has a given, limited number of connections **16a-16n**. ISP **12** interfaces the client machines **10** to the remainder of the network **18**, which includes a plurality of Internet server machines **20**. A client machine typically includes a suite of known Internet tools (e.g., Web browser **13**) to access the servers of the network and thus obtain certain services. These services include one-to-one messaging (e-mail), one-to-many messaging (bulletin board), on-line chat, file transfer and browsing. Various known Internet protocols are used for these services. Thus, for example, browsing is effected using the Hypertext Transfer Protocol (HTTP), which provides users access to multimedia files using Hypertext Markup Language (HTML). The collection of servers that use HTTP comprise the World Wide Web, which is the Internet's multimedia information retrieval system.

As will be described in more detail below, the present invention may be implemented in hardware and/or in software. The software implementation is particularly useful when the client machine is an Internet or Web appliance, such as illustrated in **Figures 6A-6D**. In the case of the software implementation, a client machine has associated therewith a software routine **15** designed to perform one or more of the functions of the digital file copy protection method, as will be described. The software is preferably a client application (although it may be implemented with the browser as a plug-in, or with a client-side proxy, or as a standalone application). Alternatively, the agent is built into the browser, or it is implemented as a Java applet or standalone application. Thus, as used herein, in this particular embodiment, the software **15** is any application running on a client machine **10** that performs the copy protection/royalty management task(s) on behalf of the user(s) of that client according to the present invention.

The discussion which follows primarily uses the words "copying" or "copies" to describe the control of the further exercise of a copyright right for a particular work. The reader should understand that "copying" could include other types of rendering of the work for different devices. That is, "copying" in a printer would entail printing on paper or another substrate. Copying on a display is presenting an image on the screen. Copying in an audio device would be the performance of an audio portion of the work. Each of these devices both storage devices, e.g., hard disks, tapes in CDR, and rendering devices, e.g., prints, display graph, audio

player, movie player, should be equipped with the present invention so that the copies are controlled throughout the systems and networks until their final rendering place.

The present invention is a method for managing copies of a digital file, which includes content subject to copyright protection, on behalf of some content provider (e.g., an author, publisher or other). It is assumed that a given payment scheme has been established with respect to the file. Thus, for example, such payment schemes include, without limitation, prepayment (for "n" copies), pay-per-copy (as each copy is made), IOU (for copies made offline), or some other payment option. In a prepayment option, a user prepays funds for the right to obtain copies of the digital file. In a pay-per-copy (or "pay as you go") option, the user pays for each copy of the digital file when the file is copied. In an IOU scheme, the user makes copies of the digital file (e.g., while the client machine is not connected to the network) and generates an IOU (or many IOUs) that are then submitted to a clearinghouse or other payment entity when the user later goes online. Other payment schemes (such as a combination of the above options) may also be implemented.

The payment scheme is preferably defined in copy control information associated with the file and established by the author, publisher or some other third party. Thus, for example, the copy control information may also include a count of the number of permitted copies, a count of the number of permitted pay-per-copy versions, a count of the number of copies that may be made under an IOU payment option, copyright management information identifying the author, publisher and/or other license or use restrictions, information about a bank or other financial institution that handles use payments and their reconciliation, one or more expiration dates (after which copying is no longer permitted), and the like.

The copy control information associated with a given file thus defines a usage scheme for the file because it includes information that controls how the content may be used, how such use is paid for, over what period the content may be used, and other such information. A particular usage scheme (or some portion thereof) may also be implemented in the devices between which the file is transferred, although preferably such restrictions are defined by the content provider.

According to the present invention as illustrated in **Figure 2**, the present invention assumes the existence of a pair of devices, a "source" **24** and a "target" **26**, that have been or are certified to use the scheme. In particular, devices that implement the inventive scheme preferably include a device certificate that is not accessible (and thus is free from tampering) and stored therein. The certificate evidences that the device

is capable of understanding a given security protocol useful in carrying out the protection scheme. A representative security protocol is CSS, or the Content Scrambling System protocol, available commercially from Matsushita Corp. Thus, for example, if the source device is a disk storage, the device certificate is typically stored inside a secure chip within the device control hardware. Typically, each of the devices is "certified" upon manufacture, although this is not a requirement.

As also illustrated in **Figure 2**, a channel **28** is established between the source and target devices over which copies of a digital file (that is subject to the scheme) are communicated in a secure fashion. Thus, prior to transfer of the digital file, the channel **28** is first established between the devices to ensure that the copy restrictions (such as set forth in the copy control information) may be enforced. Typically, this is accomplished by having each device (in accordance with the security protocol implemented) require the other device (of the pair) to verify that its device certificate is valid. An appropriate message exchange may be used for this purpose as defined in the protocol. Once the secure link has been established, each of the devices can be trusted to control the digital file in accordance with the file's copy control information.

Typically, the "source" **24** is a storage device while the "target" **26** is a rendering device. An illustrative storage device may be disk storage, system memory, or the like. An illustrative rendering device may be a printer, a display, a sound card or the like. The source and target devices may both be storage devices (e.g., a Web server and a client disk storage).

When the technique is implemented in an "open" client-server environment, hardware devices (e.g., microcontrollers) are used in the storage and rendering devices to facilitate generation and management of the secure link. When less security may be tolerated, some of these functions may be implemented in software. When the technique is implemented in a "closed" Web appliance environment (**Figures 6A-6D**), the secure link may be established in whole or in part using software resident in the control routines associated with the storage and rendering devices. The secure link may be established in software under such conditions because, in the Web appliance environment, it is possible to readily disable the secure link in the event of tampering with the appliance housing or other circuitry. Regardless of the environment, the secure link is first established between the "certified" storage and rendering devices. Thereafter, the digital file, together with at least part of its copy control information, is transferable between the storage and rendering devices in accordance with the particular usage scheme defined, for example, by the copy control information. Thus, for example, if a

prepayment scheme is implemented and an expiration date (associated therewith) has not occurred, a given number of copies of the file may be transferred between the storage and rendering devices.

Thus, as illustrated in **Figure 2** in simplified form, the digital file copy protection method and system of the present invention involves a "source" device **24** (or one or more of such devices), and a set of one or more "target" devices **26**a-n connected via the secure channel or link **28**. The physical characteristics of the channel, of course, depend on whether the source and target devices are located in the same machine or are in separate machines connected via a network. In a network connection, the link may be a conventional TCP/IP connection. Channel **28** may be a physically secure channel (such as a https connection), but this is not required as the given security protocol in the certified devices establishes a secure link. According to the invention, once the link is established, one or more digital files are transferred (under the control of a control routine or mechanism) between the certified devices in an predictable, auditable manner so that (a) a controlled number of file transfers can be made, and (b) the precise number of file transfers (and their particular use) may be readily documented to facilitate dissemination of royalties or some such other consideration, typically to providers of such content. Generalizing, prior to transfer of a given digital file (or set of files, or file component) from the source to the target via the secure link, that transfer must first be authorized, and the transfer itself is then capable of being associated with some royalty payment then due to a content provider for use of such file. The scheme thus facilitates implementation of a generalized copyright management/royalty collection and distribution scheme.

As previously mentioned, the source **24** and target **26** may be located on the same computer. **Figure 3** illustrates this particular connection for a disk storage subsystem **24'** and the target rendering devices, namely printer **26a'**, display **26b'** and sound card **26c'**. The illustrated computer is a Web appliance, in which case the secure link may be established (as noted above) using software. Thus, in this example, each source and/or target device includes appropriate control software (part of software **15** as described above) to facilitate creation of the secure channel. Although not meant to be limiting, one convenient mechanism to create the channel involves each of the devices to generate a random number **30**, which numbers are then supplied to a key generation algorithm **32** in a known manner to generate a secret of "private" key **34**. The key **34** may be generated for each digital file to be transferred over the link **28**, or a signal key may be used for a set of such files, or even for a particular browsing session. To create the secure channel, the software resident on the disk storage encrypts the digital file as it leaves the source device.

The target device then decrypts the digital file using the key prior to rendering. In this way, the digital file cannot be readily intercepted as it is being transferred between these devices. As noted above, each of the source and target devices may also include secure chips or other known hardware devices to facilitate or augment such secure transfer of the digital file between the devices.

The particular mechanism for securing the channel between the source and target may be quite varied, and the present invention contemplates the use of any now known or later-developed technique, system or method for securing such communications. Thus, for example, another technique that may be used would be a public key cryptosystem.

Figure 4 is a block diagram illustrating a representative copyright royalty management system implemented according to the present invention. In this system, it is assumed that client computers **40** access the computer network **42** (e.g., the public Internet, an intranet, an extranet, or other computer network) to obtain access to Web-like documents supported on Web servers **44**. One or more management servers **46** are connectable to the system via an access provider **48**, and a control management server **50** may be used to facilitate scaling of the architecture if required. Control management server **50** may be controlled by a regulatory or rights agency that has responsibility for managing collection and distribution of copyright royalties.

A given management server includes a database **52** and appropriate control routines **54** for establishing a royalty account **55** for content providers. It is envisioned (although not required) that given content providers will subscribe to a royalty collection service implemented by the present invention and perhaps pay a fee (e.g., a commission or service charge) for the service provided. A given content provider thus may subscribe to the service to receive royalty payments for the use of his or her copyrighted content by users of the client machines. To this end, control routines **54** are used to establish an account for each of a set of given content providers, with each account including a representation of a given royalty value (which may be $0 when the account is established). A control routine then adjusts the given royalty value in a given provider account in response to receipt of an indication that a given digital file associated with the given content provider has been transferred from a source **24** to a target rendering device **26** in a given client computer **40**. Periodically, the content provider account is adjusted for any service or processing fees, and the remainder of the account is then distributed to the content provider. In the situation where the content provider is willing to allow his or her content (a given digital file) to be used with charges for such use paid later, a given bit may be set in the file's copy

control information indicating such preference. Other data in the copy control information may be used to set or control other content provider preferences with respect to use of the file within the context of the inventive scheme.

Figure 5 is a flowchart of one method of managing royalty account collection with respect to a particular digital file when a prepayment option is utilized. In this representative example, the digital file is an image (i.e. a .jpeg file) having a copyright owned by a given content proprietor or provider. Of course, the principles of the present invention are designed to be implemented collectively with many such digital files, and the following description is thus merely representative of one type of basic payment scheme. The routine assumes initially that a usage or payment account has been established for a given client computer (or a user of that computer). This is step 60 in the flowchart. It is also assumed that a royalty account has been established for the content provider at one of the management servers as previously described. This is step 62 in the flowchart. One of ordinary skill will appreciate that steps 60 and 62 need not be in any particular sequence. Step 60 typically involves the user prepaying some amount of funds into an account from which payments may be withdrawn, although this is not required.

At step 64, a count is established by a control routine for the particular digital file. Typically, this is a count of a number of permitted copies of the digital file that may be transferred from the source to one or more target devices according to the present invention. This number, as noted above, is typically identified in the file's copy control information. The count is usually a positive integer, which is then decremented (by the control routine) down to zero as permitted or authorized copies are made. Alternatively, of course, the count may begin at zero (or any other arbitrary number), which is then incremented (by the control routine) to the threshold value identified in the copy count information. As noted above, the count may be set by the copyright proprietor, by a system operator, by a Webmaster, by hardware constraints, or by any other party or entity having authority and/or ability to set the count. Under certain circumstances, e.g., where a prepaid user account is used, it may be unnecessary to use an explicit count as the number of copies transferred may simply depend on the royalty assessed per copy. Thus, the "count" as used herein may be expressed explicitly or implicitly. The digital file may be stored on the client already, or it may be available from a Web server or other storage or archive. The particular location from which the digital file is sourced initially does not matter. Step 64 assumes, however, that the image is located already at the source device. If the file is not present at the source, it may be

necessary to obtain it (although, conceptually, the "source" may be broadly construed as the original or initial location of the file).

At step 66, a test is done repeatedly to determine whether a request for the image has been received. If not, the routine cycles on step 66 and waits for such a request. If the outcome of the test at step 66 is positive, then the routine continues at step 68 by testing whether the given client computer (which generated the request) is authorized to effect the transfer. Step 68 may comprise a simple comparison of the user's account balance and the royalty amount to be assessed. If the user's account balance is large enough, the transfer may be allowed. Or, step 68 may simply test whether the count has a value indicating that further copies may be made. More typically, step 68 will require that the count be non-zero (in the situation where the count is positive and decremented to zero) and the user have sufficient funds allocated to pay the royalty assessment for use of the image. The step 68 may also test whether a given expiration date set in the copy count information has past.

If the outcome of the test at step 68 is negative, the transfer is not authorized, and the routine branches to step 70 to so notify the user of the client machine. Such notification may be in the form of an error or "access denied" message or the like. The user may be informed merely that a preset expiration date has passed or that his or her prepaid account is exhausted and requires more funds. If, however, the outcome of the test at step 68 is positive, the digital file may be transferred to the target. The routine then branches to step 72 to initiate the copy transfer. Preferably, all bytes of the file must be transferred before the transfer is considered valid. At step 74, the control routine count is adjusted (e.g., decremented) and/or a given charge is allocated against the user's account. The given charge may be equal to the royalty or use charge, or some fixed percentage thereof (e.g., 105%) reflecting that royalty plus some service charge). At step 76, the appropriate content provider account is adjusted by the amount of the royalty payment (plus or minus appropriate service fees or other charges).

Neither step 74 nor step 76 need occur at the time of the file transfer. Typically, the account adjustments will take place in batch at a given time. Thus, for example, where the Web client is a Web appliance connected to the computer network via a dialup connection, the account information may be transferred to the management server upon establishing a given connection (e.g. perhaps once each day). Other variations regarding the timing of delivery of this information are, of course, within the scope of the present invention.

The present invention thus provides numerous advantages. Certified source and target devices first establish a secure link between themselves. Upon transfer of the file copy between source and target, the control routine records an appropriate indication thereof in the copy count, and the central authority is notified of the transfer of the digital file. Such notification may occur upon transfer of the digital file between the source and target devices, or at some later time (e.g., upon dialup connection of the computer to the network). Royalty accounts are then managed at a central authority; to facilitate distribution of royalties to content owners/publishers. When the copy count reaches the authorized limit (as set in the copy control information), the control routine destroys the file or otherwise prevents further copying of the digital file.

Thus, in one embodiment, the user establishes a "prepaid" account from which royalty or usage payments are drawn against as files are copied/transmitted. The system detects use of the file and, preferably, allows only a certain number of copies of the file to be made before the document is destroyed or otherwise rendered inaccessible (from the client machine). The resulting copyright management infrastructure is robust, secure, scaleable and easily managed.

In one embodiment of this invention as described above, the Internet client is a data processing system or a so-called "Web appliance" such as illustrated in **Figures 6A-6D** and 7. **Figure 6A** is a pictorial representation of the data processing system as a whole. Data processing system **100** in the depicted example provides, with minimal economic costs for hardware to the user, access to the Internet. Data processing system **100** includes a data processing unit **102**. Data processing unit **102** is preferably sized to fit in typical entertainment centers and provides all required functionality, which is conventionally found in personal computers, to enable a user to "browse" the Internet. Additionally, data processing unit **102** may provide other common functions such as serving as an answering machine or receiving facsimile transmissions.

Data processing unit **102** is connected to television **104** for display of graphical information. Television **104** may be any suitable television, although color televisions with an S-Video input will provide better presentations of the graphical information. Data processing unit **102** may be connected to television **104** through a standard coaxial cable connection. A remote control unit **106** allows a user to interact with and control data processing unit **102**. Remote control unit **106** allows a user to interact with and control data processing unit **102**. Remote control unit **106** emits infrared (IR) signals, preferably modulated at a different frequency than the normal television, stereo, and VCR infrared remote

control frequencies in order to avoid interference. Remote control unit **106** provides the functionality of a pointing device (such as a mouse, glidepoint, trackball or the like) in conventional personal computers, including the ability to move a cursor on a display and select items.

Figure 6B is a pictorial representation of the front panel of data processing unit **102**. The front panel includes an infrared window **108** for receiving signals from remote control unit **106** and for transmitting infrared signals. Data processing unit **102** may transmit infrared signals to be reflected off objects or surfaces, allowing data processing unit **102** to automatically control television **104** and other infrared remote controlled devices. Volume control **110** permits adjustment of the sound level emanating from a speaker within data processing unit **102** or from television **104**. A plurality of light-emitting diode (LED) indicators **112** provide an indication to the user of when data processing unit **102** is on, whether the user has messages, whether the modem/phone line is in use, or whether data processing unit **102** requires service.

Figure 6C is a pictorial representation of the rear panel of data processing unit **102**. A three wire (ground included) insulated power cord **114** passes through the rear panel. Standard telephone jacks **116** and **118** on the rear panel provide an input to a modem from the phone line and an output to a handset (not shown). The real panel also provides a standard computer keyboard connection **120**, mouse port **122**, computer monitor port **124**, printer port **126**, and an additional serial port **128**. These connections may be employed to allow data processing unit **102** to operate in the manner of a conventional personal computer. Game port **130** on the rear panel provides a connection for a joystick or other gaming control device (glove, etc.). Infrared extension jack **132** allows a cabled infrared LED to be utilized to transmit infrared signals. Microphone jack **134** allows an external microphone to be connected to data processing unit **102**.

Video connection **136**, a standard coaxial cable connector, connects to the video-in terminal of television **104** or a video cassette recorder (not shown). Left and right audio jacks **138** connect to the corresponding audio-in connectors on television **104** or to a stereo (not shown). If the user has S-Video input, then S-Video connection **140** may be used to connect to television **104** to provide a better picture than the composite signal. If television **104** has no video inputs, an external channel 3/4 modulator (not shown) may be connected in-line with the antenna connection.

Figure 6D is a pictorial representation of remote control unit **106**. Similar to a standard telephone keypad, remote control unit **106** includes buttons **142** for Arabic numerals 0 through 9, the asterisk or "star" symbol

(*), and the pound sign (#). Remote control unit also includes "TV" button **144** for selectively viewing television broadcasts and "Web" button **146** for initiating "browsing" of the Internet. Pressing "Web" button **146** will cause data processing unit **102** to initiate modem dial-up of the user's Internet service provider and display the start-up screen for an Internet browser.

A pointing device **147**, which is preferably a trackpoint or "button" pointing device, is included on remote control unit **106** and allows a user to manipulate a cursor on the display of television **104**. "Go" and "Back" buttons **148** and **150**, respectively, allow a user to select an option or return to a previous selection. "Help" button **151** causes context-sensitive help to be displayed or otherwise provided. "Menu" button **152** causes a context-sensitive menu of options to be displayed, and "Update" button **153** will update the options displayed based on the user's input, while home button **154** allows the user to return to a default display of options. "PgUp" and "PgDn" buttons **156** and **158** allows the user to change the context of the display in display-sized blocks rather than by scrolling. The message button **160** allows the user to retrieve messages.

In addition to, or in lieu of, remote control unit **106**, an infrared keyboard (not shown) with an integral pointing device may be used to control data processing unit **102**. The integral pointing device is preferably a trackpoint or button type of pointing device. A wired keyboard (also not shown) may also be used through keyboard connection **120**, and a wired pointing device such as a mouse or trackball may be used through mouse port **122**. When a user has one or more of the remote control unit **106**, infrared keyboard, wired keyboard and/or wired pointing device operable, the active device locks out all others until a prescribed period of inactivity has passed.

Referring now to **Figure 7**, a block diagram for the major components of data processing unit **102** is portrayed. As with conventional personal computers, data processing unit **102** includes a motherboard **202** containing a processor **204** and memory **206** connected to system bus **280**. Processor **205** is preferably at least a **486** class processor operating at or above **100** MHz. Memory **206** may include cache memory and/or video RAM. Processor **205**, memory **206**, and system bus **208** operate in the same manner as corresponding components in a conventional data processing system.

Video/TV converter **210**, located on motherboard **202** and connected to system bus **208**, generates computer video signals for computer monitors, a composite television signal, and an S-Video signal. The functionality of Video/TV converter **210** may be achieved through a Trident TVG**9685** video

chip in conjunction with an Analog Devices AD722 converter chip. Video/TV converter 210 may require loading of special operating system device drivers.

Keyboard/remote control interface unit 212 on motherboard 202 receives keyboard codes through controller 214, regardless of whether a wired keyboard/pointing device or an infrared keyboard/remote control is being employed. Infrared remote control unit 106 transmits signals which are ultimately sent to the serial port as control signals generated by conventional mouse or pointing device movements. Two buttons on remote control unit 106 are interpreted identically to the two buttons on a conventional mouse, while the remainder of the buttons transmit signals corresponding to keystrokes on an infrared keyboard. Thus, remote control unit 106 has a subset of the function provided by an infrared keyboard.

Connectors/indicators 216 on motherboard 202 provide some of the connections and indicators on data processing unit 102 described above. Other connections are associated with and found on other components. For example, telephone jacks 116 and 118 are located on modem 222. The power indicator within connectors/indicators 216 is controlled by controller 214.

External to motherboard 202 in the depicted example are power supply 218, hard drive 220, modem 222 and speaker 224. Power supply 218 is a conventional power supply except that it receives a control signal from controller 214 which effects shut down of all power to motherboard 202, hard drive 220 and modem 222. Power supply 218, in response to a signal from controller 214, is capable of powering down and restarting data processing unit 102.

Controller 214 is preferably one or more of the 805x family controllers. Controller 214 receives and processes input from infrared remote control 106, infrared keyboard, wired keyboard, or wired mouse. When one keyboard or pointing device is used, all others are locked out (ignored) until none have been active for a prescribed period. Then the first keyboard or pointing device to generate activity locks out all others. Controller 214 also directly controls all LED indicators except that indicating modem use. As part of the failure recovery system, controller 214 specifies the boot sector selection during any power off-on cycle.

Hard drive 220 contains operating system and applications software for data processing unit 102, which preferably includes IBM DOS 7.0, a product of International Business Machines Corporation in Armonk, New York; an operating system 221 such as Windows 3.1 (or higher), a product

of Microsoft Corporation in Redmond, Washington; and a browser 223 such as Netscape Navigator (Version 1.0 or higher), a product of Netscape Communications Corporation in Mountain View, California. Hard drive 220 may also support an SMTP mechanism to provide electronic mail, an FTP mechanism to facilitate file transfers from Internet FTP sites, and other Internet protocol mechanisms, all in a known manner. Hard drive 220 is not generally accessible to the user of the Web appliance.

Modem 222 may be any suitable modem used in conventional data processing systems, but is preferably a 33.6 kbps modem supporting the V.42bis, V.34, V.17 Fax, MNP 1-5, and AT command sets. Modem 222 is connected to a physical communication link 227, which, in turn, in connected or connectable to the Internet (not shown).

Those skilled in the art will recognize that the components depicted in Figures 6A-6D and 7 and described above may be varied for specific applications or embodiments. Such variations in which the present invention may be implemented are considered to be within the spirit and scope of the present invention.

According to the invention, the client machine (typically the hard drive 220) also includes a proxy 225. Preferably, the proxy is implemented in software and includes a cache 227 associated therewith. The cache may be integral to the proxy or logically associated therewith. The cache preferably has a size up to several hundred megabytes, which is substantially larger than the standard cache associated with a browser such as Netscape Navigator. The client machine also includes a protocol stack 229 (e.g., a TCP/IP protocol stack) and a sockets mechanism 231, which are used to support communications in a known manner. According to the invention, the proxy 225 is advantageously located on the client along with the browser. Thus, the proxy is sometimes referred to as a "client side" proxy.

Preferably, the proxy starts up when the Web appliance is booted up. Connectivity between the proxy and the browser is achieved using the sockets mechanism by configuring the browser to pass the HTTP requests to the proxy. To send an HTTP GET request, the browser creates a packet (including the URL and other information) and then opens a socket using the sockets mechanism. The packet is then sent to the IP address/port number to service the HTTP request. Thus, when the browser issues an HTTP GET request, it binds to the socket and sends the request. The request is then intercepted and processed by the proxy instead of being sent directly over the network, all in the manner previously described.

Although in the preferred embodiment the client machine is a Web "appliance", this is not a requirement of the present invention. Thus, a client machine **10** may be a personal computer such as a desktop of notebook computer, e.g., an IBM® or IBM-compatible machine running under the OS/2® operating system, an IBM ThinkPad® machine, or some other Intel **x86** or Pentium®-based computer running Windows **95** (or the like) operating system.

A representative server platform comprises an IBM RISC System/6000 computer (a reduced instruction set of so-called RISC-based workstation) running the AIX (Advanced Interactive Executive Version **4.1** and above) Operating System **21** and Server program(s) **22**. The platform **20** also includes a graphical user interface (GUI) **23** for management and administration. It may also include an application programming interface (API) **24**. HTTP GET requests are transferred from the client machine to the server platform, typically via the dial-up computer network, to obtain documents or objects formatted according to HTML or some other markup language. While the above platform is useful, any other suitable hardware/operating system/server software may be used.

One of the preferred implementations of the client side or server side mechanisms of the invention is as a set of instructions (program code) in a code module resident in the random access memory of the computer. Until required by the computer, the set of instructions may be stored in another computer memory, for example, in a hard disk drive, or in a removable memory such as an optical disk (for eventual use in a CD ROM) or floppy disk (for eventual use in a floppy disk drive), or downloaded via the Internet or other computer network.

In addition, although the various methods described are conveniently implemented in a general purpose computer selectively activated or reconfigured by software, one of ordinary skill in the art would also recognize that such methods may be carried out in hardware, in firmware, or in more specialized apparatus constructed to perform the required method steps.

As used herein, "Web client" should be broadly construed to mean any computer or component thereof directly or indirectly connected or connectable in any known or later-developed manner to a computer network, such as the Internet. The term "Web server" should also be broadly construed to mean a computer, computer platform, an adjunct to a computer or platform, or any component thereof. Of course, a "client" should be broadly construed to mean one who requests or gets the file, and "server" is the entity which downloads the file. Moreover, although the present invention is described in the context of the Hypertext Markup Language

(HTML), those of ordinary skill in the art will appreciate that the invention is applicable to alternative markup languages including, without limitation, SGML (Standard Generalized Markup Language) and XML (Extended Markup Language).

In addition, the term "Web appliance" should be broadly construed to cover the display system illustrated in **Figures 6A-6D**, as well as any other machine in which a browser application is associated with some television class or other display monitor. Moreover, while the preferred embodiment is illustrated in the context of a dial-up network, this is not a limitation of the present invention. There may be other "bottleneck" resources in a direct connect network that could be managed indirectly by using this approach.

CLAIMS

1. A method for managing use of a digital file, comprising the steps of:

establishing a secure link between a pair of devices, each of the devices being certified to operate under a given security protocol;

establishing a usage scheme defining one or more conditions under which the digital file may be transferred between the pair of devices; and

transferring one or more copies of the digital file over the secure link between the pair of devices in accordance with the established usage scheme.

2. The method as described in Claim 1 wherein the pair of devices include a storage device and a rendering device.

3. The method as described in Claim 2 wherein the storage device and the rendering device are located in a computer.

4. The method as described in Claim 2 wherein the storage device is located in a first computer and the rendering device is located in a second computer and the secure link is established over a computer network connecting the first and second computers.

5. The method as described in Claim 4 wherein the second computer is a personal computer and the rendering device includes circuitry for establishing the secure link.

6. The method as described in Claim 4 wherein the second computer is a Web appliance and the rendering device includes software for establishing the secure link.

7. The method as described in Claim 2 wherein the rendering device is selected from a group of rendering devices consisting essentially of a printer, a display, and a sound card.

8. The method as described in Claim 1 further including the step of establishing an account representing a given monetary value.

9. The method as described in Claim 8 further including the step of allocating a given charge against the given monetary value when a copy of the digital file is transferred between the pair of devices.

10. The method as described in Claim 9 further including the step of associating the given charge with a content provider account to facilitate the payment of the given consideration to the provider of the digital file.

11. The method as described in Claim 1 wherein the usage scheme includes a given payment method.

12. A method for managing use of digital material in a computer network, comprising the steps of:

establishing an account for a given client computer including a representation of a given monetary value;

establishing an account for a given content provider including a representation of a given royalty value;

establishing a count of a number of permitted copies of a digital file;

in response to a given protocol, transferring a copy of the digital file from a source to a target associated with the given client computer;

adjusting the given monetary value in the account of the given client computer; and

adjusting the given royalty value in the account of the given content provider.

13. The method as described in Claim 12 wherein the given protocol includes the steps of:

determining whether a given client computer requesting transfer of the digital file is authorized to effect the transfer;

if the client is authorized to effect the transfer of the digital file, determining whether the count has a given value; and

if the count has the given value, transferring the digital file from the source to the target.

14. The method as described in Claim 13 wherein the given value is a non-zero value.

15. The method as described in Claim 13 wherein the given protocol further includes the step of adjusting the count after a copy of the digital file has been transferred.

16. The method as described in Claim 15 wherein the count is decremented.

17. The method as described in Claim 12 wherein the source and target are located in the given client computer connected to the computer network.

18. The method as described in Claim 17 wherein the source is a disk storage device and the target is a device selected from a group of rendering devices consisting essentially of a printer, a display, and a sound card.

19. The method as described in Claim 12 wherein the source is located on a first computer and the target is located on a second computer connected to the first computer via the computer network.

20. A method for managing use of digital material in a computer network including a Web client connectable to a Web server, comprising the steps of:

establishing a count of a number of permitted copies of a digital file located at a source device in the Web client;

in response to a given protocol, transferring one or more copies of the digital file from the source device to a set of one or more target rendering devices in the Web client; and

for each such transfer from the source device to one of the target rendering devices, logging an indication that the digital file has been transferred to facilitate payment of a given consideration to a provider of the digital file.

21. The method as described in Claim 20 wherein the Web client is a Web appliance and the source device is a secure disk storage.

22. The method as described in Claim 21 wherein each target rendering device is a device selected from a group of target rendering devices consisting essentially of a printer, a display, and a sound card.

23. The method as described in Claim 20 wherein the Web client is connected to the Web server via a non-secure connection.

24. The method as described in Claim 23 wherein the given protocol further includes the step of establishing a secure channel between the source device and a target rendering device prior to transferring the digital file.

25. The method as described in Claim 24 wherein the step of establishing a secure channel includes generating a secret key shared by the source device and the target rendering device.

26. The method as described in Claim 25 wherein the source device encrypts the digital file with the secret key as the source device transfers the digital file to the target rendering device, and wherein the target rendering device decrypts the digital file with the secret key upon receipt.

27. A computer program product in computer-readable media for use in a Web client having a source device and one or more target rendering devices, the computer program product comprising:

means for establishing a count of a number of permitted copies of a digital file located at the source device;

means, responsive to a given protocol, for transferring one or more copies of the digital file from the source device to the one or more target rendering devices;

means, responsive to each transfer, for logging an indication that the digital file has been transferred to facilitate payment of a given consideration to a provider of the digital file; and

means responsive to the logging means for adjusting the count.

28. The computer program product as described in Claim 27 further including means responsive to a given occurrence for transferring the indication to a central authority.

29. The computer program product as described in Claim 28 wherein the given occurrence is establishing a dialup connection between the Web client and an Internet Service Provider.

30. A computer system connected to a computer network and including a source device and one or more target rendering devices, comprising:

a processor;

an operating system;

an application for managing use of digital material, comprising:

means for establishing a count of a number of permitted copies of a digital file located at the source device;

means, responsive to a given protocol, for transferring one or more copies of the digital file from the source device to the one or more target rendering devices;

means, responsive to each transfer, for logging an indication that the digital file has been transferred to facilitate payment of a given consideration to a provider of the digital file; and

means responsive to the logging means for adjusting the count.

31. The computer system as described in Claim 30 wherein the application further includes means for restricting transfer of the digital file when the count reaches a given value.

32. A data processing system, comprising:

a remote control unit; and

a base unit connectable to a monitor for providing Internet access under the control of the remote control unit, the base unit comprising:

a processor having an operating system;

a browser application run by the operating system;

a secure disk storage in which a digital file is stored;

one or more target rendering devices; and

means for restricting a number of copies of the digital file that may be transferred between the secure disk storage and the one or more target rendering devices.

33. The data processing system as described in Claim 32 wherein the restricting means includes means responsive to a given occurrence for transmitting an indication of a number of copies of the digital file that were transferred between the secure disk storage and the one or more target rendering devices during a given time interval.

34. The data processing system as described in Claim 33 wherein the given occurrence is a dialup connection of the data processing system to an Internet Service Provider.

35. A management server for use in managing collection and allocation of royalties among content providers, the management server connected in a computer network to an access provider servicing a plurality of Web client appliances receiving dialup access to Web content, the management server comprising:

means for establishing an account for each of set of given content providers, each account including a representation of a given royalty value; and

means for adjusting the given royalty value in the account of the given content provider in response to receipt of an indication that a given digital file associated with the given content provider has been transferred from a source to a target rendering device in a given Web client appliance.

36. A copy management system, comprising:

a first device and a second device, each of which is certified to operate under a given security protocol;

means for establishing a secure link between the first and second devices; and

means responsive to establishment of the secure link for managing transfer of a permitted number of copies of a digital file between the first and second devices in accordance with copy control information restrictions associated with the digital file.

FIG. 1

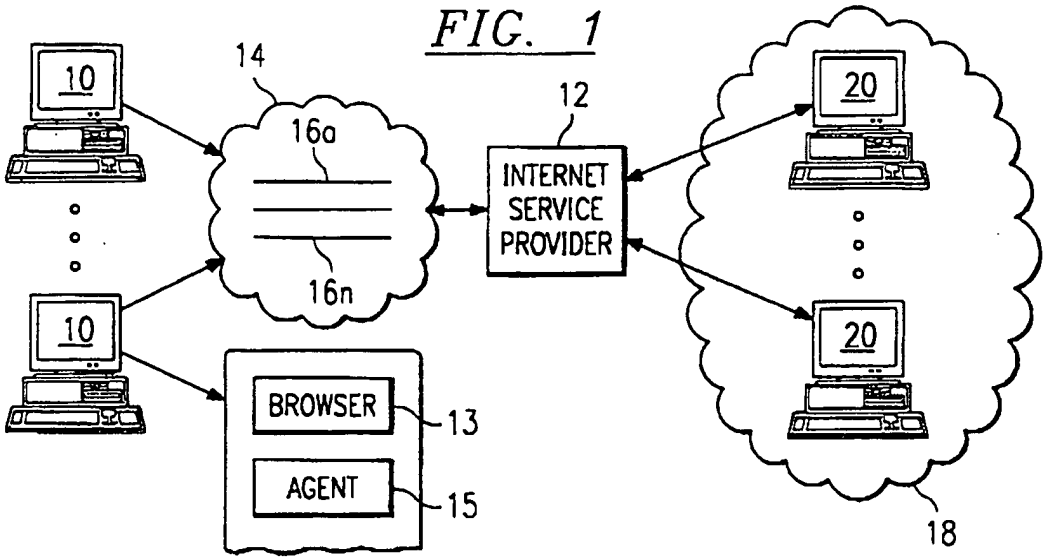


FIG. 2

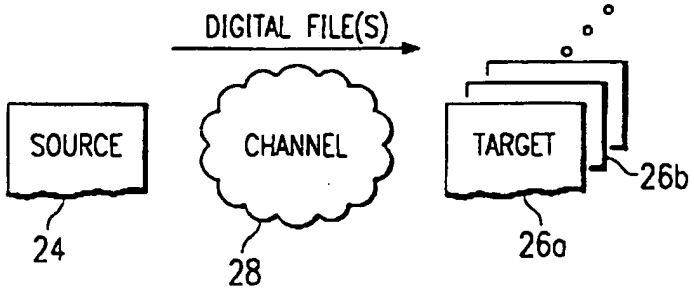


FIG. 3

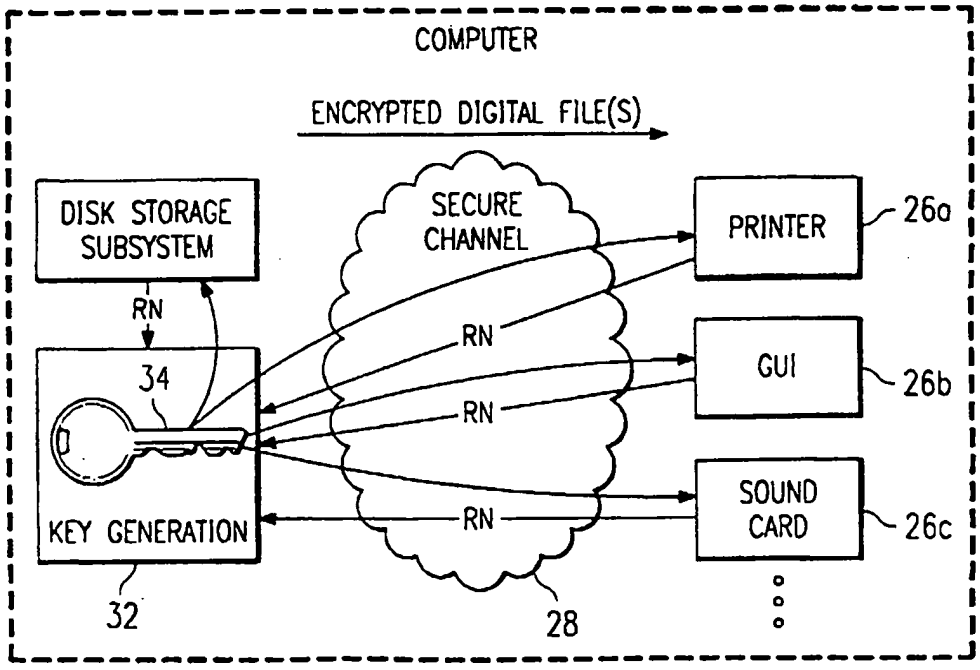
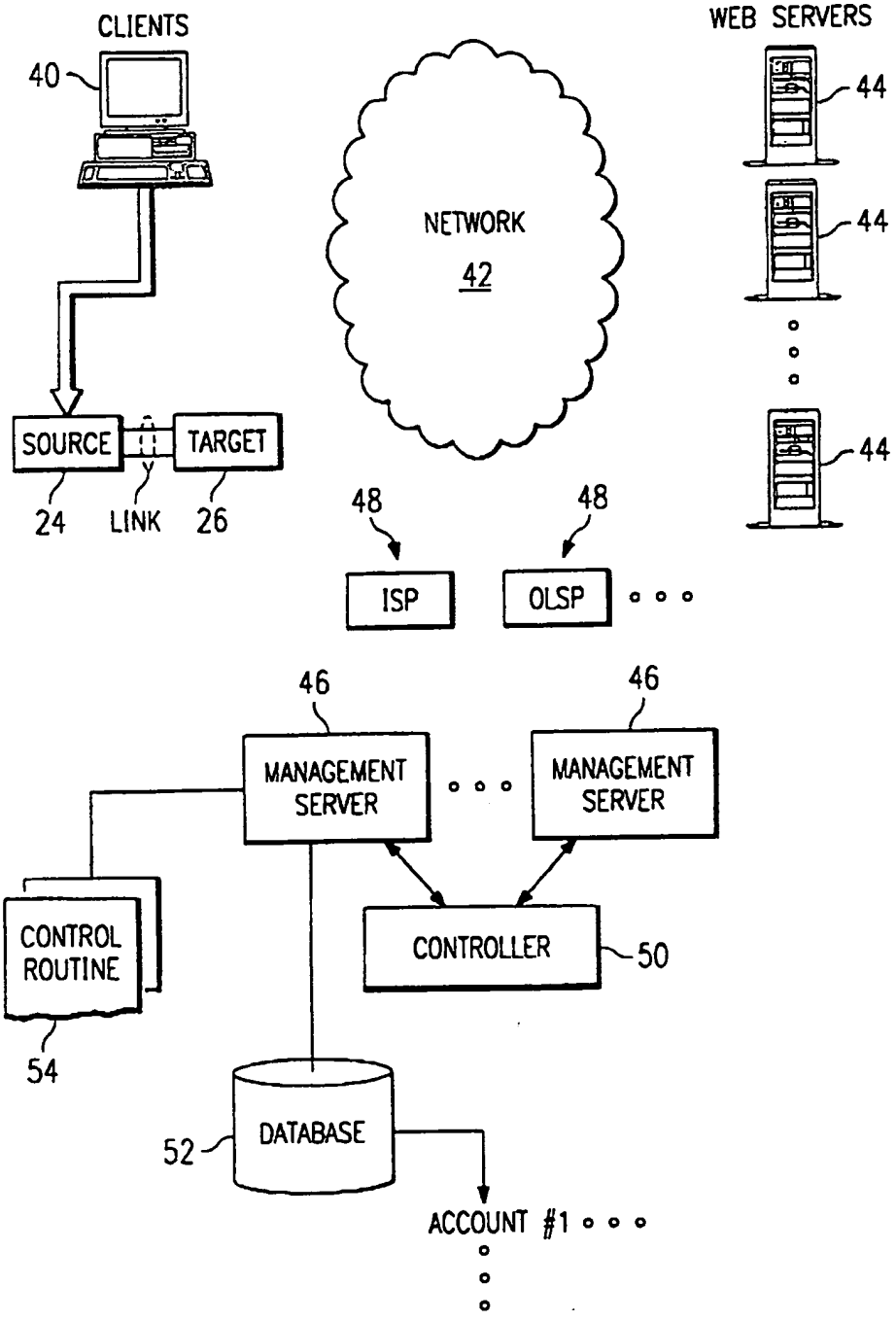


FIG. 4


CLIENTS
40
WEB SERVERS
44
44
44
NETWORK
42
SOURCE
TARGET
24
LINK
26
48
48
ISP
OLSP
46
46
MANAGEMENT SERVER
MANAGEMENT SERVER
CONTROL ROUTINE
54
CONTROLLER
50
DATABASE
52
ACCOUNT #1

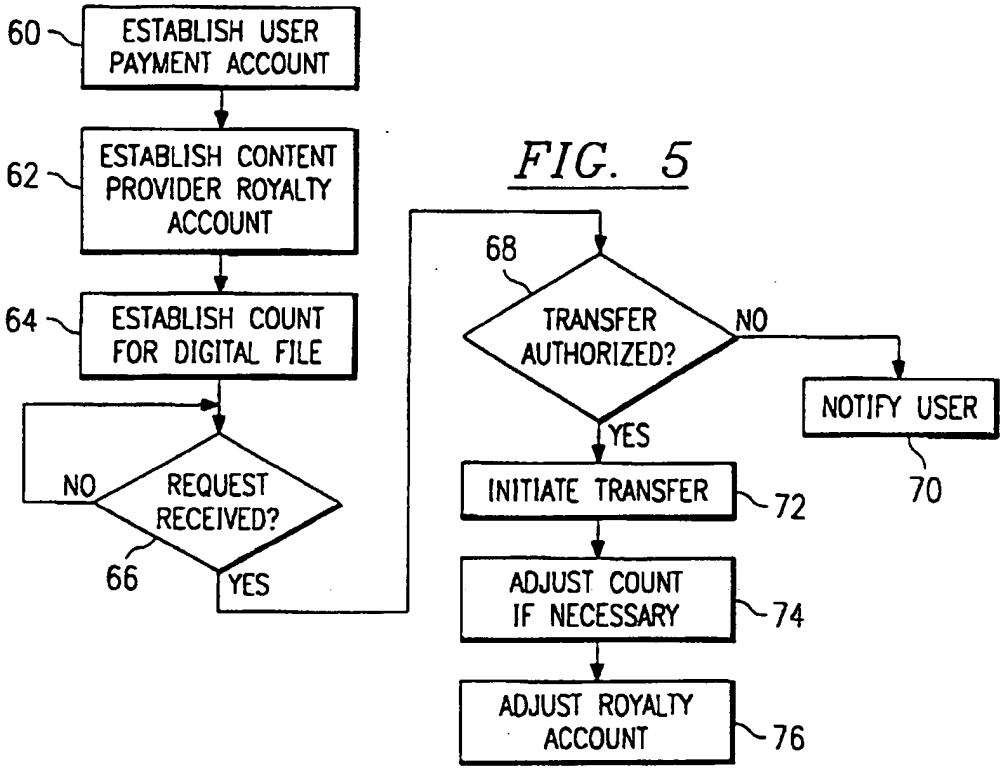

60
ESTABLISH USER PAYMENT ACCOUNT
62
ESTABLISH CONTENT PROVIDER ROYALTY ACCOUNT
64
ESTABLISH COUNT FOR DIGITAL FILE
NO
REQUEST RECEIVED?
66
YES
68
TRANSFER AUTHORIZED?
NO
NOTIFY USER
70
YES
INITIATE TRANSFER
72
ADJUST COUNT IF NECESSARY
74
ADJUST ROYALTY ACCOUNT
76

FIG. 5

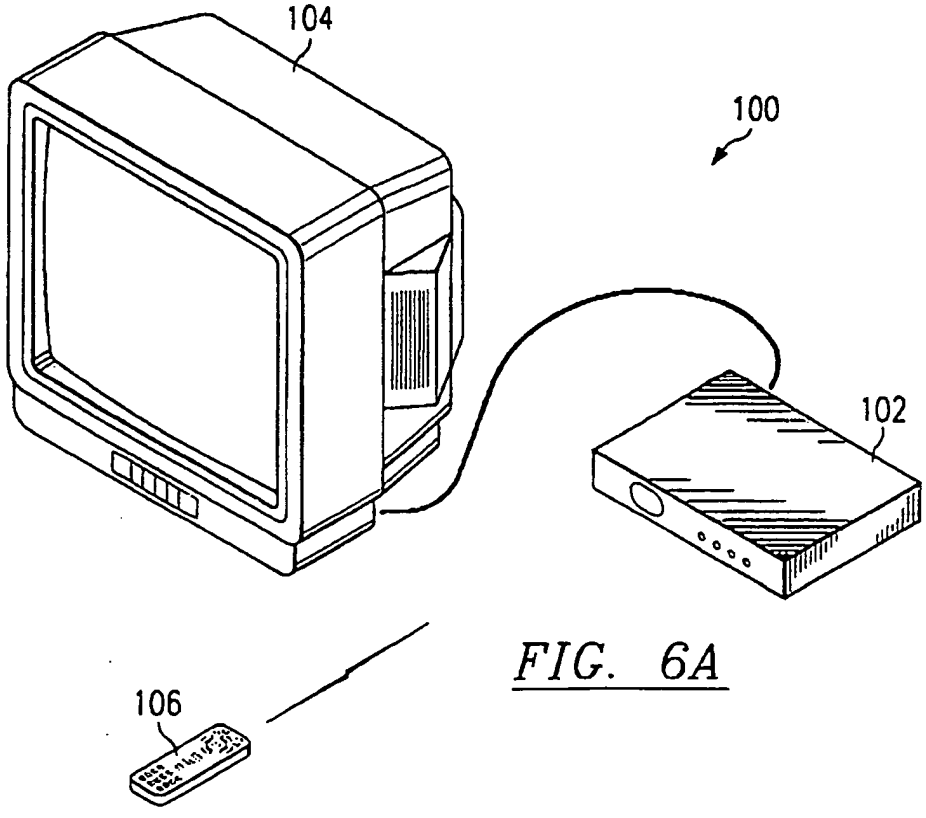

104
100
102
106

FIG. 6A

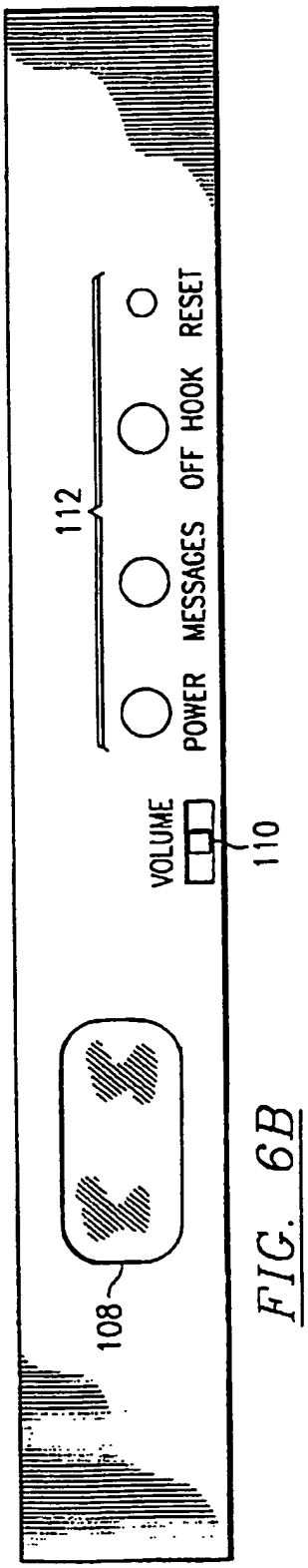

112
POWER
MESSAGES
OFF HOOK
RESET
VOLUME
110
108

FIG. 6B

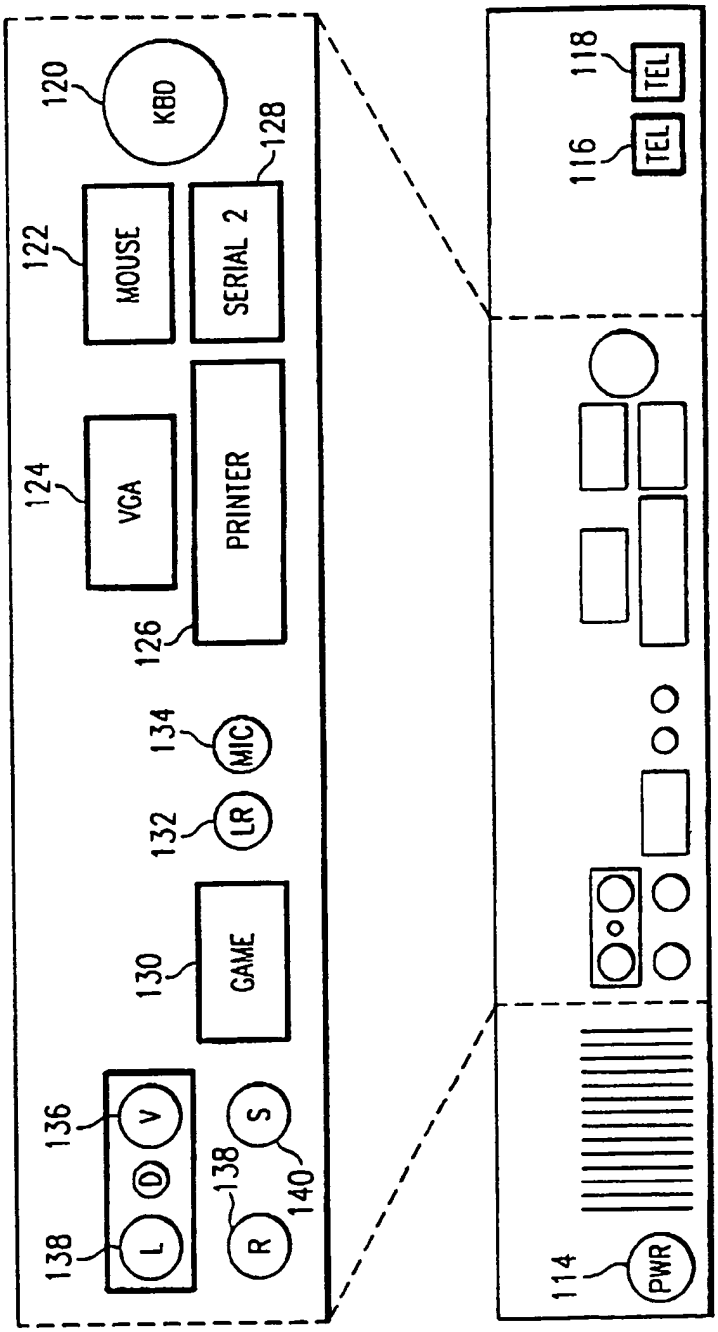

120
KBD
122
MOUSE
128
SERIAL 2
124
VGA
126
PRINTER
134
MIC
132
LR
130
GAME
136
V
138
L
138
R
140
S
116
TEL
118
TEL
114
PWR

FIG. 6C

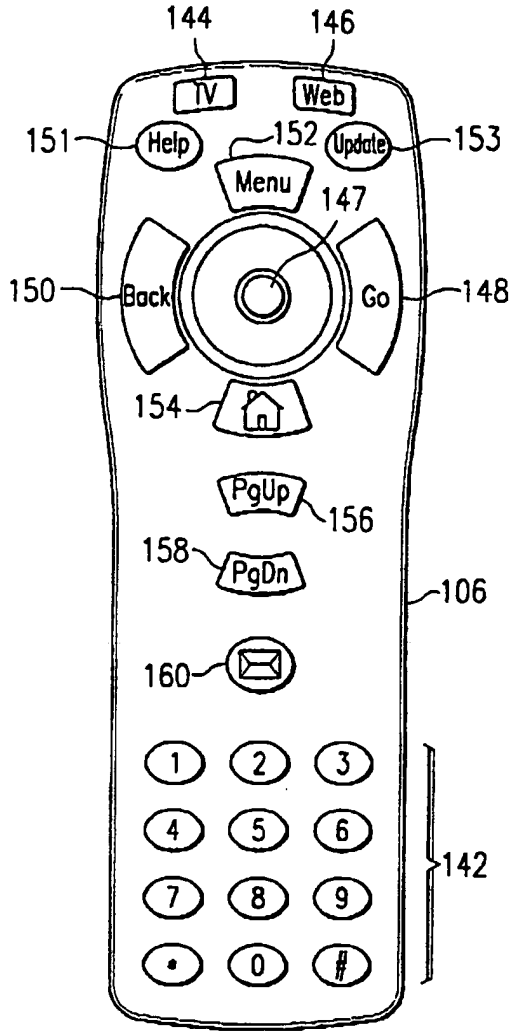
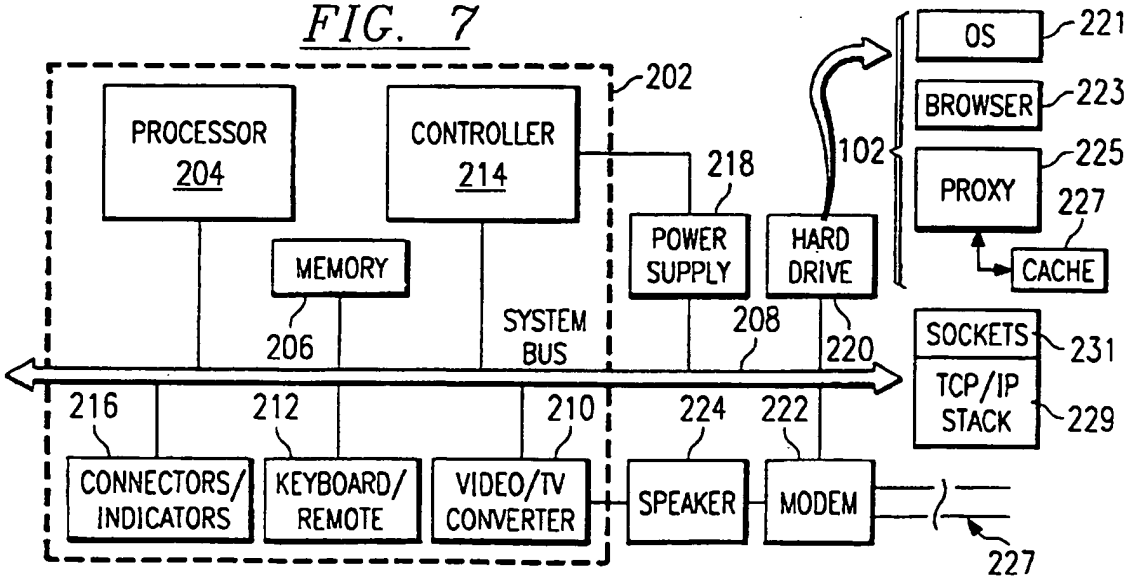

144
146
TV
Web
151
Help
152
Update
153
Menu
147
150
Back
Go
148
154
PgUp
156
158
PgDn
106
160
1
2
3
4
5
6
7
8
9
•
0
#
142

FIG. 6D

FIG. 7


202
PROCESSOR
204
CONTROLLER
214
218
MEMORY
206
SYSTEM
BUS
POWER
SUPPLY
208
HARD
DRIVE
220
102
OS
221
BROWSER
223
PROXY
225
227
CACHE
SOCKETS
231
TCP/IP
STACK
229
216
212
210
224
222
CONNECTORS/
INDICATORS
KEYBOARD/
REMOTE
VIDEO/TV
CONVERTER
SPEAKER
MODEM
227

INTERNATIONAL SEARCH REPORT

Int ional Application No
PCT/GB 98/03828

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5 532 920 A (HARTRICK THOMAS V ET AL) 2 July 1996 see figures 1,2 see column 6, line 44 - column 7, line 15 see column 14, line 51 - column 16, line 25 ---	1-20, 23-25, 27-33, 35,36
Y	EP 0 798 906 A (SUN MICROSYSTEMS INC) 1 October 1997 see figures 1-3 see column 4, line 21 - column 5, line 33 -----	1-20, 23-25, 27-33, 35,36

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 March 1999	08/04/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Weiss, P

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int ional Application No
PCT/GB 98/03828

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5532920 A	02-07-1996	EP 0567800 A	03-11-1993
		JP 2659896 B	30-09-1997
		JP 6103286 A	15-04-1994
EP 0798906 A	01-10-1997	US 5761421 A	02-06-1998
		JP 10105529 A	24-04-1998

Petitioner Apple Inc. - Exhibit 1004, p. 5915

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 332 304 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 89301510.7

(51) Int. Cl.4: G06F 1/00

(22) Date of filing: 16.02.89

(30) Priority: 07.03.88 US 164944

(43) Date of publication of application:
13.09.89 Bulletin 89/37

(84) Designated Contracting States:
DE FR GB

(71) Applicant: DIGITAL EQUIPMENT CORPORATION
111 Powdermill Road
Maynard Massachusetts 01754-1418(US)

(72) Inventor: Robert, Gregory
12 Carson Circle
Nashua New Hampshire 03062(US)
Inventor: Chase, David
28 Bay View Road
Wellesley Massachusetts 02181(US)
Inventor: Schaefer, Ronald
7 Gioconda Avenue
Acton Massachusetts 01720(US)

(74) Representative: Goodman, Christopher et al
Eric Potter & Clarkson 14 Oxford Street
Nottingham NG1 5BP(GB)

(54) Software licensing management system.

(57) A license management system which includes a license management facility that determines whether usage of a licensed program is within the scope of the license. The license management system maintains a license unit value for each licensed program and a pointer to a table identifying an allocation unit value associated with each use of the licensed program. In response to a request to use a licensed program, the license management system responds with an indication as to whether the license unit value exceeds the allocation unit value associated with the use. Upon receiving the response, the operation of the licensed program depends upon policies established by the licensor.

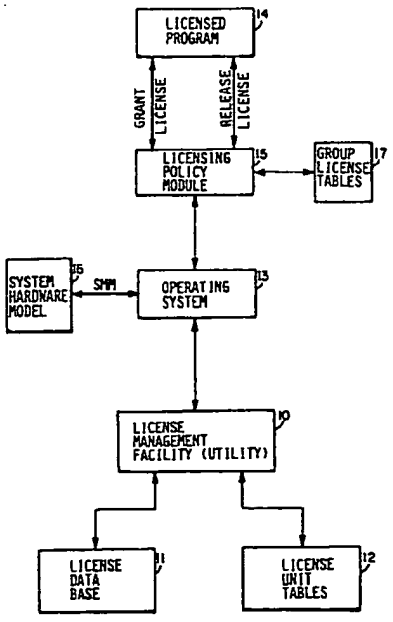


FIG. 1

EP 0 332 304 A2

Petitioner Apple Inc. - Exhibit 1004, p. 5916

SOFTWARE LICENSING MANAGEMENT SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of digital data processing systems, and more specifically to a system for managing licensing for, and usage of, the various software programs which may be processed by the systems to ensure that the software programs are used within the terms of the software licenses.

2. Description of the Prior Art

A digital data processing system includes three basic elements, namely, a processor element, a memory element and an input/output element. The memory element stores information in addressable storage locations. This information includes data and instructions for processing the data. The processor element fetches information from the memory element, interprets the information as either an instruction or data, processes the data in accordance with the instructions, and returns the processed data to the memory element for storage therein. The input/output element, under control of the processor element, also communicates with the memory element to transfer information, including instructions and data to be processed, to the memory, and to obtain processed data from the memory.

Typically, an input/output element includes a number of diverse types of units, including video display terminals, printers, interfaces to the public telecommunications network, and secondary storage subsystems, including disk and tape storage devices. A video display terminal permits a user to run programs and input data and view processed data. A printer permits a user to obtain a processed data on paper. An interface to the public telecommunications network permits transfer of information over the public telecommunications network.

The instructions processed by the processor element are typically organized into software programs. Recently, generation and sales of software programs have become significant businesses both for companies which are primarily vendors of hardware, as well as for companies which vend software alone. Software is typically sold under license, that is, vendors transfer copies of software to users under a license which governs how the users may use the software. Typically, software costs are predicated on some belief as to the amount of usage which the software program may provide and the economic benefits, such as cost saving which may otherwise be incurred, which the software may provide to the users. Thus, license fees may be based on the power of the processor or the number of processors in the system, or the number of individual nodes in a network, since these factors provide measures of the number of users which may use the software at any given time.

In many cases, however, it may also be desirable, for example, to have licenses and license fees more closely relate to the actual numbers of users which can use the program at any given time or on the actual use to which a program may be put. Furthermore, it may be desirable to limit the use of the program to specified time periods. A problem arises particularly in digital data processing systems which have multiple users and/or multiple processors, namely, managing use of licensed software to ensure that the use is within the terms of the license, that is, to ensure that the software is only used on identified processors or by the numbers of users permitted by the license.

SUMMARY OF THE INVENTION

The invention provides a new and improved licensing management system for managing the use of licensed software in a digital data processing system.

In brief summary, the license management system includes a license management facility and a licensing policy module that jointly determine whether a licensed program may be operated. The license management facility maintains a license unit value for each licensed program and a pointer to a table identifying a license usage allocation unit value associated with usage of the licensed program. In response to a request to use a licensed program, the license management facility determines whether the remaining license unit value exceeds the license usage allocation unit value associated with the use. If the license unit value does not exceed the license usage allocation unit value, the license management facility permits usage of the licensed program and adjusts the license unit value by a function of the license usage allocation unit value to reflect the usage. On the other hand, if the license unit value associated with use of the license program does exceed the li-

Petitioner Apple Inc. - Exhibit 1004, p. 5917

cense usage allocation unit value, the licensing policy module determines whether to allow the licensed program to be used in response to other licensing policy factors.

BRIEF DESCRIPTION OF THE DRAWINGS

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

Fig. 1 is a general block diagram of a new system in accordance with the invention;

Figs. 2 and 3 are diagrams of data structures useful in understanding the detailed operation of the system depicted in Fig. 1; and

Figs. 4A-1 through 4B-2 are flow diagrams which are useful in understanding the detailed operations of the system depicted in Fig. 1.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

Fig. 1 depicts a general block diagram of a system in accordance with the invention for use in connection with a digital data processing system which assists in managing software use in accordance with software licenses. With reference to Fig. 1, the new system includes a license management facility 10 which operates in conjunction with a license data base 11 and license unit tables 12, and under control of an operating system 13 and licensing policy module 15 to control use of licensed programs, such as licensed program 14, so that the use is in accordance with the terms of the software license which controls the use of the software program on a system 16 identified by a system marketing model (SMM) code in a digital data processing system.

As is conventional, the digital data processing system including the licensing management system may include one or more systems 16, each including one or more processors, memories and input/output units, interconnected in a number of ways. For example, the digital data processing system may comprise one processor, which may include a central processor unit which controls the system and one or more auxiliary processors which assist the central processor unit. Alternatively, the digital data processing system may comprise multiple processing systems, in which multiple central processor units are tightly coupled, or clustered or networked systems in which multiple central processor units are loosely coupled, generally operating relatively autonomously, interacting by means of messages transmitted over a cluster or network connection. In a tightly coupled multiple processing system, for example, it may be desirable to control the number of users which may use a particular software program at one time. A similar restriction may be obtained in a cluster or network environment by controlling the number of particular nodes, that is, connections to the communications link in the cluster or network over which messages are transferred. In addition, since the diverse processors which may be included in a digital data processing system may have diverse processing speeds and powers, represented by differing system marketing model (SMM) codes, it may be desirable to include a factor for speeds and power in determining the number of processors on which a program may be used concurrently.

As will be explained in greater detail below, the license data base 11 contains a plurality of entries 20 (described below in connection with Fig. 2) each containing information relating to the terms of the license for a particular licensed program 14. In one embodiment such information may include a termination date, if the license is for a particular time period or expires on a particular date, and a number of licensing units if the license is limited by usage of the license program. In that embodiment, the entry also includes identification of a license unit table 40 (described below in connection with Fig. 3) in the license unit tables 12 that identifies the number of allocation units for usage of the licensed program on the types of systems 16 which may be used in the digital data processing system as represented by the system marketing model (SMM) codes.

When a user wishes to use a licensed program 14, a GRANT LICENSE request message is generated which requests information as to the licensing status of the licensed program 14. The GRANT LICENSE request message is transmitted to the licensing policy module 15, which notifies the operating system of the request. The operating system 13, in turn, passes the request, along with the system marketing model of the specific system 16 being used by the user, to the license management facility 10 which determines whether use of the program is permitted under the license.

In response to the receipt of the GRANT LICENSE request from the user and the system marketing model (SMM) code of the system 16 being used by the user on which the licensed program will be processed, the license management facility 10 obtains from the license data base the entry 20 associated with the licensed program

Petitioner Apple Inc. - Exhibit 1004, p. 5918

14 and determines whether the use of the licensed program 14 is within the terms of the license as indicated by the information in the license data base 11 and the license unit tables 12.

In particular, the license management facility 10 retrieves the contents of the entry 20 associated with the licensed program. If the entry 20 indicates a termination data, the license management facility 10 compares the system data, which is maintained by the digital data processing system in a conventional manner, with the termination date identified in the entry. If the system date is after the termination date identified in the entry 20, the license has expired and the license management facility 10 generates a usage disapproved message, which it transmits to the operating system 13. On the other hand, if the termination date indicated in the entry 20 is after the system date, the license has not expired and the license management facility 10 proceeds to determine whether the usage of the licensed program 14 is permitted under other terms of the license which may be embodied in the entry 20.

In particular, the license management facility 10 then determines whether the usage of the licensed program is permitted under usage limitations. In that operation, the license management facility obtains the number of license units remaining, which indicates usage of the licensed program 14 not including the usage requested by the user, as well the identification of the table 40 in license unit tables 12 associated with the licensed program 14. The license management facility 10 then compares the number of license units which would be allocated for use of the licensed program 14, which it obtains from the table 40 identified by entry 20 in the license data base 11, and the number of remaining units to determine whether sufficient license units remain to permit usage of the licensed program 14.

If the number of remaining license units indicated by entry 20 in the license data base 11 exceeds the number, from license unit tables 12, of license units which would be allocated for use of the licensed progam 14, the usage of the licensed program is permitted under the license. Accordingly, the license management facility transmits a usage approved response to the operating system 13. In addition, the license management facility 10 adjusts the number of remaining license units in entry 20 by a function of the license units allocated to use of the licensed program to reflect the usage.

On the other hand, if the number of remaining license units indicated by entry 20 in the license data base is less than the number of license units which would be allocated for use of the licensed program 14, the usage of the licensed program 14 is not permitted by the license. In that case, the license management facility 10 transmits a usage disapproved response to the operating system 13. In addition, the license management facility 10 may also log the usage disapproved response; this information may be used by a system operator to determine whether usage of the licensed program 14 is such as to warrant obtaining an enlarged license.

Upon receipt of either a usage approved response or a usage disapproved response to the GRANT LICENSE request, the operating system 13 passes the response to the licensing policy module 15. If a usage approved response is received, the licensing policy module normally allows usage of the licensed program 14. If a usage disapproved response is received, the licensing policy module determines whether the usage of the licensed program may be permitted for other reasons. For example, usage of the licensed program 14 may be permitted under a group license, whose terms are embodied in entries in group license tables 17. Under a group license, usage may be permitted of any of a group of licensed programs. The operations to determine to whether usage is permitted may be performed in the same manner as described above in connection with license management facility 10. In addition, if the usage of the licensed program 14 is not permitted under a group license, usage may nonetheless be permitted under the licensor's licensing practices, which may be embodied in the licensing policy module 15. If the licensing policy module determines that usage of the program should be permitted, notwithstanding a usage disapproved response from the license management facility 10, because the usage is permitted under a group license or the licensor's licensing practices, the licensing policy module 15 permits usage of the licensed program. Otherwise, the licensing policy module does not permit usage of the licensed program in response to the GRANT LICENSE request.

When a user no longer requires use of a licensed program 14, it transmits a RELEASE LICENSE request to the licensing policy module 15. The operations performed by the licensing policy module depend on the basis for permitting usage of the licensed program. If usage was permitted as a result of a group license, if the group license is limited by usage, the licensing policy module 15, if necessary, adjusts the records in the group license tables 17 related to the group license to reflect the fact that the licensed program 14 related to the group license is not being used. If the usage was permitted as a result of a group license which is not limited by usage, but instead is limited in duration, or if the usage was permitted in response to the licensor's licensing policies, the licensing policy module 15 need do nothing. If the licensing

Petitioner Apple Inc. - Exhibit 1004, p. 5919

policy module 15 maintains a log of usage outside the scope of a group or program license, it may make an entry in the log of the RELEASE request.

Finally, if usage was permitted as a result of the license management facility 10 providing an approve usage response to the GRANT LICENSE request, the licensing policy module 15 transmits the RELEASE LICENSE request to the operating system 13. In response, the operating system 13 transfers the RELEASE LICENSE request to the license management facility 10, along with an identification of the system 16 using the licensed program 14. The license management facility 10 then obtains from the license data base the identification of the appropriate license usage allocation unit value table in license unit tables 12, and determines the number of allocation units associated with this use of the licensed program 14 based on the identified allocation table and the processor. The license management facility 10 then adjusts the number of license units for the licensed program 14 in the license data base 11 to reflect the release.

It will be appreciated by those skilled in the art that, the license management facility 10 may, in response to a GRANT LICENSE request, instead of deducting allocation units from the entries in the license data base 11 associated with the licensed programs 14, determine the number of allocation units which would be in use if usage of the licensed program 14 is permitted, and respond based on that determination. If the license mangement facility 10 operates in that manner, it may be advantageous for the entries in license data base 11 relating to each licensed program 14 to maintain a running record of the number of allocation units associated with its usage. The licensing policy module 15 may operate similarly in connection with group licenses that are limited by usage.

It will also be appreciated that the new license management system thus permits the digital data processing system to control use of a licensed program 14 based on licensing criteria in the license data base 11, the license unit tables 12, the group licensing tables 17 and the licensor's general licensing policies rather than requiring an operator to limit or restrict use of a licensed program or charging for the license based on some function of the capacity of all of the processors in the digital data processing system. The new license management system allows for very flexible pricing of licenses and licensing policies, since the digital data processing system itself enforces the licensing terms controlling use of the licensed programs 14 in the system.

Fig. 2 depicts the detailed structure of the license data base 12 (Fig. 1) used in the license management system depicted in Fig. 1. With reference to Fig. 2, the license data base includes a plurality of entries generally identified by reference numberal 20, with each entry being associated with one licensed program 14. Each entry 20 includes a number of fields, including an issuer name field 21 identifying the issuer of the license, an authorization number field 22 which contains an authorization number, a producer name field 23 which identifies the name of the vendor of the licensed program, and a product name field 24 which contains the name of the licensed program. The contents of these fields may be used, for example, in connection with other license management operations, such as determining the source of licensed programs in the event of detection of errors in programs, and in locating duplicate entries in the license data base or entries which may be combined as a result of licenses being obtained and entered by, perhaps different operators or at different times.

Each entry 20 in the licensing data base 11 also includes a license number field 25 whose contents identify the number of licensing units remaining. A license of a licensed program 14 identifies a number of licensing units, which may be a function of the price paid for the license. An availability table field 26 and an activity table field 27 identify license usage allocation unit value tables in the license unit tables 12 (described in connection with Fig. 3) to be used in connection with the GRANT LICENSE and RELEASE LICENSE requests.

By way of background, a license may be in accordance with a licensing paradigm which requires concurrent use of the licensed program 14 on several processors to be a function of the processor power and capacity, and the availability table field 26 identifies a license usage allocation unit table to be used in connection with that. In an alternative, a license may be in accordance with a licensing paradigm which requires concurrent use of the licensed program to be a function of the number of users using the program, and the activity table field 27 identifies a license usage allocation unit valve table in the license unit tables 12 to be used in connection with that. If either licensing paradigm is used to the exclusion of the other, one field contains a non-zero value and the other field contains a zero value. In addition, a license may be in accordance with both licensing paradigms, that is, concurrent use of a program may be limited by both processor power and capacity and by the number of concurrent users, and in that case both fields 26 and 27 have non-zero values.

In one embodiment of the licensing management system, fields 21 through 27 of an entry 20 in the licensing data base 11 are required. In that embodiment, an entry 20 in the licensing data may

Petitioner Apple Inc. - Exhibit 1004, p. 5920

also have several optional fields. In particular, an entry 20 may include a date/version number field 30 whose contents comprise either a date or version number to identify the licensed program. If a license is to terminate on a specific date, the entry 20 may include a licensor termination date field 31 or a licensee termination date field 32 whose contents specify the termination date assigned by the licensor or licensee. This may be particularly useful, for example, as a mechanism for permitting licensees to demonstrate or try a program before committing to a long or open term license.

Finally, an entry 20 in the license data base includes a checksum field 33, which includes a checksum of the contents of the other fields 21 through 27 and 30 through 32 in the entry 20, which may be established by means of a mathematical algorithm applied to the contents of the various fields. The general mechanism for establishing checksums is well known in the art, and will not be described further herein. The contents of all fields 21 through 27 and 30 through 33 of a new entry 20 are entered by an operator. Prior to establishment of an entry in the license data base 11, the license management facility 10 may verify correct entry of the information in the various fields by calculating a checksum and comparing it to the checksum provided by the operator. If the checksum provided by the operator and the checksum determined by the license management facility are the same, the entry 20 is established in the license data base 11. On the other hand, if the checksum provided by the operator and the checksum determined by the license management facility differ, the license management facility 10 determines that the information is erroneous or the license is invalid and does not establish the entry 20 in the license data base 11. It will be appreciated that, if the checksum-generation algorithm is hidden from an operator, the checksum provides a mechanism for verifying, not only that the information has been properly loaded into the entry, but also that the license upon which the entry is based is authorized by the licensor.

The structure of group license tables 17 may be similar to the structure of the license data base 11, with the addition that the entries for each license reflected in the group license tables 17 will need to identify all of the licensed programs covered thereby.

As described above, the licensing unit tables 12 (Fig. 1) contain information as to the allocation units for use in determining the number of licensing units associated with use of a licensed program. The structure of a licensing unit table 40 is depicted in Fig. 3. With reference to Fig. 3, the licensing unit table includes a plurality of entries 41(1) through 41(N) (generally identified by reference numeral 41) each identified by a particular type of processor. One entry 41 in the table 40 is provided for each type of processor which can be included in the digital data processing system which can use the licensed programs 14 which reference the license unit table 40. The processor associated with each entry is identified by a processor identification field 42. The successive fields in the entries 41 (which form the various columns in the table 40 depicted in Fig. 3) form license usage allocation unit value tables 43(1) through 43-(M) (generally identified by reference numeral 43). The contents of the availability table field 26 and the activity table field 27 identify a license usage allocation unit value table 43. If there are non-zero contents in both availability field 26 and activity field 27, the contents which identify be the same license usage allocation unit value table 43 or different license usage allocation unit value tables 43. As described above, the contents of the license usage allocation unit value table identify the number of licensing units associated with use of the licensed programs which identify the particular license usage allocation unit value table, for each of the identified processors.

The operation of the licensing management system is depicted in detail in Figs. 4A-1 through 4-B. Figs. 4A-1 through 4A-4 depict, in a number of steps the details of operation of the licensing management system in connection with the GRANT LICENSE request from a licensed program 14. Figs. 4B-1 and 4B-2 depict, in a number of steps, the details of operation in connection with the RELEASE LICENSE request from a licensed program 14. In the Figs., the particular steps performed by the licensing policy module 15, the license management facility 10 and the operating system 13 are indicated in the respective steps. Since the operations depicted in Figs. 4A-1 through 4B-2 are substantially as described above in connection with Fig. 1, they will not be described further herein.

The foregoing description has been limited to a specific embodiment of this invention. It will be apparent, however, that variations and modifications may be made to the invention, with the attainment of some or all of the advantages of the invention. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

Claims

1. A license management system for managing usage of a licensed software program comprising:
licensing storage means for storing a licensing unit value identifying a number of licensing units asso-

Petitioner Apple Inc. - Exhibit 1004, p. 5921

ciated with the licensed software program;
usage allocation value storage means for storing a usage allocation value identifying a number of licensing units associated with a use of the licensed software program; and
licensing verification means responsive to a usage request to use said licensed software program for determining, based on the contents of said licensing storage means and said usage allocation value storage means, whether usage of said licensed software program is permitted and, if usage is permitted, for adjusting the contents of said licensing storage means by a value to the contents of said usage allocation value storage means.

2. A license management system as defined in claim 1 for use in a digital data processing system which generates a system date value, said licensing storage means includes a plurality of fields including a licensing unit storage field for storing said licensing unit number identifying value and a field identifying a termination date, said licensing verification means further determining whether usage of said licensed software program is permitted in response to a comparison of said system date and said termination date.

3. A license management system as defined in claim 1 for managing usage of plurality of licensed software programs, wherein said licensing storage means includes a plurality of entries each containing a program identification field identifying a licensed software program and a licensing unit storage field for storing said licensing unit value, said licensing verification means including:
request receiving means for receiving a usage request identifying a licensed software program;
licensing unit retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of said licensing unit storage field from the entry of said licensing storage means whose program identification field identifies the licensed software program identified in said usage request; and
licensing unit processing means for determining, based on the contents of retrieved licensing unit storage field and said usage allocation value storage means, whether usage of said licensed software program is permitted and, is usage is permitted, for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

4. A license management system as defined in claim 3 for use in a digital data processing system which generates a system date value, each entry in said licensing storage means further including a termination date field identifying a termination date, said licensing unit processing means further determining whether usage of said licensed software program is permitted in response to a comparison of said system date and said termination date.

5. A license management system as defined in claim 3 wherein said usage allocation value storage means includes a plurality of usage allocation tables each storing a value identifying a number of licensing units, each entry in said licensing storage means further including a usage allocation table identification field identifying a usage allocation table, said licensing verification means further including usage allocation table retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of the usage allocation table identified by the contents of said usage allocation table identification field of said retrieved entry, said licensing unit processing means using said retrieved usage allocation table in its determination.

6. A license management system as defined in claim 5 wherein a request message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its determination.

7. A license management system as defined in claim 3 wherein a request message further includes licensing usage allocation value selection criteria and said usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation selection criterion, said licensing verification means including means for retrieving the usage allocation value associated with the licensing usage allocation selection criterion in said request message and using said retrieved usage allocation value in its determination.

8. A license management system as defined in claim 1 wherein said licensing verification means further operates in response to a release request message for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

9. A license management system as defined in claim 8 for managing usage of a plurality of licensed software programs, wherein said licensing storage means includes a plurality of entries each containing a program identification field identifying a licensed software program and a licensing unit storage field for storing said licensing unit value,

Petitioner Apple Inc. - Exhibit 1004, p. 5922

said licensing verification means including:
request receiving means for receiving a release request identifying a licensed software program;
licensing unit processing means for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

10. A license management system as defined in claim 9 wherein said usage allocation value storage means includes a plurality of usage allocation tables each storing a value identifying a number of licensing units, each entry in said licensing storage means further including a usage allocation table identification field identifying a usage allocation table, said licensing verification means further including usage allocation table retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of said usage allocation table identification field of said retrieved entry, said licensing unit processing means using retrieved usage allocation table in its adjusting.

11. A license management system as defined in claim 10 wherein a release message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its adjusting.

12. A license management system as defined in claim 8 wherein a release message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation value table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its adjusting.

13. A license management system for use in a digital data processing system including a system date generating means for generating a system date value, said license management system comprising:
licensing storage means including a plurality of entries each associated with a licensed software program, each entry containing a licensing units field for storing a licensing unit value identifying a number of licensing units associated with the license software program, a usage allocation table, and a termination date;
usage allocation table storage means for storing a plurality of usage allocation tables, each usage allocation table having a plurality of usage allocation entries each usage allocation entry being associated with a licensing usage allocation value selection criterion and storing a usage allocation value identifying a number of licensing units; and
licensing verification means including:
usage grant means including:
usage request message receiving means for receiving a usage request message from a licensed software program, said usage request message identifying said licensed software program and usage grant criteria;
entry retrieval means responsive to the receipt of a usage request message for retrieving from said licensing storage means the licensing table entry associated with said licensed software program;
usage allocation table retrieval means for retrieving from said usage allocation table storage means a usage allocation entry identified by said retrieved licensing table entry and the licensing usage allocation value selection criterion identified by the received usage request message;
licensing request processing means including:
usage determination means including licensing unit comparing means for comparing the contents of said licensing units field and said usage allocation units field and date comparison means for comparing the system date value with the contents of said termination date field to determine whether usage of said licensed software program is permitted.
response generation means for generating a message in response to the determination by said usage determination means; and
licensing unit adjusting means for adjusting the contents of said licensing units field in response to a positive determination by said usage determination means;
usage release means including:
usage release message receiving means for receiving a usage request message from a licensed software program; said usage request message identifying said licensed software program and usage grant criteria;
entry retrieval means responsive to the receipt of a usage request message for retrieving from said licensing storage means the licensing table entry associated with said licensed software program;
usage allocation table retrieval means for retrieving from said usage allocation table storage means a usage allocation entry identified by said retrieved licensing table entry and the licensing usage allocation value selection criterion identified by the

Petitioner Apple Inc. - Exhibit 1004, p. 5923

received usage request message;
licensing release processing means for adjusting the contents of said licensing units field in relation to the value of said usage allocation entry.

Petitioner Apple Inc. - Exhibit 1004, p. 5924

Nouvellement déposé

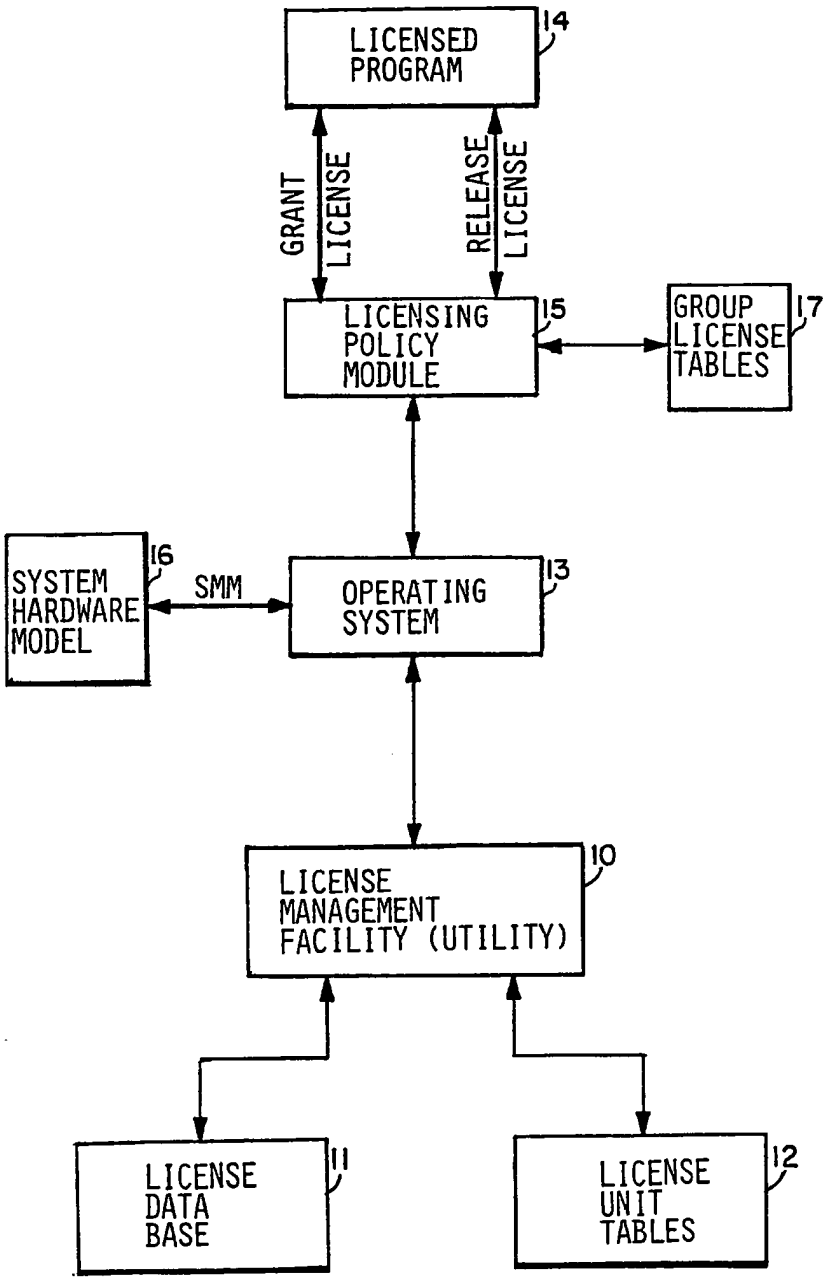


FIG. 1

zu eingereicht / Newly filed
Nouvellement déposé

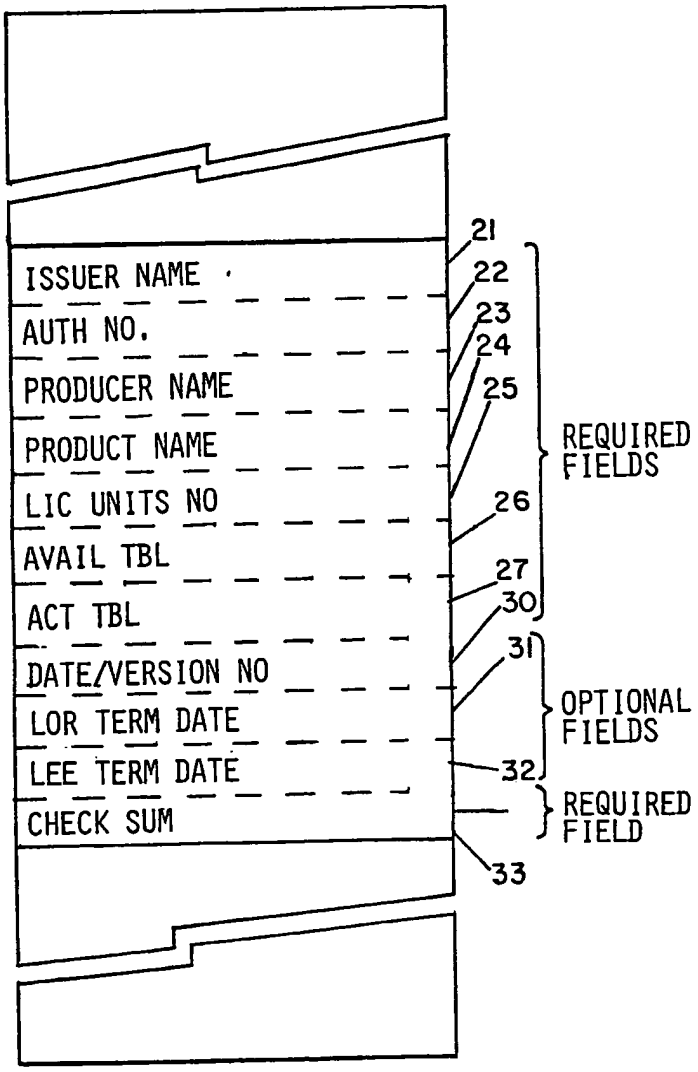
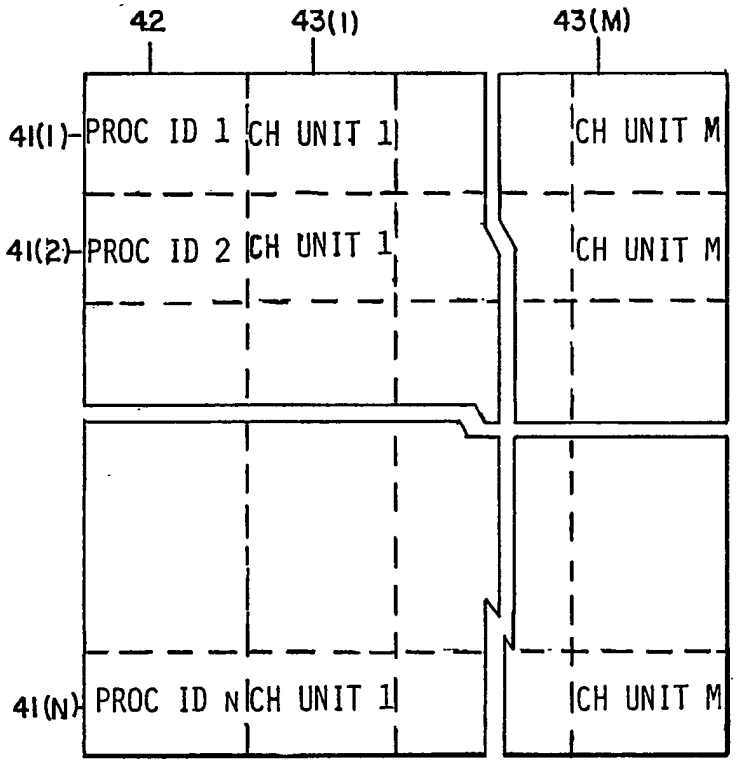


FIG. 2

eingereicht / Newly filed
Nouvellement déposé



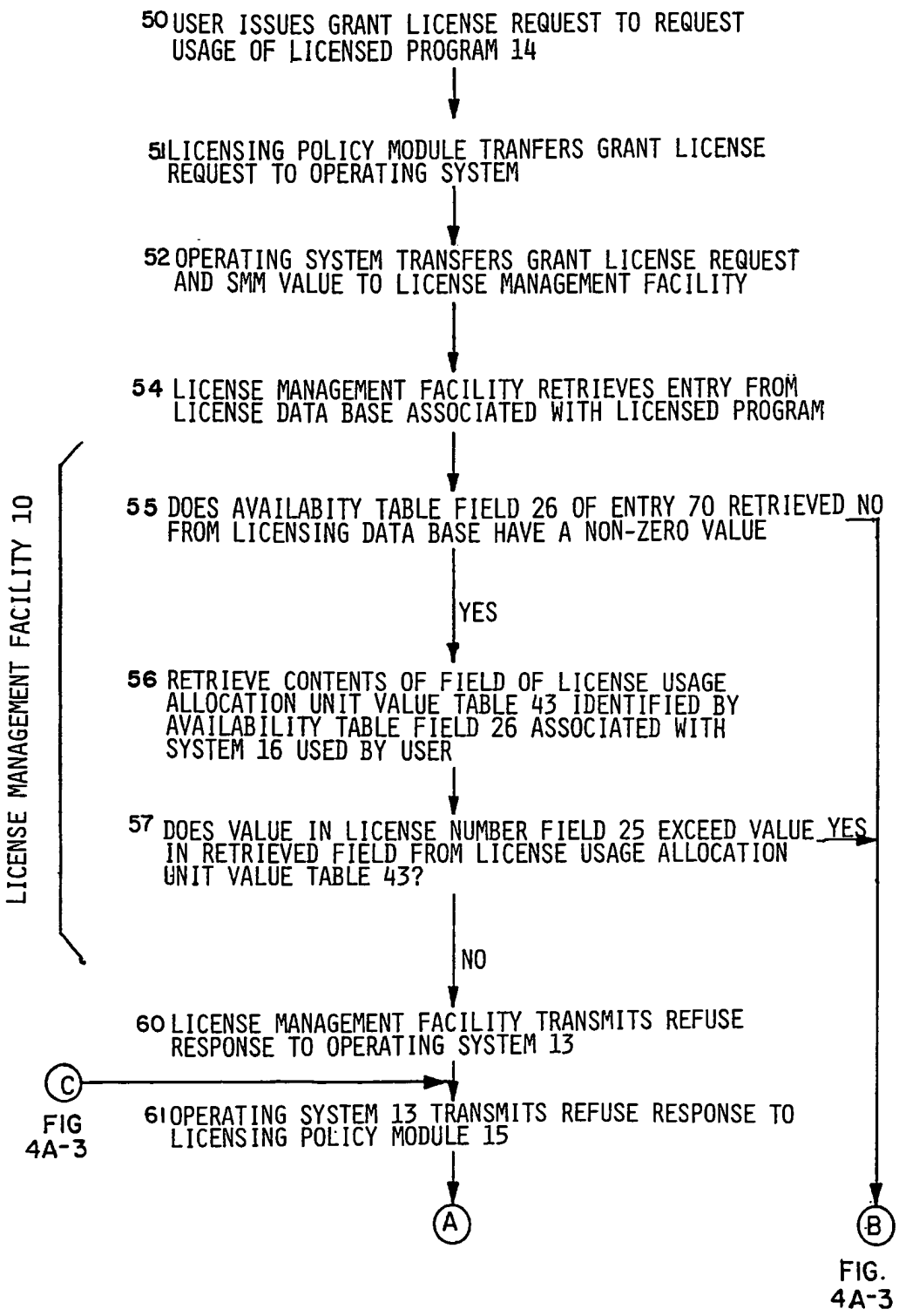
LICENSE UNIT TABLE 40

FIG. 3

Nouvellement déposé

FIG. 4A-1

GRANT LICENSE



Nouvellement déposé

FIG. 4A-2

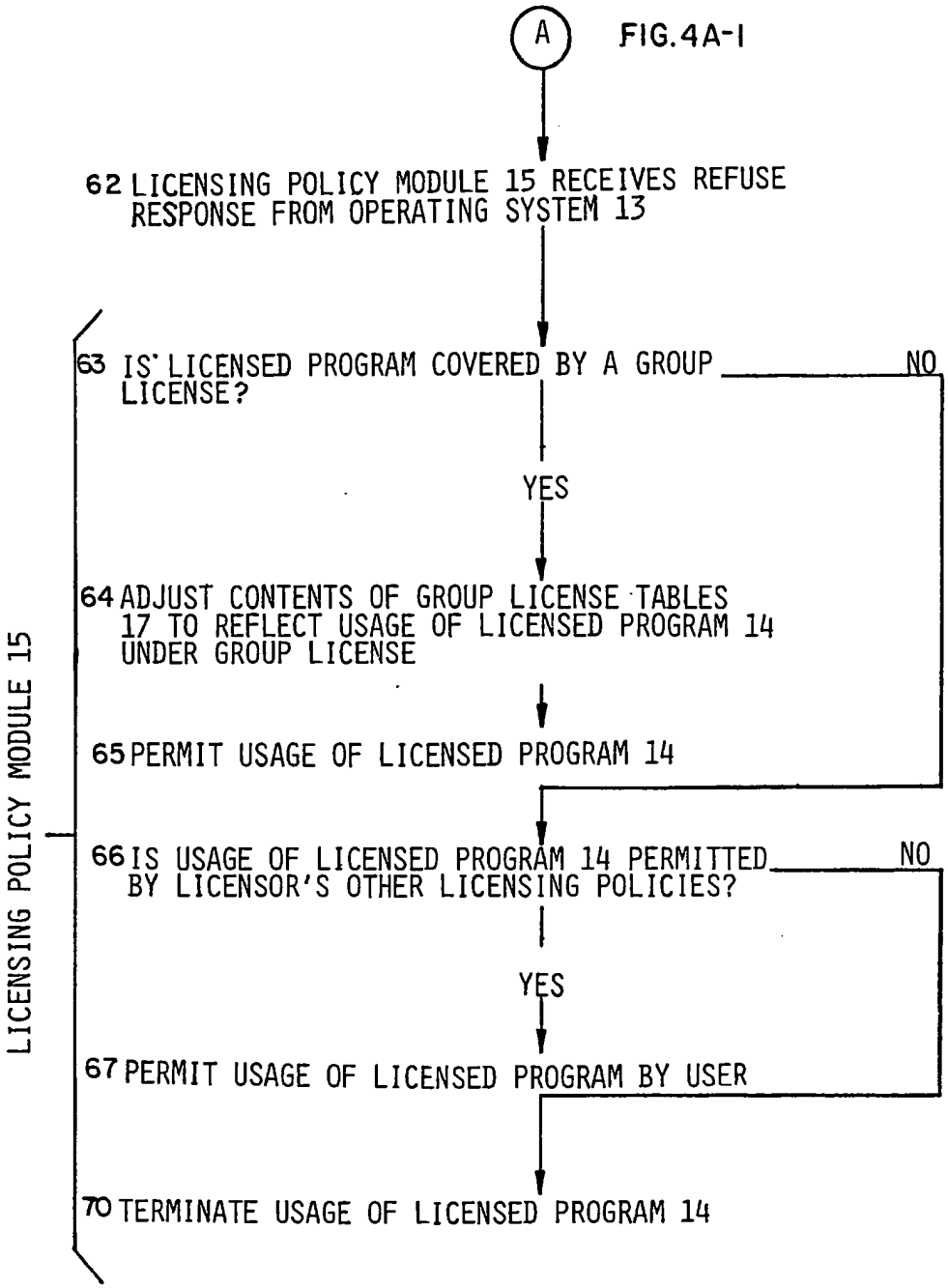


FIG. 4A-3

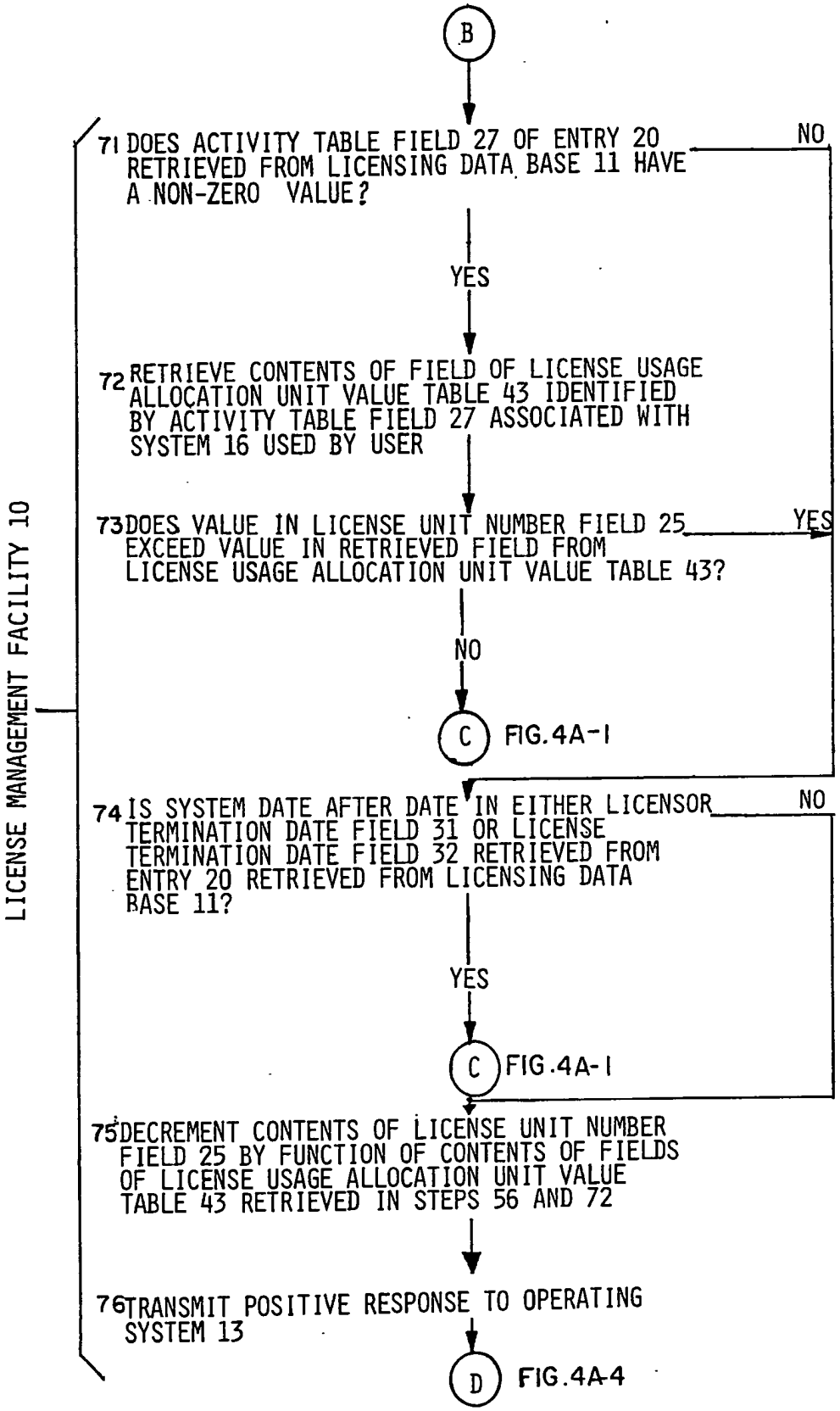

B
71 DOES ACTIVITY TABLE FIELD 27 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
YES
72 RETRIEVE CONTENTS OF FIELD OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43 IDENTIFIED BY ACTIVITY TABLE FIELD 27 ASSOCIATED WITH SYSTEM 16 USED BY USER
73 DOES VALUE IN LICENSE UNIT NUMBER FIELD 25 EXCEED VALUE IN RETRIEVED FIELD FROM LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43?
YES
NO
C FIG.4A-1
74 IS SYSTEM DATE AFTER DATE IN EITHER LICENSOR TERMINATION DATE FIELD 31 OR LICENSE TERMINATION DATE FIELD 32 RETRIEVED FROM ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11?
NO
YES
C FIG.4A-1
75 DECREMENT CONTENTS OF LICENSE UNIT NUMBER FIELD 25 BY FUNCTION OF CONTENTS OF FIELDS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43 RETRIEVED IN STEPS 56 AND 72
76 TRANSMIT POSITIVE RESPONSE TO OPERATING SYSTEM 13
D FIG.4A-4
LICENSE MANAGEMENT FACILITY 10

FIG. 4A-4

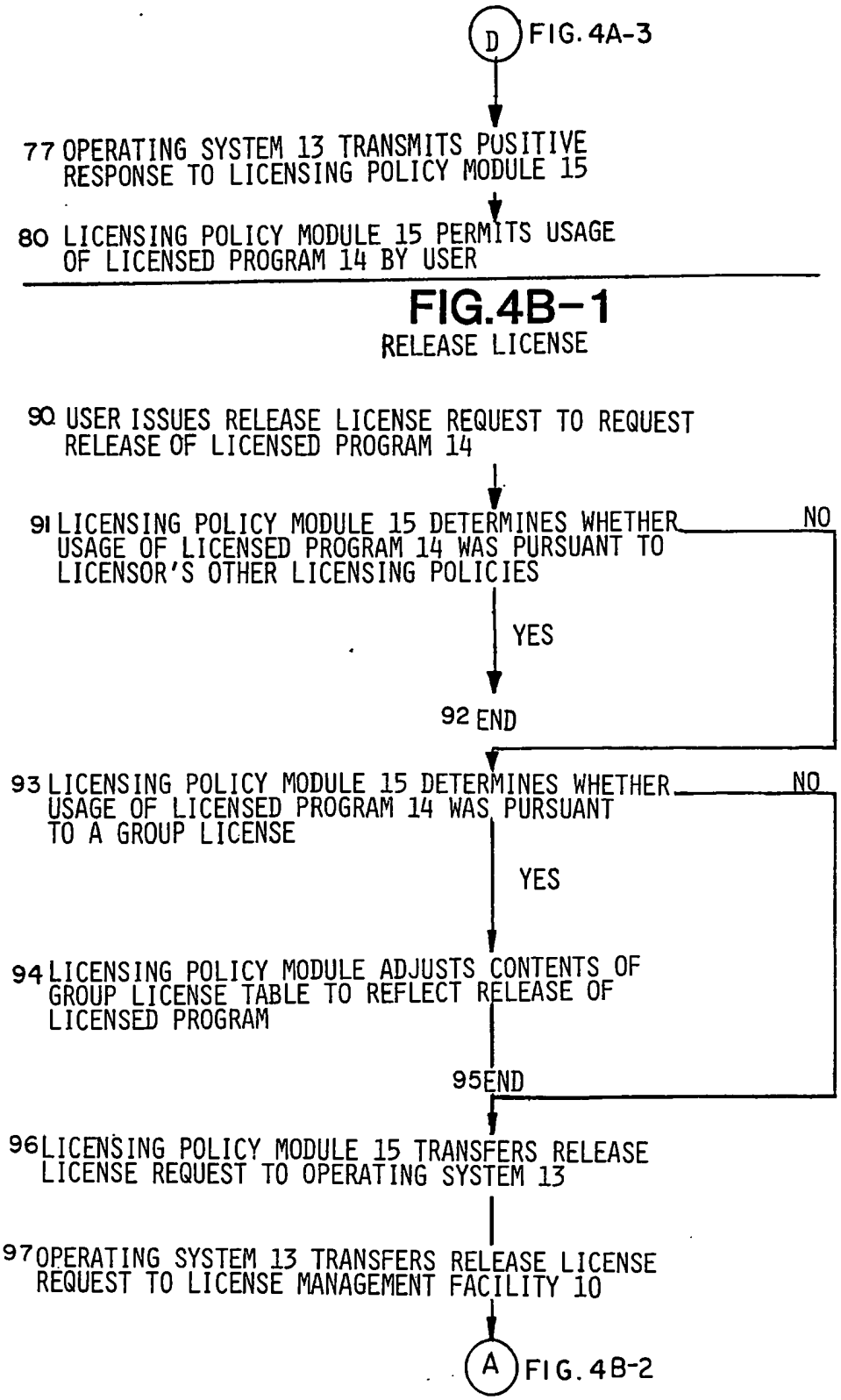
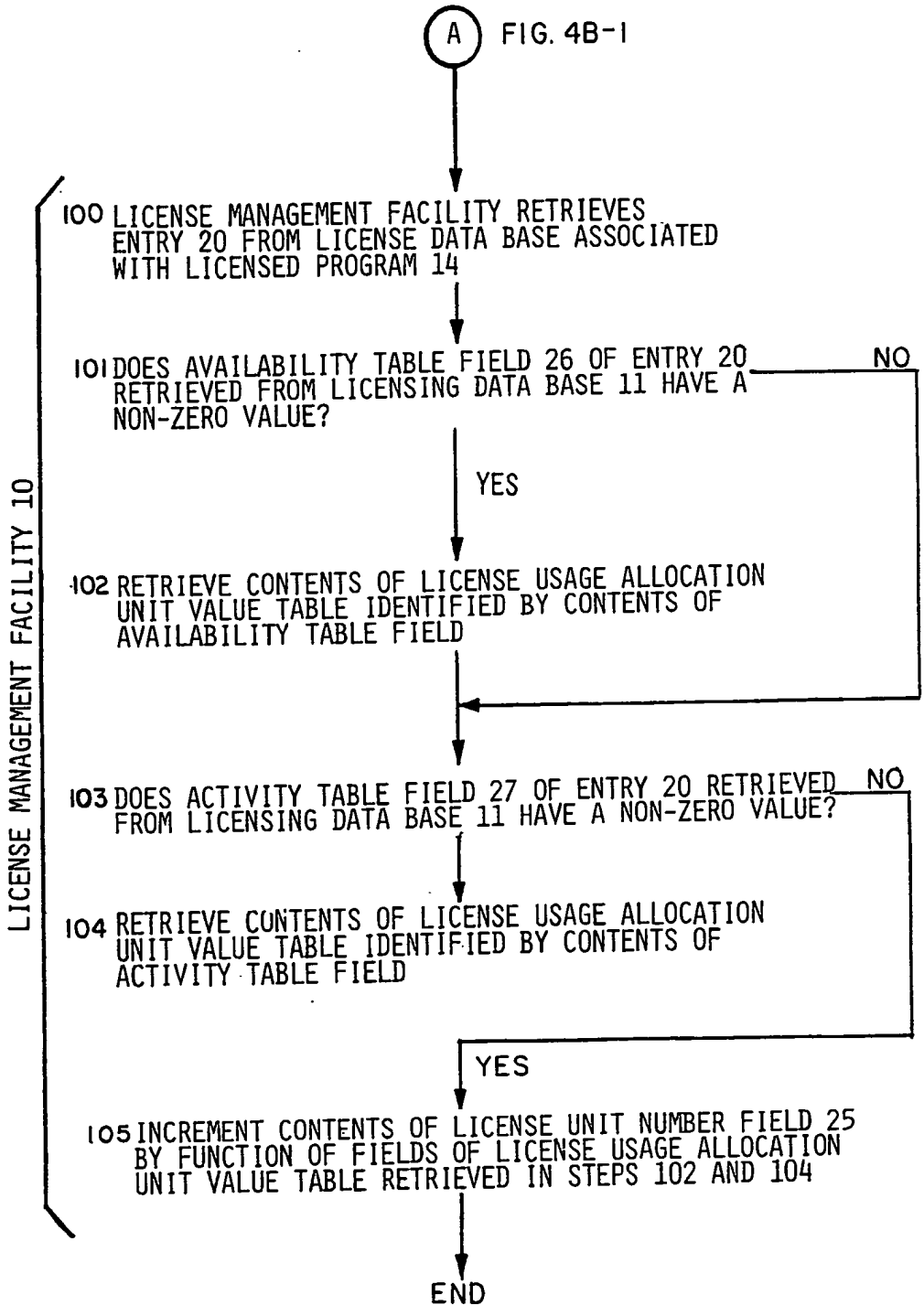


FIG.4B-1

RELEASE LICENSE

FIG. 4B-2


A
FIG. 4B-1
LICENSE MANAGEMENT FACILITY 10
100 LICENSE MANAGEMENT FACILITY RETRIEVES ENTRY 20 FROM LICENSE DATA BASE ASSOCIATED WITH LICENSED PROGRAM 14
101 DOES AVAILABILITY TABLE FIELD 26 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
YES
102 RETRIEVE CONTENTS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE IDENTIFIED BY CONTENTS OF AVAILABILITY TABLE FIELD
103 DOES ACTIVITY TABLE FIELD 27 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
104 RETRIEVE CONTENTS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE IDENTIFIED BY CONTENTS OF ACTIVITY TABLE FIELD
YES
105 INCREMENT CONTENTS OF LICENSE UNIT NUMBER FIELD 25 BY FUNCTION OF FIELDS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE RETRIEVED IN STEPS 102 AND 104
END

09

(12) UK Patent Application (19) GB (11) 2 316 503 (13) A

(43) Date of A Publication 25.02.1998

(21) Application No 9617596.3

(22) Date of Filing 22.08.1996

(51) INT CL[6]
G06F 1/00

(52) UK CL (Edition P)
G4A AAP

(56) Documents Cited
GB 2236604 A EP 0332304 A2 WO 93/11480 A1
US 5375206 A US 4924378 A

(58) Field of Search
UK CL (Edition O) G4A AAP
INT CL[6] G06F

(71) Applicant(s)
ICL Personal Systems Oy

(Incorporated in Finland)

PO Box 458, SF-00101 Helsinki, Finland

(72) Inventor(s)
Tapani Lindgren

(74) Agent and/or Address for Service
S M Dupuy
International Computers Limited, Cavendish Road,
STEVENAGE, Hertfordshire, SG1 2DY,
United Kingdom

(54) Software licence management

(57) A software licence management method and system is for a computer system including at least one server (1,5) and particularly for a plurality of computers connected via a network. Before a service (2) can offer functionality to a user it has to check that the user has a licence for that service. A licensing subsystem (3) has associated with it a ticket database (4) that hold tickets corresponding to existing licences. Tickets, if available, are issued to a service on request, thereby verifying the existence of a licence. The receipt of a ticket allows a service to offer functionality.

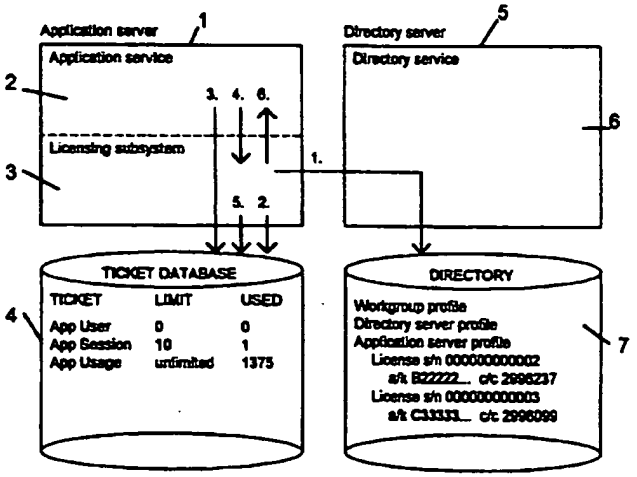


FIG 1

GB 2 316 503 A

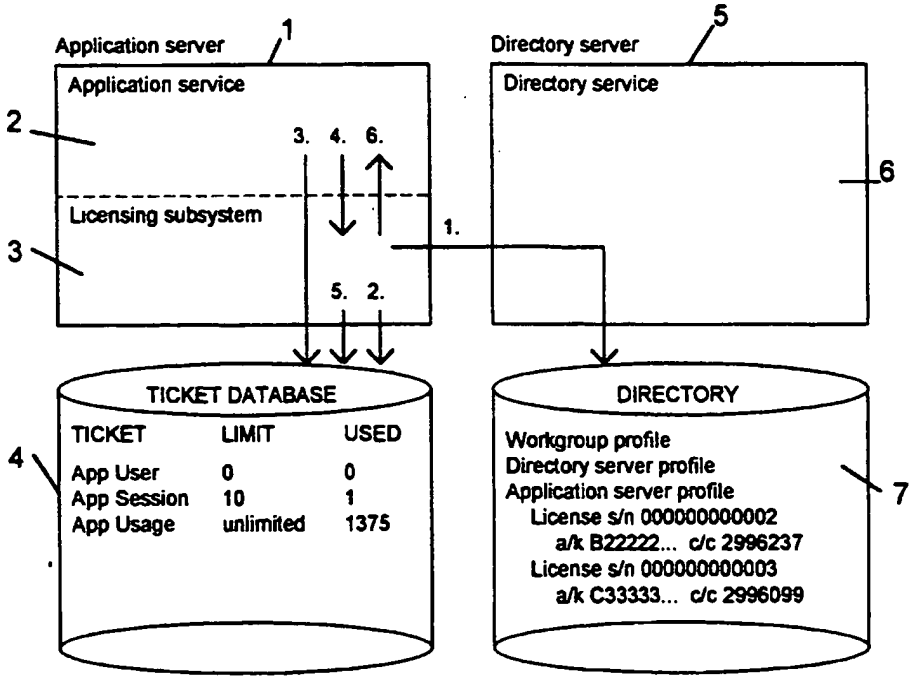

Application server
1
Application service
2
3. 4. 6.
Licensing subsystem
3
1.
5. 2.
Directory server
5
Directory service
6
TICKET DATABASE
TICKET LIMIT USED
4
App User 0 0
App Session 10 1
App Usage unlimited 1375
DIRECTORY
Workgroup profile
Directory server profile
Application server profile
License s/n 000000000002
a/k B22222... c/c 2996237
License s/n 000000000003
a/k C33333... c/c 2996099
7

FIG 1

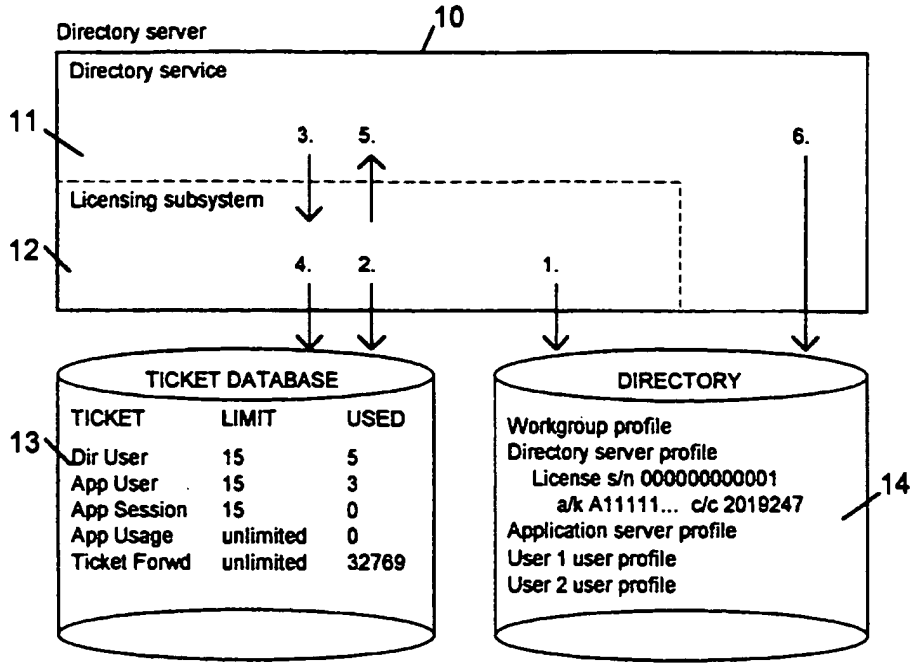

Directory server
10
Directory service
11
3.
5.
6.
Licensing subsystem
12
4.
2.
1.
TICKET DATABASE
TICKET
LIMIT
USED
13
Dir User
15
5
App User
15
3
App Session
15
0
App Usage
unlimited
0
Ticket Forwd
unlimited
32769
DIRECTORY
Workgroup profile
Directory server profile
License s/n 000000000001
a/k A11111... c/c 2019247
14
Application server profile
User 1 user profile
User 2 user profile

FIG 2

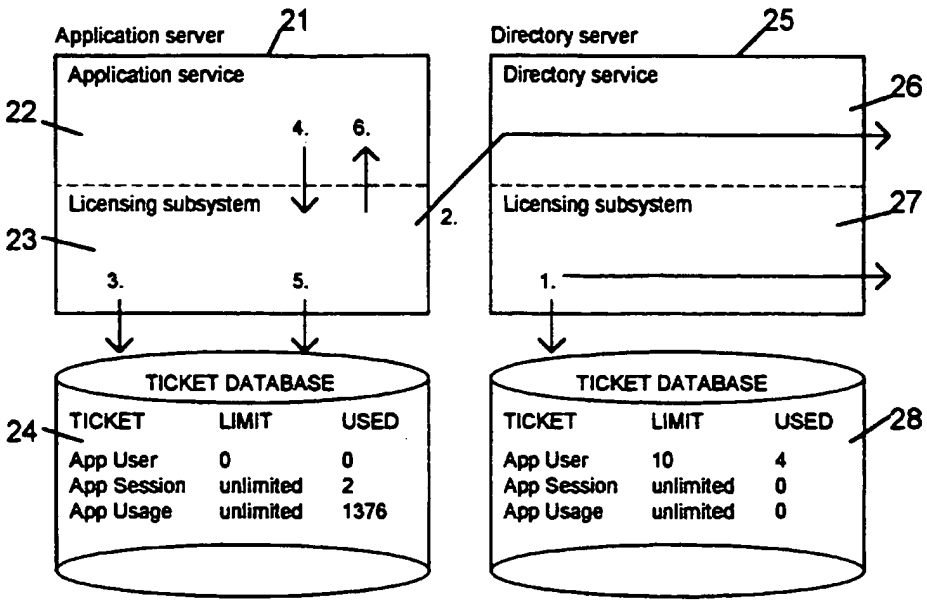

Application server
21
Application service
22
4.
6.
Licensing subsystem
23
2.
3.
5.
TICKET DATABASE
24
TICKET LIMIT USED
App User 0 0
App Session unlimited 2
App Usage unlimited 1376
Directory server
25
Directory service
26
Licensing subsystem
27
1.
TICKET DATABASE
28
TICKET LIMIT USED
App User 10 4
App Session unlimited 0
App Usage unlimited 0

FIG 3

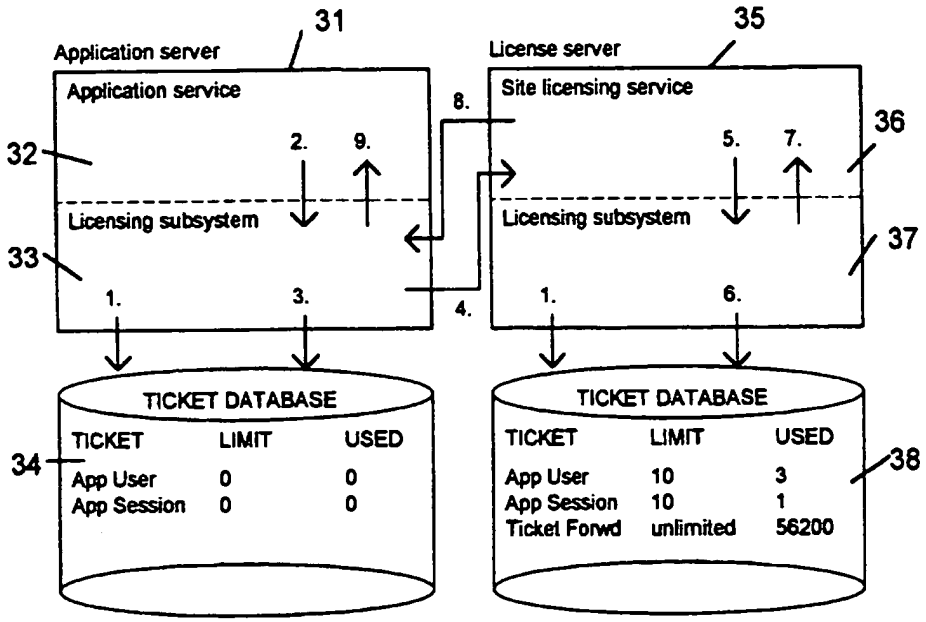

31
Application server
Application service
32
2.
9.
Licensing subsystem
33
1.
3.
TICKET DATABASE
TICKET
LIMIT
USED
34
App User
0
0
App Session
0
0
8.
4.
35
License server
Site licensing service
36
5.
7.
Licensing subsystem
37
1.
6.
TICKET DATABASE
TICKET
LIMIT
USED
38
App User
10
3
App Session
10
1
Ticket Forwd
unlimited
56200

FIG 4

2316503

SOFTWARE LICENCE MANAGEMENT

This invention relates to software licence management and in particular to licence management for software running on a plurality of computers connected via a network.

Conventionally, licences have been provided by software vendors as separate licences for individual workstations or as a single licence for a number of workstations. Various schemes have been proposed in order to try and make unlicensed software unusable, in particular pirated (illegal) copies of software. Other schemes have been proposed such as in order to achieve low initial software costs but licensing royalties consistent with the extent of use, in order not to deter low-usage users from purchasing particular forms of software, and thus to reduce piracy, whilst still enabling a vendor to collect higher dues from high-usage users.

The present invention is, particularly, concerned with a distributed system consisting of various server and client programs running on various computers which are connected via a local or wide area network, and an object is to provide server software licensing which ensures that all software running in the network has been purchased legally.

According to one aspect of the present invention there is provided a software licence management method for use with a computer system including at least one server, the method being such that before a service can offer functionality to a user, the said service shall verify that the user has a licence for said service, and wherein the computer system further includes a licensing subsystem with which are associated service tickets corresponding to existing licences, the method including the steps of the said service requesting a respective service ticket from the licensing

Petitioner Apple Inc. - Exhibit 1004, p. 5938

subsystem prior to offering functionality to the user, and the licensing subsystem issuing a said service ticket to the said service, if one is available, thereby verifying the licence exists and allowing the said service to offer functionality.

According to another aspect of the present invention there is provided a computer system including at least one server and a software licence management system, the management system being such that before a service can offer functionality to a user, the service shall verify that the user has a licence for said service, the management system including a licencing subsystem with which are associated service tickets corresponding to existing licences, and the management system being such that a said service ticket is issued to a service, if one is available, upon request by the service, thereby verifying existence of a licence and allowing the said service to offer functionality.

Embodiments of the invention will now be described with reference to the accompanying drawings, in which:

Figure 1 Illustrates obtaining a session or usage ticket for an application,

Figure 2 Illustrates obtaining a user ticket for a directory server,

Figure 3 Illustrates independent licence sharing, and

Figure 4 Illustrates licence sharing with a site licencing service,

Various terms used in the following will first be defined. For the purposes of the description the software is considered to relate to a Groupware Office system which provides various facilities including mail, for example.

Definitions

"Server"	An instance of server software running on a server computer. Usually only one such instance runs on any one computer. Each "server" implements one or more collections of related functions called "function sets", examples of which are directory, mail, library etc. The "directory function set" includes functions to access a database that contains information about the Groupware Office system.
"Client"	Any piece of software that connects to the "server" using a "client-server protocol" to access the functions offered by the "server". A "client" may be a program run by a user on a workstation, or a part of any other program.
"Session"	An instance of client-server dialogue between one "client and one "server". Each "session" allows the "client" to use the functions of one or more "function sets".
"Directory Server"	A server that implements the directory function set.
"Mail Server"	A server that implements the mail function set. [A server may be a directory server and a mail server simultaneously.]
"User"	A person (actual or virtual) listed in the database of a directory server.

Petitioner Apple Inc. - Exhibit 1004, p. 5940

"User profile"	The information pertaining to one user stored in a directory database entry, such as the name of the user, user authentication information, the list of servers and function sets the user is permitted to access, etc.
"Server profile"	The name and network address of a server and the list of services offered by it, as stored in a directory database entry.
"Service profile"	Information stored in a directory database entry about one service in one server. If the same type of service is offered by more than one server, each instance has its own profile.
"Site profile"	Information stored in a directory database entry about one site.
"Site"	A set of servers connected to a single directory server. Each server belongs to exactly one site, and each site has exactly one directory server. Other servers in the site are optional, usually unlimited in number, and sometimes called member servers or application servers.
"Enterprise"	A set of sites that share their directory databases. The directory servers in each site replicate directory information to other directory servers in the enterprise. Each directory server contains both "local" and "external" information. One of the directory servers, the "enterprise directory server" controls the others, which are

"site directory servers".

"Subsystems" — Collections of programs and/or subroutines that perform a set of interrelated functions. Some subsystems implement a function set within a server program, while others run independently as stand-alone applications. Many subsystems are collections of common subroutines called by other subsystems. Client programs are also subsystems.

"Subsystem id" — A respective unique number identifying every type of subsystem. Some systems use two ids, a "real subsystem id" when dealing with licensing issues and an "alias subsystem id" when performing a task on behalf of a virtual entity, such as "generic gateway no 9".

Not every subsystem software needs to be purchased individually. Most collections of subroutines can be used freely by other subsystems.

"Services" — Those subsystems that need to be explicitly purchased.

Service types may include directory service, mail service, fax gateway, X.400 gateway, enterprise option, library service, power library option, etc. Each service is located in one server, either as a function set of the server program or a standalone application running in the same computer. Many services of the same type can exist in different servers.

The software licence management system of the present invention proceeds from the premise that before offering any usable functionality to the users, services shall verify from a "licensing subsystem" that a licence for the service exists. To achieve that, it is proposed that the licensing subsystem holds "service tickets" and a service requests a permission to offer its functions to the user by requesting a corresponding "service ticket" be provided from the licensing subsystem. Each service knows that kind of tickets are needed to fulfil the service's functionality. The licensing subsystem has to keep track of the available licences and of the service tickets it has issued. A service ticket may be considered as partially the equivalent of a password in that one must be provided before a service can operate.

A "licence" is a permission to use one or more services within certain limits. Typically these limits are "license duration", which specifies the maximum length of the period when the licence may be used (the "active" period) and the "licence size", which specifies the maximum number of users of the licence. The interpretation of "number of users" varies from service to service. It may, for example, mean the number of users in the local directory that are allowed to use the service, or the number of concurrent sessions that are connected to the service. Licence duration and/or size may also be unlimited.

When a customer purchases a Groupware, for example, software product which employs the software licence management method and system of the invention from a supplier, as well as the media containing the software itself and associated documentation, there is obtained a single licence to one or more services. Each said product has a unique serial number. The license is supplied in the form of a licence agreement document on which the licence information is printed. This licence information consists of the serial number of the product and an "activation key" for the licence. The licence

size and duration and the included services are encoded into the activation key.

The software license management method and system of the invention is such that the Groupware software may be copied and installed by the customer without any technical restrictions, but before any of the services can be used, a corresponding licence must be installed and activated. Licence installation consists of entering the license information (serial number and activation key) into the server profile of a server in the directory server's database, ie in the site directory, in the server profile of the server in question. Licence activation consists of setting the active period of the installed licence so that service tickets can be issued. Typically licence installation and licence activation are performed simultaneously by the server setup program. The license information is stored in the site directory, in the server profile of the server in question.

As will be appreciated, there also exists "evaluation licences" which allow a prospective customer to use a service for a short trial period before actually purchasing it. These licences typically have a very short duration and a relatively small size. The product serial numbers associated with such evaluation licences are not necessarily unique, since the licence information may be distributed on CD-ROMs or via public networks.

As mentioned above, each software product contains just one licence, although that licence may include a large number of services, for example, enough to build a complete Groupware Office site with all of the basic services. Alternatively, the licence may include just one service. Product with that kind of licence could be used to expand the capacity or functionality of an existing Groupware Office System.

The core of the process of designing a product is, therefore, determining what services will be included and the size and duration of the licence. This information is encoded into a number, the covert code, which may be a 7-digit number, for example. The building of the covert code is discussed in more detail hereinafter.

The amount of information that can be encoded into the covert code is limited by the size of the code. Therefore, there are some necessary restrictions on what kinds of licences are possible. The most obvious limitation is that the size and duration can only take certain discrete values. Also, the same size and duration will apply to all services covered by the licence. Another restriction is that only the most common groups of services can be combined freely into a multi-service licence. Other services will have to be licensed individually.

The covert code, which specifies the properties of the software licence, is thus a part of the product description in the logistics database. When the product is manufactured it has the unique serial number, referred to above, assigned to it. The actuation key for the license is calculated as a function of the serial number and the covert code using a secret algorithm. The serial number and activation key may be printed on a label, which is attached to the licence agreement document.

When creating a site, a customer must have a licence that includes a site creation ticket. This licence is installed for the directory server. The customer may also install additional licences for the directory server and for other servers. Each licence may apply to one or more services. Some licences are valid only in that server for which they are installed, whilst other licences may be shared with other servers at the same site (see later). Shared licences would usually be installed in the server profile of the directory

server, although optionally, and with some restrictions, another server may be designated as the licence server. The serial number of the first licence installed in a directory server's profile can be used to identify the site uniquely. Thus the directory server is computer number 1. Other servers in the site will use the same site id but differing computer numbers for identification.

When a service program is about to execute an action which requires that a customer possesses a licence for that service, the service program must first obtain a corresponding service ticket from the licensing subsystem. The actions concerned are ones which are potentially profitable for the customer and may, for example, include namely:

setting up a new Groupware site;

creating a new user account;

setting up an instance of the mail service;

enabling mail usage for a user and creating a user mailbox;

starting a session between a mail UI client and the mail server;

sending a mail message;

relaying a mail message to an X.400 mail network.

Each kind of action requires a specific kind of service ticket. To obtain a ticket the service needs to specify the ticket type and the number of tickets. The service tickets are only identified by ticket type. There is a licensing subsystem in each server and it counts the number of

different tickets in all available licences and keeps track of how many licences of each type are being used in the server.

The steps involved in obtaining various licences will now be described in greater detail. With respect to Figure 1 there will, firstly, be described the case of an application service running in a separate server from the directory server obtaining a session or usage ticket.

Figure 1 illustrates schematically an application server 1, providing an application service 2 and having a licensing subsystem 3, with an associated ticket database 4, a directory server 5 providing a directory service 6 and having an associated directory 7. The ticket database 4 has stored therein details of ticket types, the limit, if any, of the number of such tickets which are available and the number of used tickets for each type. The ticket types as illustrated are "App User" (Application User), "App Session", "App Usage". The directory 7 has stored therein, the "Workgroup profile", the "Directory server profile", the Application server profile. In the example illustrated, the application server has two associated licenses whose serial numbers (s/n) are 000000000002 and 000000000003, respectively, whose activation keys (a/n) are of the form B22222... and C33333..., respectively, and whose covert codes (c/c) are 2996237 and 2996099, for example, respectively.

When the licensing subsystem 3 on the application server starts, it fetches the application server's server profile from the directory service 6, 7 using the directory API (Application Programming Interface) (Step 1 in Figure 1). The licensing subsystem 3 analyses the licences and updates the limits of each ticket type in the local ticket database 4. The numbers of used tickets are not modified at this time, the old accumulated values being maintained (step 2).

When the application service 2 starts, and before any user logs in, it tells the licensing subsystem 3 to set the number of used session tickets to zero. This frees any session tickets that may have been left unreturned at the end of a session because of a system crash etc. (Step 3). The application service 2 then requests a service ticket (session or usage) from the licensing subsystem 3, since without a ticket it cannot proceed. (Step 4). The licensing subsystem checks the ticket availability in the local ticket database 4 and updates the used ticket count (step 5), to take into account the requested ticket, before issuing the ticket to the application service (step 6), which then proceeds since it has determined that there exists the appropriate licence.

In the embodiment of Figure 2, the procedure whereby a directory service obtains a ticket for adding a user to a directory is illustrated.

A directory server 10 provides a directory service 11 and includes a licensing subsystem 12 with an associated ticket database 13, the directory service 11 having an associated directory 14. The ticket database 13 has stored therein details of ticket types, the limit, if any, of the number of such tickets which are available, and the number of used tickets for each type. The ticket types are illustrated as "Dir User" (Directory User), "App User", "App Session", "App Usage" and "Ticket Forwd" (Ticket Forwarding). The directory 14 has stored therein the "Workgroup profile", the "Directory server profile", the "Application Server profile" and the user profile of two users, User 1 and User 2. The Directory server has a license serial number (s/n) 000000000001, with an actuation key (a/) of the form A11111..., and a corresponding covert code (c/c) 2019247, for example.

When the licensing subsystem 12 on the directory server 10 starts, it fetches the directory server's server profile directly from the directory 14 (step 1). The licensing

subsystem 12 analyses the licenses and updates the limits of each ticket type in the ticket database 13. The numbers of used tickets are not modified at this time, the old accumulated values being maintained (step 2).

When it is desired to add a new user to the directory, the directory service 11 requests a user ticket from the licensing subsystem 12 (step 3). The licensing subsystem 12 checks ticket availability in the local ticket database 13 and updates the used ticket count (step 4) to take into account the requested ticket. The licensing subsystem 12 issues the requested user ticket to the directory service 11 (step 5). The directory service then adds the new user to the directory 14, that is it adds its user profile.

To ensure consistency, the directory service 11 may periodically count the number of users in the directory 14 and tell the licensing subsystem 12 to set the used ticket count accordingly.

When a licence is installed, the start time of its active period will be fixed. By default this is the same as the installation time, but any time in the past or in the future may be specified. If the licence has a limited period, the end time will also be set. The licence will be active whenever the current time is after the start time and before the end time.

A customer may wish to deactivate a licence so that it cannot be used. Thus can be done at any time by altering the end time of the licence with the server setup program. The end time may be altered freely, as long as the active period does not exceed the licence duration.

Once installed, limited-duration licences are fixed, ie they cannot be removed, except by remaining the entire site directory, or moved to another server, and their start time

may not be altered. These restrictions, however, do not apply to unlimited-duration licences. They may be removed, reinstalled, moved or altered freely. The only restriction that remains is that a licence may only be installed for one server at a time.

A further restriction applies to the licence that has been used to create a site. This licence cannot be removed or deactivated, except by removing the entire site.

The licensing method described with reference to Figures 1 and 2 applies only to local licences, ie the tickets included in a license can only be issued in one server, the server whose server profile contains the licence. Often there is a need to share a single licence between two or more servers, so that tickets can be issued in all of them. Most commonly, the user tickets for an application are needed in the directory server, and session and usage tickets in the application servers.

If a licence includes an unlimited number of a certain kind of service ticket, sharing the licence is not very complicated. Any server can read the licences in any other server's profile. If the licensing subsystem in a server can verify that another server's licence contains an unlimited supply of freely shareable tickets, it will deduce that these tickets may be issued without limit in any server, independently of other servers. This is independent license sharing.

Not all licences are necessarily shareable, even if they contain an unlimited number of tickets. Whether each licence is shareable or not is a licence-specific property, which is coded in the covert code together with other licence properties.

The first implementation of the licensing subsystem capable

of independent licence sharing will not scan every server profile for available licences. It will only scan its own server profile and the directory server's profile. Therefore, all licenses that are meant to be shared, should be installed for the directory server.

If the tickets to be shared are limited in number, the situation is more complicated. For each "pool" of shareable tickets, there must be a single process that is responsible for keeping track of their usage. It has to co-ordinate the activities of the licensing subsystems in various servers and make sure that no ticket is issued more than once. To achieve this a site licensing service can be implemented. This is an extension to the licensing subsystem that allows the licensing subsystems of various servers to communicate using a client-server protocol. The site licensing service, together with the licensing subsystem in the same server, control the usage of tickets installed for that server. Another server's licensing subsystem may connect to the site licensing service and ask the latter to obtain a service ticket on its behalf.

Licenses that are shareable by independent sharing would also be shareable by the site licensing service, with the addition that also limited-number tickets could be shared. Some types of licences will still be unshareable, since shareability is a licence-specific property. The licensing service could itself require a licence. A site licensing service could be expanded to support also client licensing and enterprise-wide licence sharing.

An example of independent licence sharing will now be described with reference to Figure 3 in which an application server 21 provides an application service 22 and includes a licensing subsystem 23 with an associated ticket database 24. A directory server 25 provides a directory service 26 and includes a licensing subsystem 27, with a associated ticket

database 28, and a directory (not shown but containing information of the type illustrated in Figures 1 and 2). The ticket databases 24 and 28 have details of ticket type, limit and usage as indicated.

The licensing system 27 on the directory server 25 fetches the server profile from the directory (not shown), analyses the licences therein, and updates the ticket limits (step 1). The licensing system 23 on the application server 21 fetches the application server's server profile from the directory (not shown) using the directory API. It also fetches the directory server's server profile (step 2).

The Application server's licensing subsystem 23 analyses the licences in the server's own profile. In this case there are none, since the example is concerned with licence sharing. The licensing subsystem 23 then analyses the directory server's licences. Because there are unlimited session and usage tickets in a shareable licence, the local limit is also set to unlimited. The user ticket limit is set to 0, because they are limited (10 according to ticket database 28) and limited tickets cannot be shared with this method (step 3).

The application service 22 then requests an application session ticket from its licensing subsystem 23 (step 4). The ticket is granted because there are an unlimited supply of them. The used ticket count is updated in the local ticket database 24 (step 5), although it is only needed for statistics as the number is unlimited. The session ticket is then issued to the application service 22, which then proceeds since it has determined that there exists an appropriate licence.

License sharing in the case of a site licensing service will now be described with reference to Figure 4, in which an application server 31 provides an application service 32 and includes a licensing subsystem 33 with an associated ticket

database 34. A license server 35 provides a site licensing service 36 and includes a licensing subsystem 37 with an associated ticket database 38.

The licensing subsystems 33 and 27 of the servers 31 and 35, fetch their corresponding server profiles from a directory (not shown), analyse installed licences and store the ticket limits in the local databases 34 and 38 (step 1). The application server 31 need not have any licences.

The application service 32 requests a service ticket, for example an application session ticket, from the local licensing subsystem 33 (step 2). The local licensing subsystem 33 in the application server 31 will first attempt to issue the ticket locally, but this will fail as there are no licences installed for the application server 31, as indicated by the lack of available tickets in the ticket database 34 (step 3). The licensing subsystem 33 in the application server 31 will then connect to the site licensing service 36 using the client-server protocol and request the ticket remotely (step 4). The site licensing service 36 requests the ticket from the local licensing subsystem 37 and it also request a ticket-forwarding ticket (step 5). The licensing subsystem 37 of the license server 35 checks ticket availability and updates the used ticket counts in the ticket database 38 (step 6). The tickets are issued to the site licensing service 36 (step 7) which forwards the application ticket to the client ie licensing subsystem 33 (step 8), which as a result issues the application ticket to the application service 32, allowing that to proceed (step 9).

Whenever a licensing subsystem issues a service ticket, or a ticket is returned such as because it is an unused ticket (any number can be requested) or because it is a session ticket, which are required to be returned at the end of a session, the transaction can, optionally, be logged to a log file which is separate from other log files in the system.

The information in this separate log file may be used to implement a pay-by-usage licensing scheme (delayed billing). Logging can be enabled or disabled by an administrator. Each server has its own log file and all kinds of tickets issued in the server will be logged the same way. Logging parameters for each kind of ticket could be specified for certain types of licences, although such a licence could not be shared by the independent sharing method.

The proposed licensing method allows for introducing new services while retaining compatibility with old licences. The licensing subsystems will initially support some types of licences and service tickets that are not yet connected to any particular service. New services can be assigned to these items without making any modifications to existing administration programs and the licensing subsystem. The method could be extended further by adding new license/ticket combinations to the licensing subsystem, although all existing combinations would need to be kept unchanged. This would involve updating the licensing subsystem in all servers where the new services would be used. Older subsystems would not accept the new kind of licenses not issue tickets for the new services. The licenses and tickets could be defined statically, as they are now, although there could be other possibilities.

As discussed above, the covert code specifies the licence duration, licence size and included services. An example of a covert code comprises a 7-digit decimal number, with the digits numbered from right to left, starting from zero eg in number 6543210, digit no 0 is "0", digit no 1 is "1" etc.

Licence duration may be encoded in the last digit ie digit 0, as follows:

Digit No 0	Licence Duration
"0"	10 days
"1"	1 month (31 days)
"2"	3 months (92 days)
"3"	6 months (184 days)
"4"	1 year (366 days)
"5"	2 years
"6"	3 years
"7"	Unlimited (small size)
"8"	Unlimited (medium size)
"9"	Unlimited (large size)

Licence size may be coded in the next-to-last digit, digit no 1. However, its interpretation may depend on the licence duration. Limited duration licences may be one of, for example, 30 different sizes; duration digits "7", "8" or "9" select small, medium or large licence sizes respectively.

Digit No 1	Licence size for each duration type			
	Limited	Unlimited Small	Unlimited Medium	Unlimited Large
"0"	1	1	60	400
"1"	2	2	80	500
"2"	5	5	100	600
"3"	10	10	125	800
"4"	15	15	150	1000
"5"	20	20	175	1200
"6"	30	25	200	1500
"7"	50	30	225	2000
"8"	100	40	250	3000
"9"	Unlimited	50	300	Unlimited

The services that are included in a licence may be encoded into four digits, digits no 2 to no 5, of the covert

code. These digits are called the service code. The licence may apply to one kind of service tickets only, to a group of related service tickets that are used by one service, or to a group of selected services. The service code can be chosen to represent a particular name of service, such as "basic directory service", "basic mail service", "basic calendar service", in any desired manner but it will indicate what types of tickets are included and how many licence service tickets are included for each type of service.

The digit no 6, the most significant digit, may be used to specify a particular product line. In the examples shown in the drawings the covert codes all commence with the number 2, indicating they relate to the same product line.

Any number of licences may be installed in the server profile of any server. The activation key is verified, and the covert code calculated from the serial number and the activation key at license installation time. The mapping of covert code to service ticket is, preferably, not stored in the directory, rather it is recalculated by a licensing subsystem every time it starts up. All tickets of the same type are indistinguishable. The licensing subsystems do not keep track of individual tickets issued.

Any number of identical tickets may be obtained at once by a service from the corresponding licensing subsystem, providing of course that they are available. Tickets can be returned if they are not used.

The licensing subsystem does not force services to obtain tickets rather it is the service's responsibility to offer services only to legal users and without obtaining a respective ticket, a service which requires a licence will not function.

Session tickets are associated with client-server sessions.

Unless a service wants to allow unlimited usage, it should obtain a session ticket whenever a session starts. Determining when each session starts and ends is the responsibility of the service. Session tickets may not be applicable to all services. It is important that session tickets are returned when the sessions end, otherwise they will be unusable, at least until the licencing subsystem is resynchronised. This is achieved at server start up, when there are no sessions in existence, by setting the used session ticket count to zero.

When a user is given the right to use a service, the associated user ticket should be obtained first. Because in a currently preferred embodiment, users are created and user rights given by the directory service, the licenses that include user tickets should be installed into the directory server. The directory service is the only service that requests user tickets and it is responsible for maintaining consistency of the used ticket counts. It periodically counts all users in the directory and their user rights and sets the number of tickets in use as appropriate.

Some kinds of tickets are "consumable" e.g. for sending mail messages, and these will not be returned unless, for example, the message is cancelled.

Clearly if an originally purchased licence becomes inadequate, due for example to an increased number of users, then supplemental licences can be purchased which when installed will increase the number of available tickets for a service. Additional functionality can of course also be purchased subsequently, in order to add new features to a system, and the appropriate software and licence installed in an appropriate server.

It is considered that with the above description of the licence management system and method proposed by the

invention, a software developer will have difficulty producing the corresponding code for licence management for a particular software product written in a particular language, and hence no further description is considered necessary in this respect.

CLAIMS

1. A software licence management method for use with a computer system including at least one server, the method being such that before a service can offer functionality to a user, the said service shall verify that the user has a licence for said service, and wherein the computer system further includes a licensing subsystem with which are associated service tickets corresponding to existing licences, the method including the steps of the said service requesting a respective service ticket from the licensing subsystem prior to offering functionality to the user, and the licensing subsystem issuing a said service ticket to the said service, if one is available, thereby verifying the licence exists and allowing the said service to offer functionality.

2. A method as claimed in Claim 1, including the step of installing licence information comprising a licence serial number and a licence activation key into the computer system, the activation key containing encoded details of the licensed services, and wherein the computer system calculates, from the serial number and the activation key, information including the types of service tickets associated with a particular licence, the numbers of service tickets, and the duration of the licence.

3. A method as claimed in Claim 2 wherein the licensing subsystem maintains a log of the numbers of the maximum available and issued service tickets.

4. A method as claimed in Claim 2 or Claim 3, wherein a covert code is calculated by the computer system from the serial number and activation key and wherein mapping of the covert code to service tickets is calculated by

the licencing subsystem each time it is started.

5. A method as claimed in any one of the preceding claims wherein the computer system comprises a plurality of computers connected in a network and wherein a said server comprises a directory server, providing a directory service and including a respective licencing subsystem, together with a directory database and a ticket database, wherein stored in the directory database are directory server profile details, licence details and user profile details, and wherein the ticket database includes details of service tickets available in accordance with the respective licence details and issued, and wherein adding a user to the computer system includes the steps of starting the directory server licensing subsystem, the directory server licensing subsystem fetching the directory server profile with licence details from the directory database and updating the ticket database, the requesting of a user service ticket by the directory service from the licensing subsystem, the checking of ticket availability in the ticket database by the licensing subsystem, the issuing of a ticket by the licensing subsystem to the directory service, and the adding to the directory database of the new user's profile by the directory service.

6. A method as claimed in any one of Claims 1 to 4 wherein the computer system comprises a plurality of computers connected in a network, wherein a said server comprises an application server providing an application service and including a respective licensing subsystem with a respective ticket database, and another said server comprises a directory server providing a directory service and with a respective directory database, wherein stored in the directory database are directory server profile details, application server profile details and licence details, and wherein the ticket

database includes details of service tickets available in accordance with the licence details and issued, and wherein obtaining a use ticket for the application service includes the steps of starting the application server licensing subsystem, the subsystem fetching the application server profile and licence details from the directory database and updating the ticket database accordingly, starting the application service without providing funtionality, the requesting by the application service of a user service ticket from the licensing subsystem, the checking of ticket availability in the ticket database by the licensing subsystem, and the issuing of a service ticket to the application service by the licensing subsystem, the application service then providing its functionality to a user.

7. A method as claimed in any one of Claims 1 to 4 and for independent licence sharing, wherein the computer system comprises a plurality of computers connected in a network, wherein a said server comprises an application server providing an application service and including a respective licensing subsystem with a respective ticket database, and another said server comprises a directory server providing a directory service and including a respective licensing subsystem with a respective ticket base and with a respective directory database, wherein stored in the directory database are directory server profile details, application server profile details and shareable licence details, the number of service tickets being unlimited, wherein the directory server ticket database includes details of service tickets available in accordance with the shareable licence details and issued, and wherein the application server ticket database includes details of service tickets issued, and wherein obtaining a service ticket for the application service includes the steps of the directory server licensing system fetching the server profile from the

directory database, analysing the shareable licence details and updating the corresponding ticket types and ticket limits in the directory server ticket database, the application server licensing subsystem fetching the application server and the directory server profiles and shareable licence details from the directory database and analysing them and updating the corresponding ticket types in the application server ticket database, starting the application service without providing functionality, the requesting by the application service of a service ticket from the application server licensing system, the granting of a service ticket, and the issuing of the service ticket to the application service by the application server licensing system, the application service then providing its functionality to a user.

8. A method as claimed in any one of Claims 1 to 4 and for licence sharing with site licensing, wherein the computer system comprises a plurality of computers connected in a network, wherein a said server comprises an application server providing an application service and including a respective licensing subsystem with a respective ticket database, another said server comprises a site licensing server providing a site licensing service and including a respective licensing subsystem with a respective ticket database, and a further said server comprises a directory server providing a directory service and having a directory database, wherein stored in the directory database are directory server profile details, site licensing server profile details, application server profile details and licence details, wherein the site licensing subsystem ticket database includes details of service tickets available in accordance with the licence details and issued, and wherein obtaining a service ticket for the application service when the application server has no

respective licence includes the steps of the licensing subsystems fetching their corresponding server profiles from the directory database, analysing the installed licence details and the site licensing server updating the respective ticket database, starting the application service without providing functionality, the requesting by the application service of a service ticket from the site licensing service, the requesting of a service ticket and a ticket-forwarding ticket by the site licensing service from its licensing subsystem, the checking of ticket availability and the issuing of the service and ticket-forwarding tickets to the site licensing service, the forwarding of the service ticket to the application server licensing subsystem, and the issuing of the service ticket to the application service, the application service then providing its functionality to a user.

9. A computer system including at least one server and a software licence management system, the management system being such that before a service can offer functionality to a user, the service shall verify that the user has a licence for said service, the management system including a licencing subsystem with which are associated service tickets corresponding to existing licences, and the management system being such that a said service ticket is issued to a service, if one is available, upon request by the service, thereby verifying existence of a licence and allowing the said service to offer functionality.

10. A computer system as claimed in Claim 9, wherein the management system includes means for calculating from an input licence serial number and input licence activation key, information including the types of service tickets associated with a particular licence, the numbers of service tickets and the duration of the licence, said

information being encoded in the activation key.

11. A computer system as claimed in Claim 10, and including a log in which are stored the numbers of the maximum available and issued service tickets.

12. A computer system as claimed in Claim 9 or Claim 10, and wherein the calculating means include means for calculating a covert code from the serial number and activation key, and the licensing subsystem including means for mapping the covert code into service tickets each time the licensing subsystem is started.

13. A computer system as claimed in any one of Claims 9 to 12 and comprising a plurality of computers connected in a network, wherein a said server comprises a directory server, providing a directory service and including a respective licensing subsystem, together with a directory database and a ticket database, wherein stored in the directory database are directory server profile details, licence details and user profile details, and wherein the ticket database includes details of service tickets available in accordance with respective licence details and issued.

14. A computer system as claimed in any one of Claims 9 to 12 and comprising a plurality of computers connected in a network, wherein a said server comprises an application server providing an application service and including a respective licensing subsystem with a respective ticket database, and another server comprises a directory server providing a directory service with a respective directory database, wherein stored in the directory database are directory server profile details, application server profile details and licence details, and wherein the ticket database includes details of service tickets available in accordance with the licence

details and issued.

15. A computer system as claimed in Claim 14 and for independent licence sharing, wherein the directory server includes a respective directory licensing subsystem and a respective ticket database, shareable licence details, for which the number of service tickets available is unlimited, being stored in the directory database, the directory ticket database including details of service tickets available in accordance with the shareable licence details and issued, and the application server ticket database including details of service tickets issued.

16. A computer system as claimed in Claim 14 and for licence sharing with site licensing, and including another said server comprising a site licensing server providing a site licensing service and including a respective licensing subsystem with a respective ticket database, the directory database also including site licensing server profile details, and wherein the site licensing ticket database includes details of service tickets available in accordance with the licence detailed and issued.

17. A software licence management method substantially as herein described with reference to an as illustrated in Figure 1, Figure 2, Figure 3, or Figure 4, of the accompanying drawings.

18. A computer system including at least one server and a software licence management system substantially as herein described with reference to and as illustrated in Figure 1, or Figure 2, or Figure 3, or Figure 4 of the accompanying drawings.

The Patent Office

29

Application No: GB 9617596.3
Claims searched: 1-18

Examiner: Mike Davis
Date of search: 26 September 1996

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.O): G4A (AAP)

Int Cl (Ed.6): G06F

Other:

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	GB 2236604 A	(SUN MICROSYSTEMS)	1,9 at least
X	EP 0332304 A2	(DIGITAL EQUIPMENT)	•
X	WO 93/11480 A1	(INTERGRAPH)	•
X	US 5375206	(HUNTER ET AL)	•
X	US 4924378	(HERSHEY ET AL)	•

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Petitioner Apple Inc. - Exhibit 1004, p. 5966

(12) UK Patent Application (19) GB (11) 2 309 364 (13) A

(43) Date of A Publication 23.07.1997

(21) Application No 9700921.1

(22) Date of Filing 17.01.1997

(30) Priority Data
(31) 08588848 (32) 19.01.1996 (33) US

(51) INT CL[6]
H04L 9/30

(52) UK CL (Edition O)
H4P PDCSC
U1S S2204 S2208 S2209

(56) Documents Cited
EP 0328232 A2 WO 95/23468 A1

(58) Field of Search
UK CL (Edition O) H4P PDCSA PDCSC
INT CL[6] H04L 9/30 9/32
Online:WPI,INSPEC

(71) Applicant(s)
Northern Telecom Limited

(Incorporated in Canada - Quebec)

World Trade Center Of Montreal,
380 St Antoine Street West, 8th Floor, Montreal,
Quebec H2Y 3V4, Canada

(72) Inventor(s)
David Allan
Liam Casey
Adrian Jones

(74) Agent and/or Address for Service
M C Dennis
Nortel Patents, London Road, HARLOW, Essex,
CM17 9NA, United Kingdom

(54) Public/private key encryption/decryption

(57) In a hybrid fiber-coax distribution network, communications between a central station and particular end stations are encrypted using a working key (WK) of a symmetric encryption scheme. The central station has a public and private key (PPK) of a PPK encryption scheme, and some of the end stations can also each have a respective PPK. To provide secure communications for each end station, if the end station has a PPK, then the respective WK is generated in the central station and communicated, encrypted using the end station's public key (PK), to the end station. Otherwise, the WK is generated in the end station and communicated, encrypted using the central station's PK, to the central station. An individual identifier for each end station, and a cryptographic signature at least for end stations not having a PPK, can be communicated to the central station for authentication of the end stations.

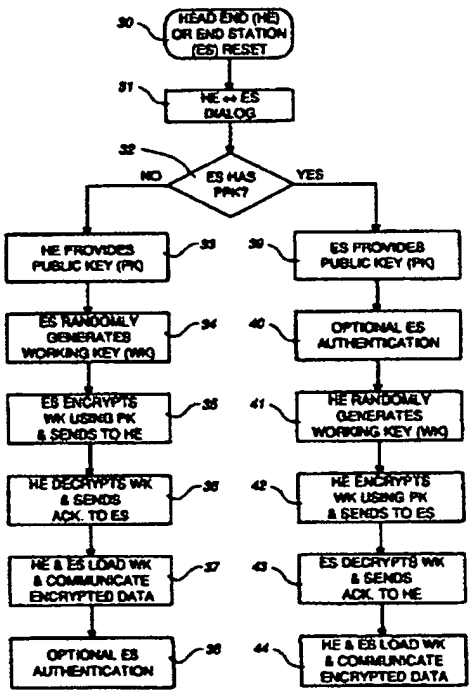


Fig. 2

GB 2 309 364 A

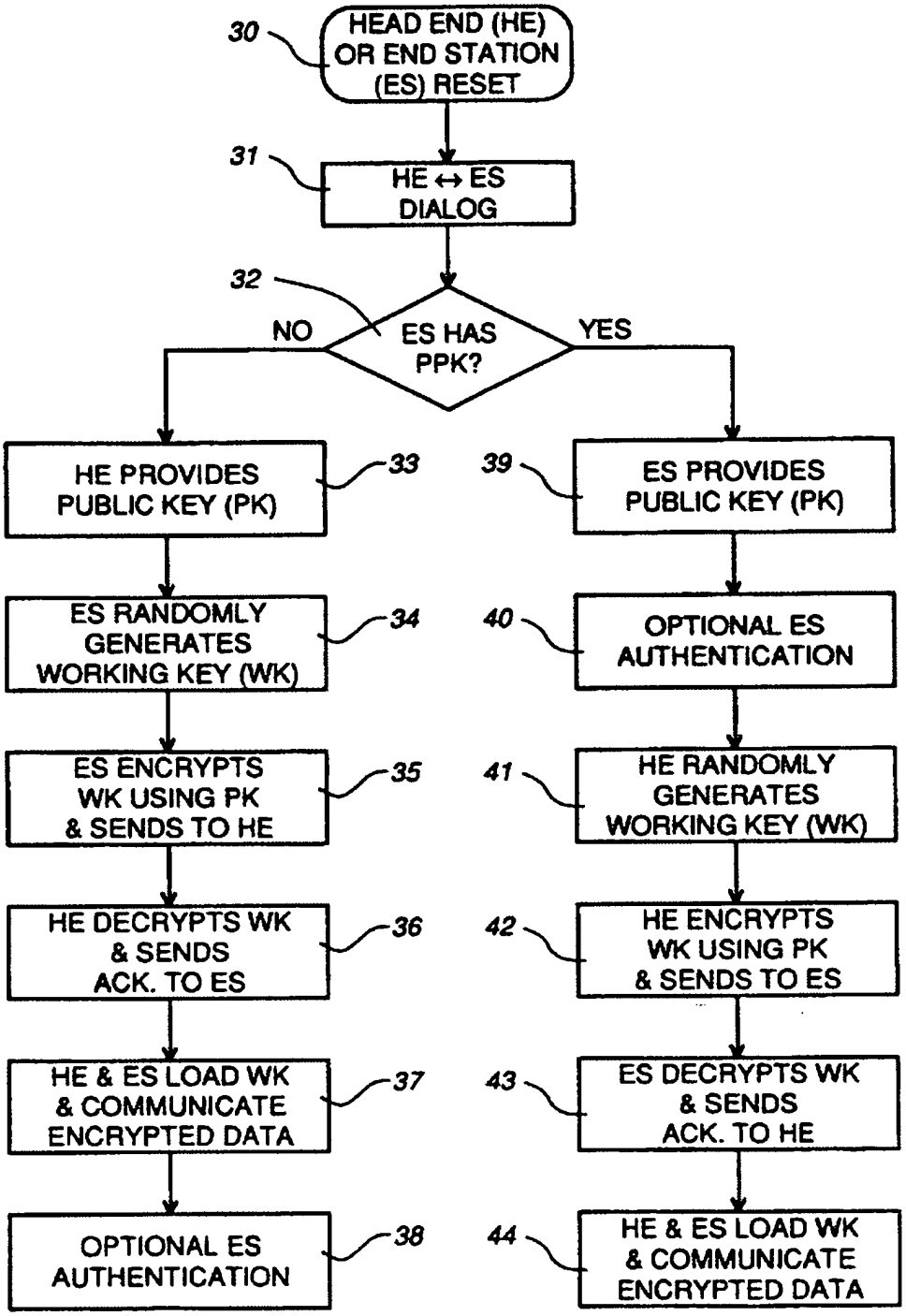

30
HEAD END (HE) OR END STATION (ES) RESET
31
HE ↔ ES DIALOG
32
ES HAS PPK?
NO
YES
HE PROVIDES PUBLIC KEY (PK)
33
ES RANDOMLY GENERATES WORKING KEY (WK)
34
ES ENCRYPTS WK USING PK & SENDS TO HE
35
HE DECRYPTS WK & SENDS ACK. TO ES
36
HE & ES LOAD WK & COMMUNICATE ENCRYPTED DATA
37
OPTIONAL ES AUTHENTICATION
38
39
ES PROVIDES PUBLIC KEY (PK)
40
OPTIONAL ES AUTHENTICATION
41
HE RANDOMLY GENERATES WORKING KEY (WK)
42
HE ENCRYPTS WK USING PK & SENDS TO ES
43
ES DECRYPTS WK & SENDS ACK. TO HE
44
HE & ES LOAD WK & COMMUNICATE ENCRYPTED DATA

Fig. 2

FACILITATING SECURE COMMUNICATIONS IN A DISTRIBUTION NETWORK

This invention relates to methods of facilitating secure communications in a distribution network, such as for example a coaxial cable or hybrid fiber-coax (HFC) network.

Background of the Invention

A distribution network, such as an HFC network in which data is communicated to subscriber end stations via optical fiber and coaxial distribution cables, is a point-to-multipoint network in which data addressed to and intended for any particular subscriber is also inevitably supplied via the network to other subscribers. If the data is not scrambled or encrypted, it can be easily monitored by these other subscribers, leading to a loss of subscriber privacy and a loss of revenues for data suppliers when the data (e.g. television programs) is supplied for a fee. Accordingly, it is important to provide a desired level of security in the data communications in a distribution network.

While various encryption and decryption schemes are known, these have a number of disadvantages associated with them in the environment of a distribution network. A significant factor in this respect is the cost and security of subscriber end stations. As a distribution network will contain large numbers of subscriber end stations, it is commercially necessary that the cost of each end station be kept relatively low. It is therefore desirable to avoid incorporating expensive security schemes in the subscriber end stations. However, subscriber end stations are also easily subject to theft, tampering, and duplication, so that complicated schemes have been considered necessary to provide adequate security.

For example, a security scheme can be implemented using an encryption key which can be stored in the subscriber end station. To prevent access to the encryption key, the store in the subscriber end station, and data lines to and from this store, must also be made physically secure. This leads to extra complexity and costs. Different subscribers may have differing security and privacy needs, which makes it desirable for the network to accommodate differing security schemes and end station costs.

A further security-related desirable aspect of a distribution network is an ability for authentication of subscriber end stations, typically using a unique end station identity which can be physically incorporated (e.g. hard wired) into the end station during manufacture.

Encryption schemes can be divided into those involving public and private keys (PPK) and those involving symmetric keys. In PPK schemes, a first station can distribute its public key, in accordance with which a second station can encrypt data and send the encrypted data to the first station, which decrypts the data using its private key. Because the private key is retained at the first station, and is not practically discoverable

by other parties, PPK schemes are considered to be secure. However, the encryption and decryption processes are relatively slow, so that such schemes are not practical for encryption of real-time high-speed data, such as television program signals, for which distribution networks are primarily intended.

In symmetric key schemes, a single key, referred to as a working key, is used by both of first and second stations to encrypt and decrypt data being communicated between the stations. The nature of the working key is such that encryption of real-time high-speed data, such as television program signals, is practical. However, these schemes require that the working key be present in both stations, and make it desirable for the working key to be periodically changed or updated. Thus symmetric key schemes require generation of a working key in one of the stations or in a third station referred to as a key distribution agent, and communication of the working key to the other station(s).

This communication itself presents a risk of the working key being insecure, and this risk increases with the frequency with which the working key is updated. It is also known to avoid this risk by using a PPK scheme for communication of a working key, and then to use the working key for data encryption.

An object of this invention is to provide a method of facilitating secure communications in a distribution network.

Summary of the Invention

One aspect of this invention provides a method of facilitating secure communications using encryption and decryption processes in a distribution network comprising a central station and a plurality of addressable end stations, in which communications from the central station addressed to and intended for a particular end station are delivered via the network to a plurality of end stations, wherein the central station has, and one or more of the end stations can each have, a respective public and private key (PPK) of a PPK encryption scheme, comprising the steps of:

(a) determining in communications between the central station and an end station whether the end station has a PPK, if so proceeding with step (b) and if not proceeding with step (c);

(b) at the central station, determining the public key (PK) of the end station, generating a working key (WK) for encryption of communications to the end station, encrypting the WK using the PK of the end station, and communicating the encrypted WK to the end station; at the end station, decrypting the WK using the private key of the end station; and proceeding with step (d);

(c) at the end station, determining the public key (PK) of the central station, generating a working key (WK) for encryption of communications to the central station, encrypting the WK using the PK of the central station, and communicating the encrypted WK to the central station; at the central station, decrypting the WK using the private key of the central

station; and proceeding with step (d);

(d) using the WK to encrypt at the central station, and to decrypt at the end station, communications from the central station to the end station.

Another aspect of this invention provides a method of facilitating secure communications in a distribution network comprising a central station and a plurality of addressable end stations, in which communications from the central station addressed to and intended for a particular end station are delivered via the network to a plurality of end stations, wherein the central station has a public and private key (PPK) of a PPK encryption scheme and each end station has an individual identity (ID) and an individual cryptographic signature encrypted using a private key of a predetermined PPK encryption scheme, comprising the steps of: communicating the ID of an end station to the central station; at the end station, generating a working key (WK) for encryption of communications between the end station and the central station and encrypting the WK using the public key of the central station; communicating the encrypted WK from the end station to the central station; at the central station, decrypting the encrypted WK using the private key of the central station; communicating the cryptographic signature of the end station to the central station; and at the central station, decrypting the cryptographic signature using a public key of the predetermined PPK scheme for authentication of the end station.

Brief Description of the Drawings

The invention will be further understood from the following description with reference to the accompanying drawings, in which:

Fig. 1 illustrates parts of a distribution network to which the invention is applied; and

Fig. 2 is a flow chart illustrating steps of a method for facilitating secure communications in the network in accordance with the invention.

Detailed Description

The invention is described below in the context of a hybrid fiber-coax (HFC) distribution network in which signals are distributed from a central station or head end (HE) to a large number of subscriber end stations (ES) via optical fibers and coaxial cables in known manner. An example of such a network is described in Warwick United States Patent No. 5,408,259 issued April 18, 1995 and entitled "Data Modulation Arrangement For Selectively Distributing Data". Typically in such a network digital data communications are provided between any ES and the HE using asynchronous transfer mode (ATM) cells which are communicated in both directions, i.e. downstream from the HE to the ES and upstream from the ES to the HE, using suitable modulation schemes and carrier frequencies outside the bands used for analog television signals also carried on

Petitioner Apple Inc. - Exhibit 1004, p. 5971

the coaxial cables. However, it is observed that the invention is equally applicable to other forms of distribution network.

Referring to Fig. 1, there is illustrated parts of a distribution network in which many end stations, only two of which are shown and are referenced 10 and 12, are connected via branched cables 14 of the distribution network to a head end 16, via which the end stations have access to a network (not shown) which for example supplies digital television program signals subscribed to by end station subscribers. The cables 14 can comprise both optical fiber and coaxial cables forming a hybrid fiber-coax arrangement, on which the digital signals can be communicated in known manner using ATM cells.

As can be appreciated from the illustration in Fig. 1, signals communicated by the head end 16 and intended for any particular end station will actually be delivered via the cables 14 to all of the end stations. For secure and/or private communication of the signals, the head end 16 includes an encryption engine 18 which encrypts the signals in accordance with a working key known only by the head end and the intended end station, which also includes an encryption engine 20 which decrypts the signals for use. These working keys are similarly used for communications in the opposite direction, from the end station to the head end 14. The working keys of this symmetric key encryption scheme are provided in the head end and the end station in a manner which is described in detail below.

The end stations 10 and 12 are of two types, with differing levels of security to enable different security needs of subscribers to be accommodated. The end station 12 represents a relatively secure end station, which includes its own public and private keys of a PPK encryption scheme. As explained in the introduction, such an end station has a relatively high complexity and cost, because of the need for secure storage of the keys and operation of the PPK encryption. Other end stations, which do not have their own public and private keys and accordingly can be provided at a much lower cost, are represented by the end station 10. The network as a whole may have an arbitrary mix of these two types of end station.

Each end station 10 or 12 also has an individual, unique identity number, which is stored (e.g. hard wired) into the ES during its manufacture. This is referred to as a global ID (identity). The global IDs of all of the end stations are stored in a database 22, which can be colocated with the head end 16 or separately from it and with which the head end 16 communicates via a path 24. The head end 16 also has its own public and private keys of a PPK encryption scheme.

Fig. 2 shows steps of a process which is followed in order to set up secure communications between the head end 16 and one of the end stations 10 or 12. This process takes place between the head end and the respective end station without involvement of any other node such as a central key distribution agent, and is described

below as being initiated in each case following any reset (e.g. following a power-up) of either the head end 16 or the respective end station. Consequently, the working key which is used for encrypting the communications between the head end and the end station is changed on any reset. However, the same process can alternatively or additionally be carried out on demand, and/or periodically to provide periodic changes of the working key. It is also observed that the encrypted communications take place between the encryption engines 18 in the head end 16 and 20 in the respective end station 10 or 12, and communications on the network access side of the head end 16 are not subject to the same encryption.

In Fig. 2, a block 30 represents a reset of the head end (HE) or end station (ES), in response to which, as shown by a block 31 in Fig. 2, a dialog or handshake is carried out between the HE and the ES to establish communications between them. These communications are effected using unencrypted ATM cells using addresses of the end station and the head end. As a part of this dialog, as shown by a block 32 in Fig. 2 the head end 16 interrogates the end station to determine whether or not the end station has its own public and private keys. If not, i.e. if the end station is an end station 10 as described above, then the process continues with successive blocks 33 to 38 in Fig. 2. If the interrogation establishes that the end station is an end station 12 having its own public and private keys, then the process instead continues with blocks 39 to 44 in Fig. 2.

In the former case of an end station 10, as shown by the block 33 the head end 16 communicates its public key (PK) to the end station 10; this communication can form part of the dialog block 31. The end station 10 randomly generates (block 34) a working key (WK) for communicating signals in a symmetric key encryption scheme, and encrypts (block 35) this working key in accordance with the supplied public key, sending the encrypted working key in a message to the head end 16. The head end 16 decrypts (block 36) the encrypted working key from this message in accordance with its private key, which is not known to others so that the communication of the working key from the end station 10 to the head end 16 is secure, and optionally but preferably sends an acknowledgement to the end station 10. As shown by the block 37, the head end 16 and the end station 10 then load their encryption engines 18 and 20 respectively with the working key, and thereafter (until this process is repeated, for example in response to a subsequent reset at either end) communications between them take place with data encrypted in accordance with the working key. An optional additional step represented by the block 38 provides for authentication of the end station 10 in a manner described below.

Conversely, in the latter case of an end station 12, as shown by the block 39 the end station 12 communicates its public key (PK) to the head end 16; this communication can form part of the dialog block 31. An optional authentication step for the end station

12 can be carried out by the head end 16 as represented by the block 40 in a manner described below. The head end 16 randomly generates (block 41) a working key (WK) for communicating signals in a symmetric key encryption scheme, and encrypts (block 42) this working key in accordance with the supplied public key of the end station 12, sending the encrypted working key in a message to the end station 12. The end station 12 decrypts (block 43) the encrypted working key from this message in accordance with its private key, which is not known to others so that the communication of the working key from the head end 16 to the end station 12 is secure, and optionally but preferably sends an acknowledgement to the head end 18. As shown by the block 44, the head end 16 and the end station 12 then load their encryption engines 18 and 20 respectively with the working key, and thereafter (until this process is repeated, for example in response to a subsequent reset at either end) communications between them take place with data encrypted in accordance with the working key.

It can be seen from the above description that, in the relatively secure but more expensive situation in which the end station 12 includes its own public and private keys, these are used for communicating a working key generated in the head end, whereas in the other case the end station 10 generates the working key and this is communicated to the head end using the latter's public key.

The optional step of authentication of the end station 12 in the block 40 as described above can make use of the global ID of the end station 12 together with data in the database 22, in which the public key of the end station 12 is stored in association with this global ID. As part of the dialog block 31, the end station communicates its global ID to the head end 16. In the step 40, therefore, the head end 16 can communicate via the path 24 with the database 22 to confirm that the public key which it has received from the end station 12 in the step 39 matches that stored in the database 22 for this end station's global ID, the subsequent steps 41 to 44 only being followed if this authentication step is successful.

Alternatively, or in addition, the optional end station authentication step of block 40 can comprise the steps of the head end sending an unencrypted message to the end station 12 with a request that it be cryptographically signed. In accordance with this request, the end station 12 produces a digest of the message using a known hashing function (thereby reducing the data to be encrypted), encrypts this digest in accordance with its private key, and sends the encrypted message digest to the head end 16. The head end 16 then decrypts this in accordance with the public key of the end station, retrieved from the database 22, to confirm the digest of its original message which the head end also produces using the hashing function.

It can be seen that, alternatively, the steps represented by the blocks 39 and 40 in Fig. 2 could be replaced by a single step in which the head end 16 determines the public

key of the end station 12 from the database 22 in accordance with the global ID of the end station 12 supplied in the dialog 31, without any authentication of the end station or any communication of the public key from the end station 12.

The above sequences provide a particularly strong or secure authentication of the end station 12. For the end station 10 which does not have its own public and private keys, a weaker but still valuable authentication can be provided as shown by the block 38. The authentication block 38 is shown in Fig. 2 as the final block in the process because this enables the exchange of data in the authentication process to be encrypted in accordance with the working key, but this authentication step could alternatively be provided anywhere else in the sequence of steps from the blocks 31 to 37.

For this optional authentication step, the end station 10 is manufactured (e.g. hard wired) with not only its global ID, but also a cryptographic signature. Conveniently, the end station 10 is manufactured with a certificate comprising data including the global ID of the end station and the public key of the manufacturer and a cryptographic signature comprising an encryption, in accordance with the private key of the manufacturer, of a digest of that data produced using a known hashing function. The public key of the manufacturer can also or instead be stored in the database 22. The optional end station authentication step of the block 38 comprises a communication of the cryptographic signature from the end station 10 to the head end 16 (as explained above this could be a part of the dialog 31 or any later step, but the encryption after the block 37 obstructs public observation in the network of cryptographic signatures). The head end 16 then confirms the authenticity of the end station 10 by decrypting the cryptographic signature using the manufacturer's public key, producing a digest from the same data (global ID and public key, both of which can be communicated in the dialog step 31 or later) and the known hashing function, and matching these.

This is a relatively weak authentication, in that identical copies of the end station 10, including duplicated data and cryptographic signatures, could operate at different times on the network without this being detected. However, simultaneous operation of two or more such duplicates would be detected by the fact that two or more end stations would be supplying the same global ID which is supposedly unique. Thus even such a weak authentication is valuable especially in detecting illicit large-scale duplication of end stations.

The processes in accordance with the invention as described above provide a number of significant advantages over known configurations. In particular, requirements for secure storage of public and private keys are minimized in the network as a whole, and eliminated for the end stations 10 which can accordingly be provided at relatively lower cost. At the same time, end stations 12 with greater security can be provided, and the head end 16 can operate simultaneously with both types of end station. This, combined

with optional authentication of the end stations as described above, enables different degrees of security to be easily provided in the network in accordance with service requirements.

Furthermore, renewal of the working keys at reset is simpler than providing time-based schedules for changing encryption keys, and key exchanges take place only between the head end and the end station which use the keys, thereby enhancing security compared with distribution of keys from a key distribution agent. In addition, all of the data flowing between the head end and any particular end station 10 or 12, between successive resets, can be encrypted using a single working key, thereby simplifying the encryption and decryption processes. However, it is observed that different working keys could be generated, communicated, and used in the same manner as described above for encrypting and decrypting different types of information, or different services, for a single end station 10 or 12.

Although particular embodiments of the invention have been described in detail, it should be appreciated that numerous modifications, variations, and adaptations may be made without departing from the scope of the invention as defined in the claims.

WHAT IS CLAIMED IS:

1. A method of facilitating secure communications using encryption and decryption processes in a distribution network comprising a central station and a plurality of addressable end stations, in which communications from the central station addressed to and intended for a particular end station are delivered via the network to a plurality of end stations, wherein the central station has, and one or more of the end stations can each have, a respective public and private key (PPK) of a PPK encryption scheme, comprising the steps of:

(a) determining in communications between the central station and an end station whether the end station has a PPK, if so proceeding with step (b) and if not proceeding with step (c);

(b) at the central station, determining the public key (PK) of the end station, generating a working key (WK) for encryption of communications to the end station, encrypting the WK using the PK of the end station, and communicating the encrypted WK to the end station; at the end station, decrypting the WK using the private key of the end station; and proceeding with step (d);

(c) at the end station, determining the public key (PK) of the central station, generating a working key (WK) for encryption of communications to the central station, encrypting the WK using the PK of the central station, and communicating the encrypted WK to the central station; at the central station, decrypting the WK using the private key of the central station; and proceeding with step (d);

(d) using the WK to encrypt at the central station, and to decrypt at the end station, communications from the central station to the end station.

2. A method as claimed in claim 1 wherein each end station has an individual identity (ID) and step (a) includes the step of communicating the ID of the end station to the central station.

3. A method as claimed in claim 2 wherein in step (b) the PK of the end station is determined by the central station from a database using the ID of the end station.

4. A method as claimed in claim 1, 2, or 3 wherein step (b) further comprises an end station authentication step comprising the steps of communicating an unencrypted message from the central station to the end station, producing an encrypted message at the end station using the private key of the end station, communicating the encrypted message to the central station, decrypting the message at the central station using the PK of the end station, and comparing the decrypted message with the original message.

5. A method as claimed in claim 4 wherein in step (b) the end station authentication step is carried out before the step of communicating the encrypted WK to the end station.

6. A method as claimed in any of claims 1 to 5 wherein in step (b) the PK of the end station is communicated to the central station from the end station.

7. A method as claimed in claims 2 and 6 wherein in step (b) the PK of the end station is verified by the central station from a database using the ID of the end station.

8. A method as claimed in any of claims 1 to 7 wherein a plurality of end stations which do not have a PPK each have an individual cryptographic signature encrypted using a private key of a predetermined PPK scheme, step (a) or (c) includes the step of communicating the cryptographic signature of the end station to the central station, and step (c) further comprises an end station authentication step comprising, at the central station, decrypting the cryptographic signature using a public key of the predetermined PPK scheme.

9. A method as claimed in claims 2 and 8 wherein the individual cryptographic signature comprises an encryption of data derived from the ID of the respective end station.

10. A method as claimed in claim 8 or 9 wherein the predetermined PPK scheme uses a private key and a public key of a source of the end station.

11. A method as claimed in claim 8, 9, or 10 wherein the cryptographic signature is communicated to the central station in step (c).

12. A method as claimed in claim 11 and including the steps of encrypting the cryptographic signature at the end station, and decrypting the encrypted cryptographic signature at the central station, using the WK.

13. A method as claimed in any of claims 1 to 12 and further comprising the step of using the WK to encrypt at the end station, and to decrypt at the central station, communications from the end station to the central station.

14. A method of facilitating secure communications in a distribution network comprising a central station and a plurality of addressable end stations, in which communications from the central station addressed to and intended for a particular end station are delivered via the network to a plurality of end stations, wherein the central station has a public and private key (PPK) of a PPK encryption scheme and each end station has an individual identity (ID) and an individual cryptographic signature encrypted using a private key of a predetermined PPK encryption scheme, comprising the steps of:

communicating the ID of an end station to the central station;

at the end station, generating a working key (WK) for encryption of communications between the end station and the central station and encrypting the WK

using the public key of the central station;

communicating the encrypted WK from the end station to the central station;

at the central station, decrypting the encrypted WK using the private key of the central station;

communicating the cryptographic signature of the end station to the central station; and

at the central station, decrypting the cryptographic signature using a public key of the predetermined PPK scheme for authentication of the end station.

15. A method as claimed in claim 14 wherein the individual cryptographic signature comprises an encryption of data derived from the ID of the respective end station.

16. A method as claimed in claim 14 or 15 wherein the predetermined PPK scheme uses a private key and a public key of a source of the end station.

17. A method as claimed in claim 14, 15, or 16 wherein the step of communicating the cryptographic signature of the end station to the central station comprises the steps of encrypting the cryptographic signature at the end station using the WK, communicating the encrypted cryptographic signature from the end station to the central station, and decrypting the encrypted cryptographic signature at the central station using the WK.

18. A method of facilitating secure communications in a distribution network, substantially as hereinbefore described with reference to Figs 1 and 2 of the accompanying drawings.

The Patent Office

12

Application No: GB 9700921.1 **Examiner:** Mr B J Spear
Claims searched: 1-13 **Date of search:** 19 March 1997

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.O): H4P (PDCSC)

Int Cl (Ed.6): H04L 9/30

Other: Online: WPI, INSPEC

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
	NONE	

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Petitioner Apple Inc. - Exhibit 1004, p. 5980

The Patent Office

Application No:	GB 9700921.1	**Examiner:**	Mr B J Spear
Claims searched:	14-17	**Date of search:**	21 May 1997

Patents Act 1977
Further Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.O): H4P (PDCSA)

Int Cl (Ed.6): H04L 9/32

Other: Online: WPI, INSPEC

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
A	EP0328232A2 (Fischer)	-
A	WO 95/23468A1 (Merdan)	-

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Petitioner Apple Inc. - Exhibit 1004, p. 5981

(12) UK Patent Application (19) GB (11) 2 236 604 (13) A

(43) Date of A publication 10.04.1991

(21) Application No 9009655.3

(22) Date of filing 30.04.1990

(30) Priority data
(31) 415984 (32) 02.10.1989 (33) US

(51) INT CL[5]
G06F 1/00

(52) UK CL (Edition K)
G4A AAP

(56) Documents cited
EP 0002390 A1 WO 88/02202 A1

(58) Field of search
UK CL (Edition K) G4A AAP
INT CL[5] G06F 1/00 12/14
Online database: WPI

(71) Applicant
Sun Microsystems Inc

(Incorporated in the USA - Delaware)

2550 Garcia Avenue, Mountain View, California 94043, United States of America

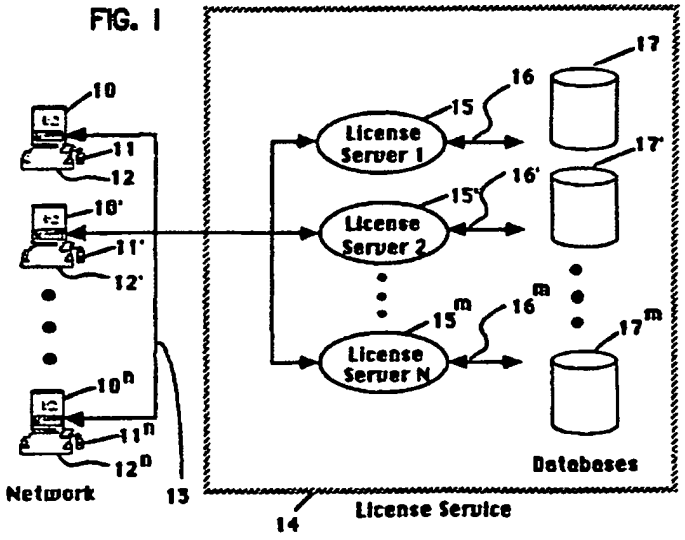
(72) Inventor
John Richard Corbin

(74) Agent and/or Address for Service
Potts Kerr and Co
15 Hamilton Square, Birkenhead, Merseyside, L41 6BR, United Kingdom

(54) **Protecting against the unauthorised use of software in a computer network**

(57) The present invention provides to a software application the verification and licence check out functions which are normally performed by a licence server. The encrypted licence information is contained in a licence token, and is stored in a database 17 controlled by the licence server 15. In contrast to the prior art where the server either grants or denies the request after verifying the user's credentials, the server in the preferred embodiment of the present invention finds the correct licence token for the software application and transmits the token to a licencing library. A licence access module attached to the application decodes the token. Routines in the licencing library coupled to the software application verify the licence information before issuing the licence and updating the token. The access module then encodes the updated token before returning it to the server. Because the verification and issuing function of a token are performed by a software application, the application rather than the server becomes the point of attack by unauthorised users. Reverse engineering the access module is less rewarding than attacking the server because the module reveals the contents of a small fraction of a database of licences.

FIG. 1



At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

GB 2 236 604 A

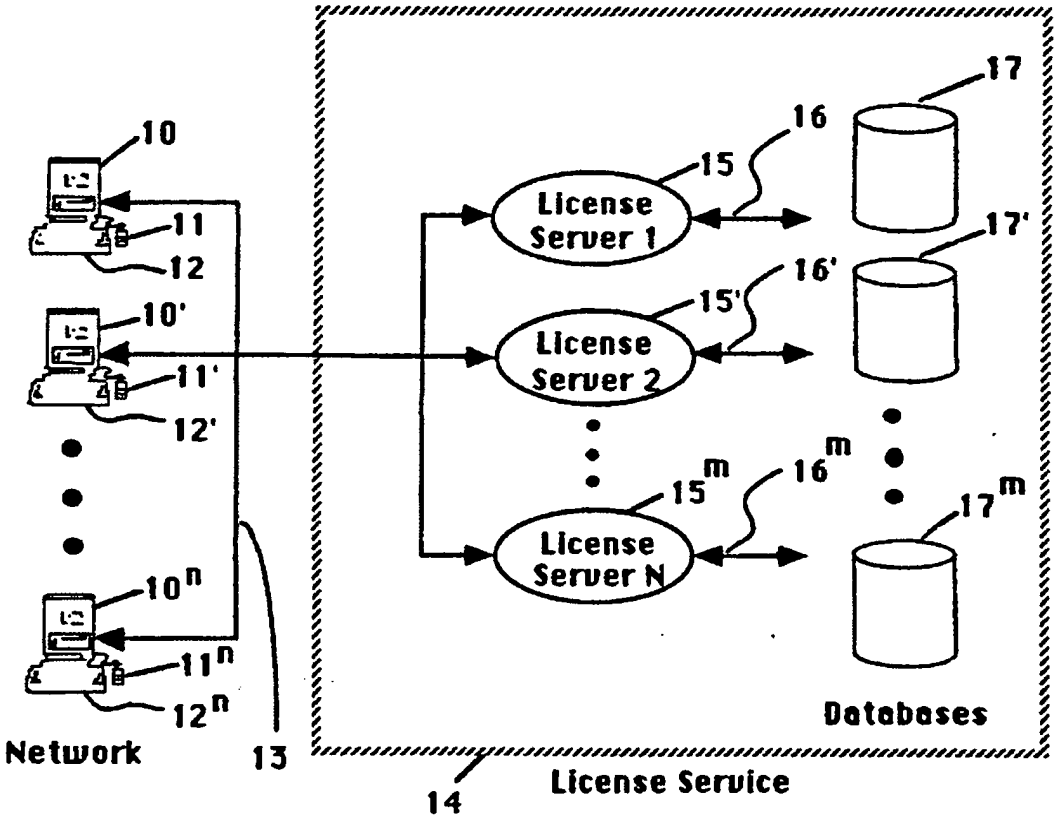

10
11
12
10'
11'
12'
10^n
11^n
12^n
Network
13
14
License Service
License Server 1
License Server 2
License Server N
15
15'
15^m
16
16'
16^m
17
17'
17^m
Databases

FIG. 1

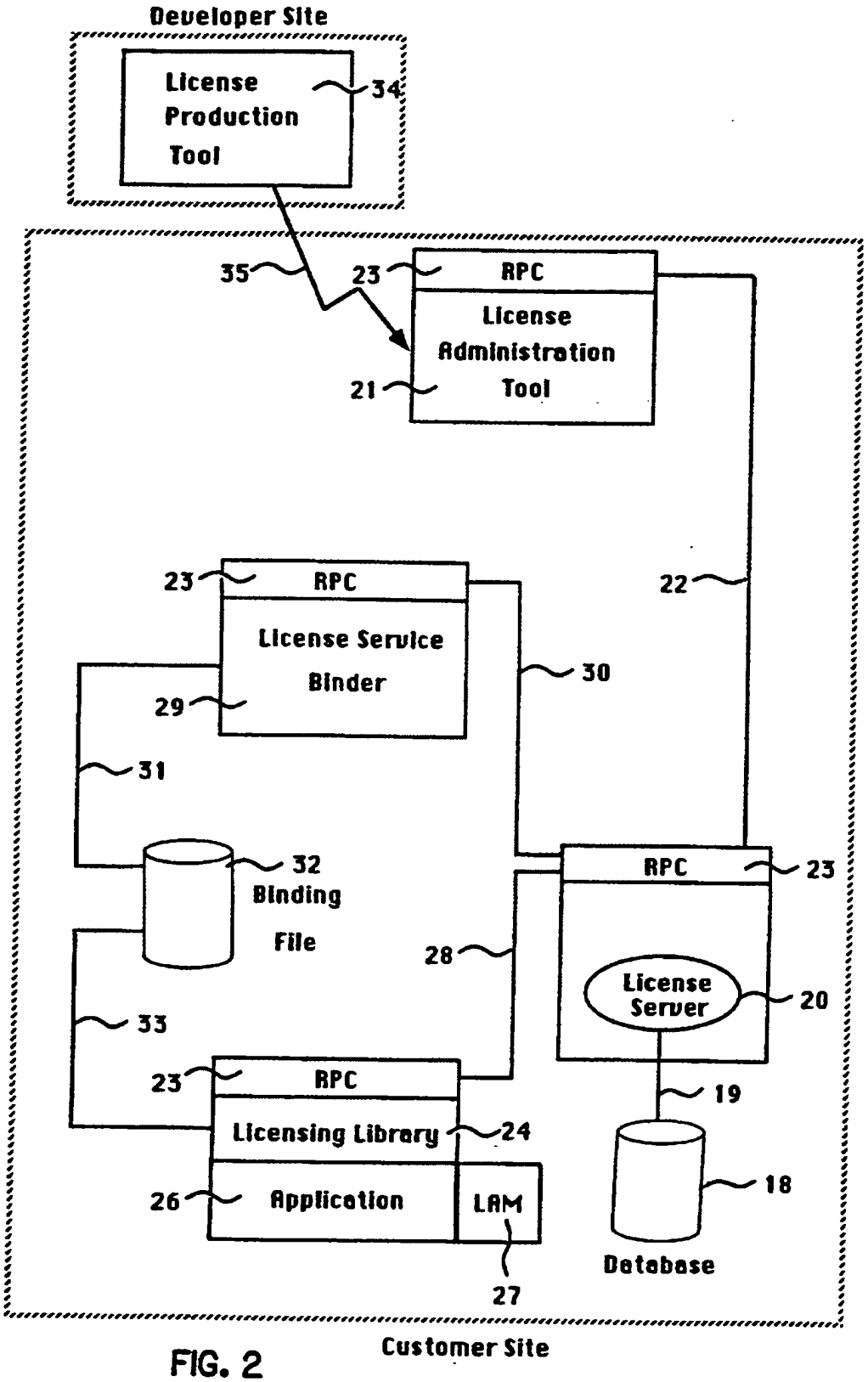

Developer Site
License Production Tool
34
35
23
RPC
License Administration Tool
21
22
23
RPC
License Service Binder
29
30
31
32
Binding File
RPC
23
28
License Server
20
33
23
RPC
Licensing Library
24
19
26
Application
LAM
27
18
Database
Customer Site

FIG. 2

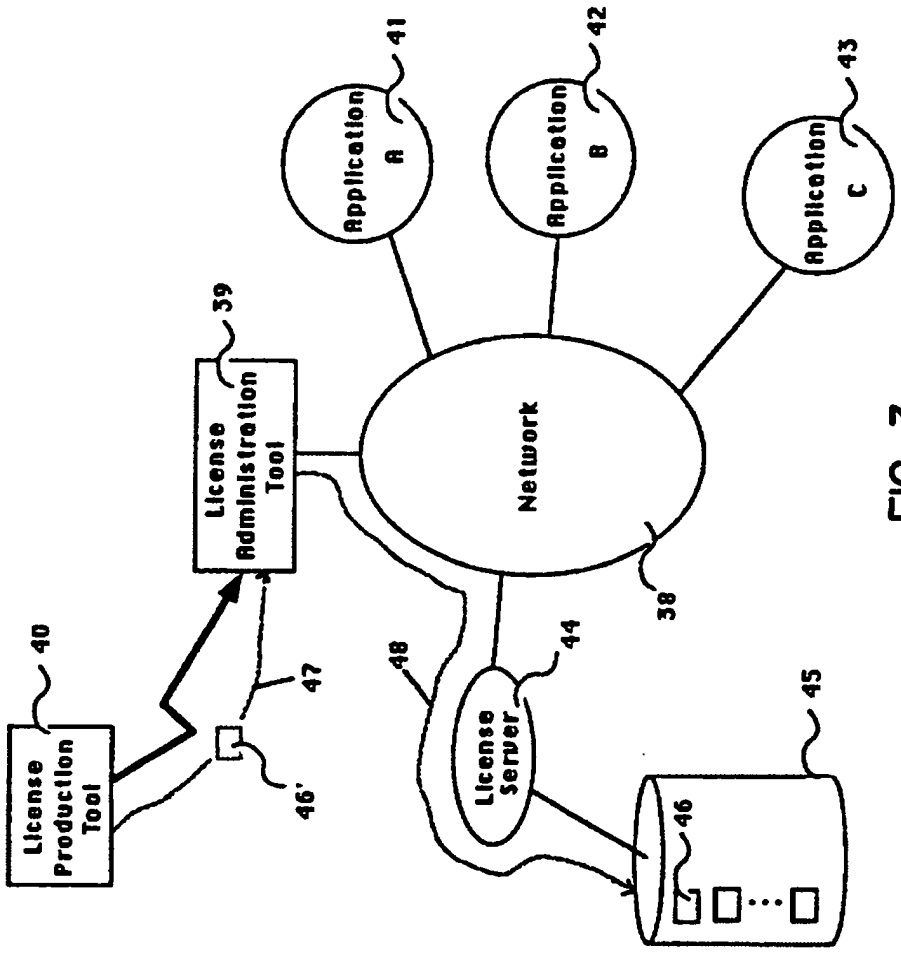

Application A
41
Application B
42
Application C
43
License Administration Tool
39
Network
38
License Production Tool
40
47
46'
48
License Server
44
45
46

FIG. 3

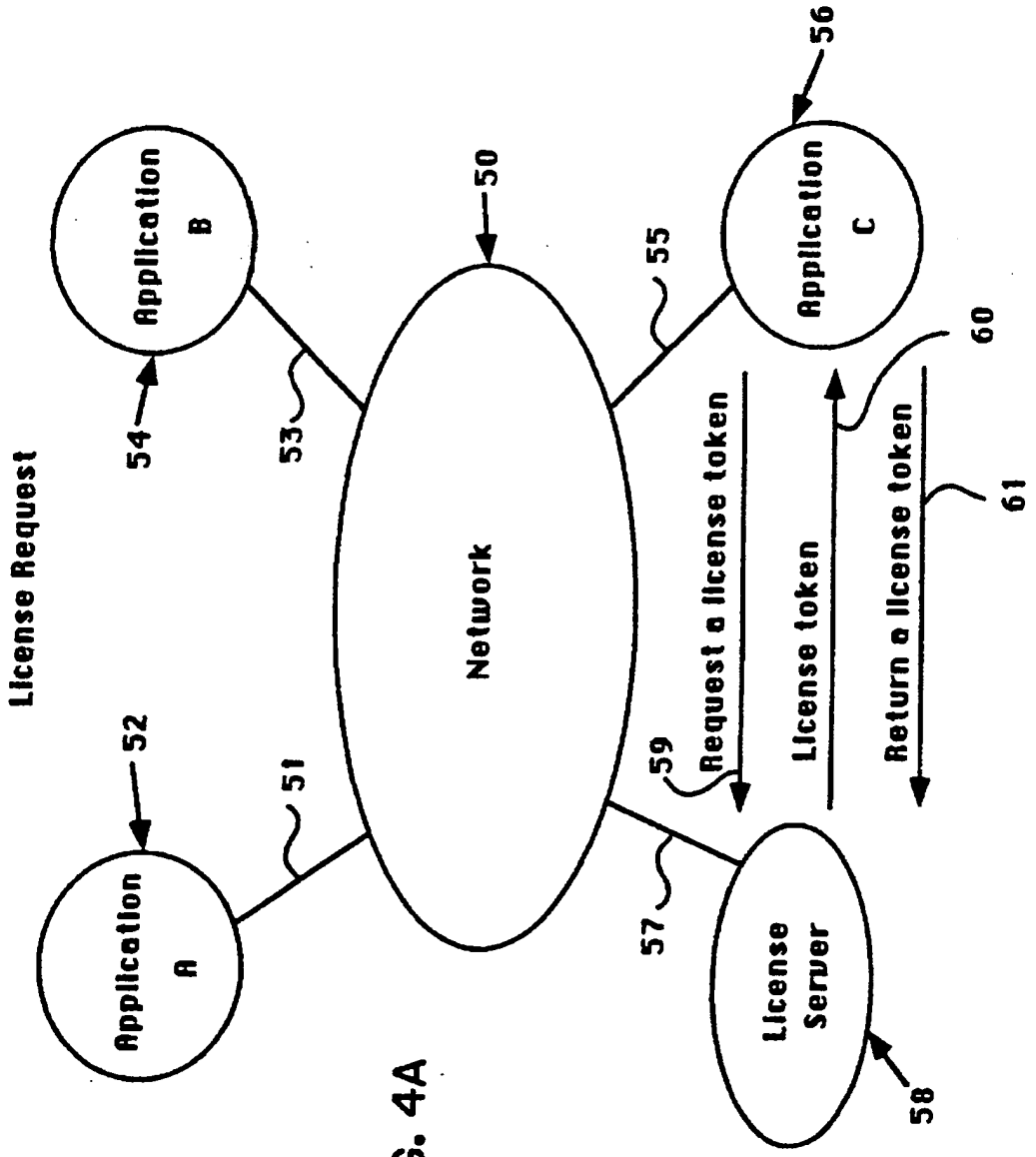

License Request
Application A
Application B
Application C
Network
License Server
50
51
52
53
54
55
56
57
58
59
60
61
Request a license token
License token
Return a license token

FIG. 4A

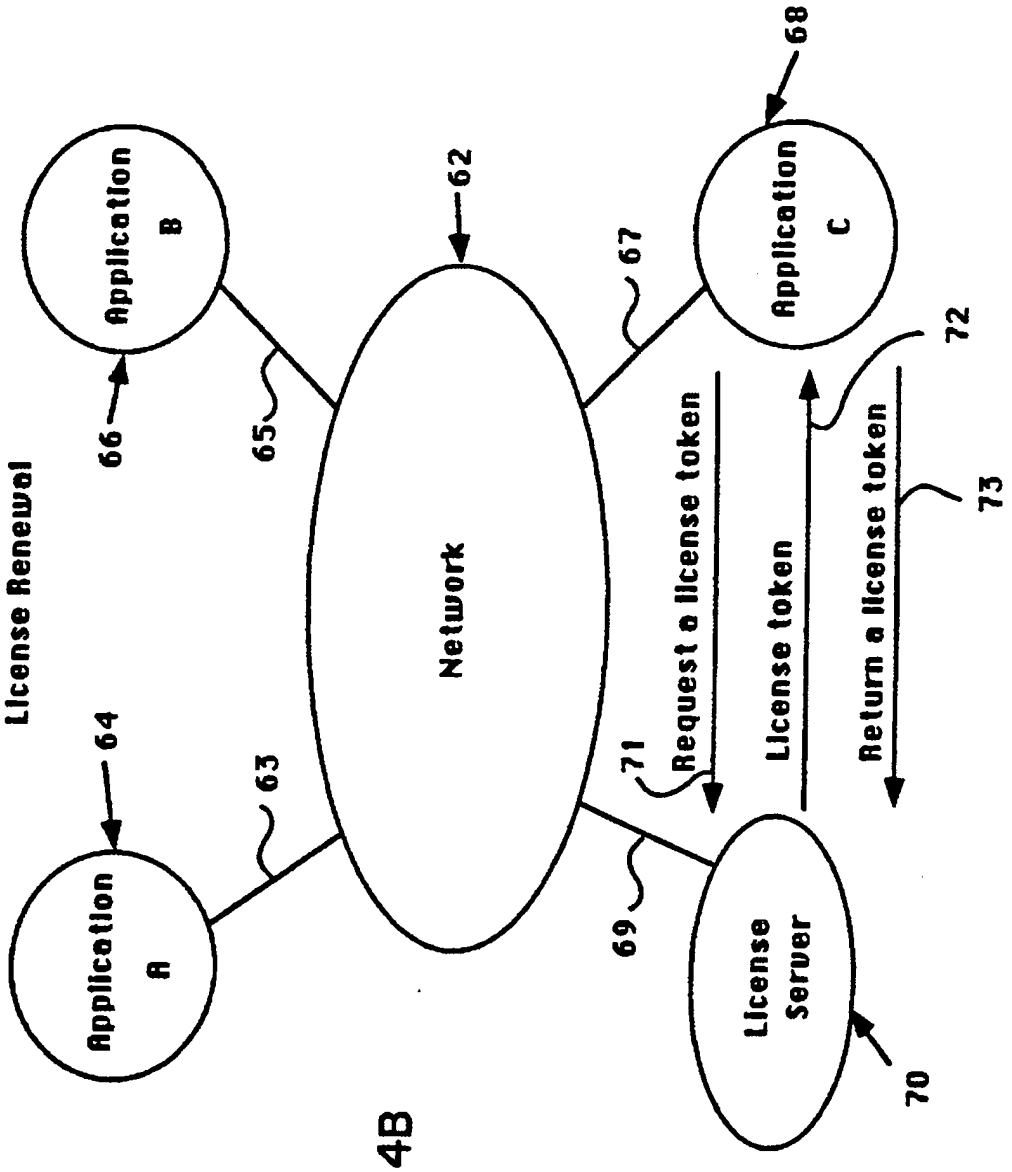

License Renewal
Application A
Application B
Application C
Network
License Server
64
63
66
65
62
67
68
69
70
71
Request a license token
License token
72
Return a license token
73

FIG. 4B

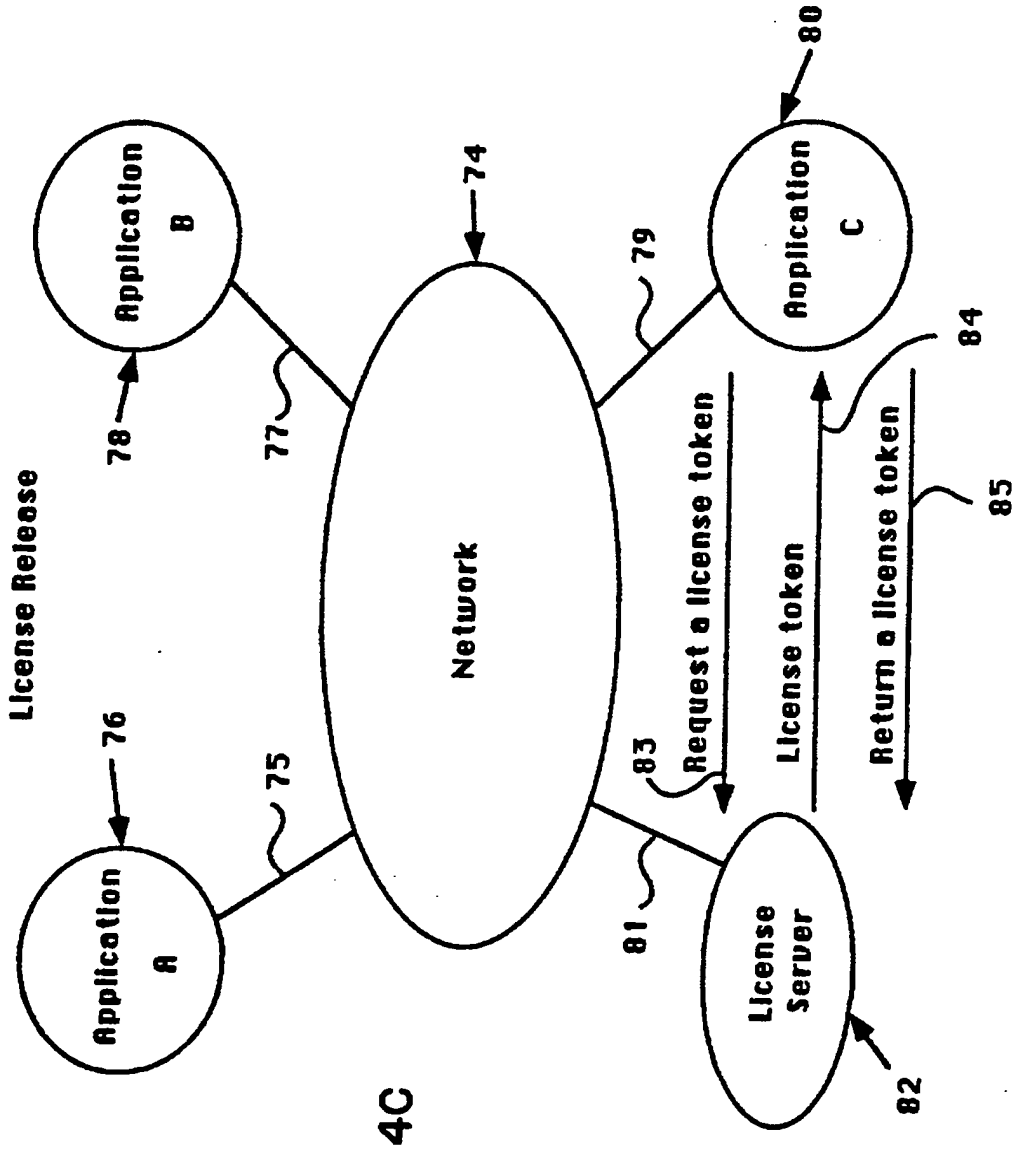

Application B
78
77
74
79
Application C
80
84
License Release
Network
Request a license token
83
License token
Return a license token
85
76
75
81
Application A
License Server
82

FIG. 4C

METHOD FOR PROTECTING AGAINST THE UNAUTHORIZED USE OF SOFTWARE IN A COMPUTER NETWORK ENVIRONMENT

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

The present invention relates to a method for protecting against the unauthorized use of a software application in a computer network environment.

2. ART BACKGROUND

A computer network is typically an interconnection of machines or agents over links or cables. The open access characteristics of a computer network presents opportunities for the unauthorized copying of software, thus eroding the licensing revenue potential of software developers. Traditionally, either the entire network must be licensed (commonly referred to as a site license), or each node where the software is run must be licensed (commonly referred to as a node license). A node refers to a single machine, agent or system in a computer network. A license is an authorization given by a software developer to a customer to use a software application in a specific manner.

A site license lets all users at a designated location or network use the software application, regardless of their position on the network. This flat-fee approach is an overkill for a low usage software application. A node license not only ties a software application to a particular machine in a network, but also is not cost effective for the infrequent use of a software application. See, for example, U.S. Patent No. 4,688,169. Furthermore, if new users of licensed nodes wish to use the software application, they are often required to purchase additional licenses.

An alternative to a site license or a node license is the concept of a concurrent usage license. A concurrent usage license restricts the number of users allowed to use a software application at any given time, regardless of their location on the network. Just as renters check out available copies of a

Petitioner Apple Inc. - Exhibit 1004, p. 5989

movie video from a video rental store, users on a network check out a software application from an agent on a first-come-first-serve basis. Thus, a concurrent usage license charges a fee for the use of a software application proportional to its actual use.

Methods to license a software application for concurrent use in a network environment are currently offered by Highland Software, Inc. and Apollo Computer, Inc. See, M. Olson and P. Levine, "Concurrent Access Licensing", *Unix Review*, September 1988, Vol. 6, No. 9. In general, the license for a software application is stored in a database controlled by a license server. A license server is a program that not only stores the license, but also verifies the user's credentials before checking out the license to the authenticated user. To protect against the authorized use, these methods to license concurrent usage rely on secured communications such as public/private key encryption. Under public/private key encryption, each user of the system has two keys, one of which is generally known to the public, and the other which is private. The private transformation using the private key is related to the public one using the public key but the private key cannot be computationally determined from the public key. See Denning, D., *Cryptography and Data Security*, Addison-Wesley, 1982. The encryption key is hidden in the license server to encrypt the database of licenses. Well designed public/private key encryption schemes are difficult to crack, especially if the license server is located in a trusted environment. A trusted environment is one whose access is limited to users having the proper credentials. However, a license server is more likely to be located at a customer's site and hence in an hostile environment. It follows that the license server is vulnerable to sophisticated intruders. Once the private key is decrypted, all sensitive information on the license server such as licenses are compromised.

Petitioner Apple Inc. - Exhibit 1004, p. 5990

It is therefore an object of the present invention to provide a more secure method to protect against the unauthorized use of software in a concurrent use licensing environment.

Petitioner Apple Inc. - Exhibit 1004, p. 5991

SUMMARY OF THE INVENTION

The present invention provides to the software application the verification and license check out functions which are normally performed by a license server. The preferred embodiment of the present invention comprises a computer network including a plurality of agents running at least one license server and at least one software application. The license server controls a database of an agent containing the license information for the software application. The license information is contained in a license token, and is stored in the database controlled by the license server. The license token is a special bit pattern or packet which is encrypted by the software vendor of the application software. The software application communicates with the license server through a licensing library. The licensing library is a collection of library routines that the software application invokes to request or renew a license from the license server. Before a software application obtains a license, the license token must be decoded by a license access module. The license access module, which is linked with the software application and the licensing library is a program that decodes the license token from a vendor specific format to a licensing library format.

When an user wishes to run a software application, the licensing library invokes a call to request a license token from the license server. In contrast to the prior art where the license server either grants or denies the request after verifying the user's credentials, the license server in the preferred embodiment of the present invention finds the correct license token for the software application and transmits the license token to the licensing library. The license access module attached to the licensing library decodes the licensing token. Routines in the licensing library coupled to the software application verify the license information before checking out the license and updating the license token. The license access module encodes the updated license token before returning it to the license server.

Petitioner Apple Inc. - Exhibit 1004, p. 5992

Because the verification and check out function of a license token are performed by a software application, the software application rather than the license server becomes the point of attack by unauthorized users. Reverse engineering the license access module is less rewarding than attacking the license server because the license access module reveals the contents of a fraction of a database of licenses. By the time most attackers crack the license access module, the software vendors would most likely introduce newer versions of the software application and new license access modules for them. Thus the present invention provides a more secure method for protecting against the unauthorized use of a software application in a computer network environment without modifying the underlying computer network.

Petitioner Apple Inc. - Exhibit 1004, p. 5993

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a network environment employing the present invention.

Figure 2 describes the architecture of a network licensing scheme employing the preferred embodiment of the present invention.

Figure 3 describes the installation of a license token in the preferred embodiment of the present invention.

Figure 4a illustrates the use of a license token to request a license from a license server in the preferred embodiment of the present invention.

Figure 4b illustrates the use of a license token to renew a license from a license server in the preferred embodiment of the present invention.

Figure 4c illustrates the use of a license token to release a license from a license server in the preferred embodiment of the present invention.

Petitioner Apple Inc. - Exhibit 1004, p. 5994

NOTATION AND NOMENCLATURE

The detailed description that follows is presented largely in terms of algorithms and symbolic representations of operations on data bits and data structures within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art.

An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. These steps are those requiring physical manipulation of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bit patterns, values, elements, symbols, characters, data packages, or the like. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

Further, the manipulations performed are often referred to in terms, such as adding or comparing, that are commonly associated with mental operations performed by a human operator. No such capability of a human operator is necessary, or desirable in most cases, in any of the operations described herein that form part of the present invention; the operations are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or other similar devices. In all cases there should be borne in mind the distinction between the method of operations in operating a computer and the method of computation itself. The present invention relates to method steps for operating a computer in processing electrical or other (e.g. mechanical, chemical) physical signals to generate other desired physical signals.

Petitioner Apple Inc. - Exhibit 1004, p. 5995

The present invention also relates to an apparatus for performing these operations. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The algorithms presented herein are not inherently related to any particular computer or other apparatus. In particular, various general purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct a more specialized apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description given below.

Petitioner Apple Inc. - Exhibit 1004, p. 5996

DETAILED DESCRIPTION OF THE INVENTION

The following detailed description is divided into several sections. The first of these sections describes a general network environment for accessing a database of licensed software programs. Subsequent sections discuss the details of a method for protecting against the unauthorized use of a software application.

I. General Network Environment

Referring to **Figure 1**, computer network environment comprises a plurality of data processing devices identified generally by numerals **10** through 10^n (illustrated as **10**, **10'** and 10^n). These data processing devices may include terminals, personal computers, workstations, minicomputer, mainframes and even supercomputers. For the purposes of this Specification, all data processing devices which are coupled to the present invention's network are collectively referred to as "agents". It should be understood that the agents may be manufactured by different vendors and may also use different operating systems such as MS-DOS, UNIX, OS/2, MAC OS and others. Particular examples of suitable agents include machines manufactured by Sun Microsystems, Inc., Mountain View, Calif. Each of the agents has an input device such as a keyboard **11**, **11'** and 11^n or a mouse **12**, **12'** and 12^n. As shown, agents **10** through 10^n (illustrated as **10**, **10'** and 10^n) are interconnected for data transfer to one another by a common cable **13**. It will be appreciated by one skilled in the art that the common cable **13** may comprise any shared media, such as coaxial cable, fiber optics, radio channel and the like. Furthermore, the network resulting from the interconnection of the cable **13** and agents **10** through 10^n (illustrated as **10**, **10'** and 10^n) may assume a variety of topologies, such as ring, star, bus, and may also include a collection of smaller networks linked by gateways or bridges.

Petitioner Apple Inc. - Exhibit 1004, p. 5997

Referring again to Figure 1 is a license service 14. The license service 14 is a resource shared by every agent connected to the network. In the preferred embodiment of the present invention, the license service 14 comprises license servers 15 through 15^m (illustrated as 15, 15' and 15^m) and databases 17 through 17^m (illustrated as 17, 17' and 17^m), where m is less than or equal to n. A license server is a program that runs on an agent with a memory storage capability. Each license server 15 (illustrated as 15, 15' and 15^m) communicates with a database 17 stored in memory on the agent over an interface 16 (illustrated as 16, 16' and 16^m). As will be described in detail below, the database 17 stores licensing information for various software applications which are purchased and authorized to run in the computer network environment. The license server is not limited to run on a specific agent, but can operate on any agent including the agent on which the user is to operate the application. Thus, any agent connected to the network may function as a license server as well as a device on which a user may operate application software. As will be described below, the license server does not perform verification of licenses of application software; rather the license server is passive and provides storing, locking, logging, and crash recovering function for the application software.

Figure 2 illustrates the architecture of a network licensing scheme of the present invention. The architecture comprises a database 18, database interface 19, license server 20, licensing library 24, License access module 27, license administration tool 21, license service binder 29, and license production tool 34.

The database 18 stores licensing information and application usage data. Preferably the database 18 comprises a plurality of records which contain the following information:

Petitioner Apple Inc. - Exhibit 1004, p. 5998

Database Element	Description
Unique Key Table	Keys for all other tables
Vendor Table	Vendor's ID and name
Product Table	Product number and name
Version Table	Version number and date
License Table	License #, exp date, total units
License Token Table	Stores encoded license token
Unit Group Table	A group's allocation of license
Group List Table	Name of the group
Allowed Users Table	Credentials of allowed users
Current License Use Table	Applications using a license
Lock Table	Locked records in database
Authorized administrator Table	Login names of administrators
License Operation Log Table	Administrator's log information
License Usage Log Table	Request handle plus Client Log
License Queue Log Table	License wait queue
Application Message Log Table	Application specific messages

A database interface 19 provides communication between the license server 20 and the database 18 in order to prevent concurrent access to the same database record by multiple users which can cause the data in the record to become corrupted. Thus, only the owner of the lock can read from and write to the locked record during the usage of the application.

The license server 20 operates on an agent and interfaces the database 18 to license administration tool 21, licensing library 24 and license service binder 29. The license server 20 communicates with the license administration tool 21, licensing library 24 and license service binder 29 via an interface 23. Preferably the interface 23 is a remote procedure call

Petitioner Apple Inc. - Exhibit 1004, p. 5999

mechanism which permits a process operating on one device or agent connected to the network to request a resource or service from a remote device or agent connected to the network. See A. Birrell and B. Nelson, "Implementing Remote Procedure Calls," *ACM Transaction on Computer Systems*, February 1984, Vol. 2, No. 1.

Multiple license servers may reside on multiple agents. Preferably the license server 20 operates in a background mode of the agent such that its operation is transparent to a user of that agent. More particularly, as will be described below, the license server 20 provides the following functions: 1) servicing the requests from the licensing library 24 for license token; (2) maintaining a wait queue for requests to the database 18 when no licensing units are available; (3) generating `locks` for exclusive access to database 18; and (4) providing access to information in the database 18.

The licensing library 24 is a set of library routines which enable the application 26 to request licensing service from the license server 20. Upon receiving the request for service from the licensing library 24, the license server 20 retrieves a license token from the database 18 and transmits it to the licensing library 24. The licensing library 24 is linked with the application 26 and communicates with the license server 20 over a path 28 with, preferably, a remote procedure call mechanism 23. Among the major library calls in the licensing library 24 is the application's request for a license from the license server 20. Other important library calls include the request to renew and to release a license. The use of the license token to accomplish the request for the various licensing service will be described in detail below.

The license access module (LAM) 27 is prepared by the software vendor 24 to decode the license token. Once decoded, the application 26 via routines in the licensing library verifies the licensing information in the license token and determines whether a license may be checked out. The LAM 27

Petitioner Apple Inc. - Exhibit 1004, p. 6000