Information Sharing in Collaborative Environments

M. Kolland, A.Jarczyk, P. Loeffler
Siemens AG
Corporate Research and Development
Munich, Germany

Abstract

In the 80ties, the personal computing paradigm has fundamentally changed the way people work within organizations. The nineties will see a similiar revolution - workgroup computing. This term describes IT support of work groups in the context of complex, interdisciplinary tasks in geografically dispersed organizations. Efficient information sharing is the key to effective collaboration support. In the context of typical collaborative environments however, (geographically dispersed, mobile and location independent sites, independent component failures, concurrent activities) building applications which support information sharing between separated and autonomous operating users is extremely difficult. This paper describes an infrastructure which addresses these issues by providing the abstraction of a shared information space and in this context offers a variety of collaboration specific consistency models and coordination mechanisms.

1 Introduction

Each Groupware application deals to a major extent with two fundamental issues, coordination and management of shared information. Coordination of activities supports the group members in achieving their common goal. Making the respective group information uniformly accessible within one orthogonal, easy-to-use model makes cooperation more efficient and productive [9]. Such a group information model is often referred to as *shared information space* [10]. In a Group Calendar system for example this shared information space may consist of the individual calendars of group members plus information on availability of meeting rooms. In a Cooperative Software Development system on the other hand the shared information space would comprise all files necessary to build an instance of the respective system plus related documents (review protocols, test protocols).

Implementation of a shared information space in a centralized architecture is relatively straight-forward. All relevant information is managed by a central server which ensures consistency and enforces specific coordination policies. Centralized architectures however fall short in several aspects [17] :

- **Responsiveness** - for highly interactive multi-user applications delays caused by slow WAN connections or other resource bottlenecks may prevent a user-friendly implementation and reduces user acceptance. This is a serious problem for environments with mobile, location independent components which are especially aimed for by groupware applications.
- **Autonomy** - communication/component failures or disconnected operation should not prevent the group activity from proceeding. In future Groupware settings (geographic dispersion, mobile components) component failures must be tolerated by the application, in order to achieve maximum group productivity.
- **Openness** - centralized architectures assume a homogeneous environment throughout the participating group members. The more Groupware is extending its reach in terms of group size and geographic dispersion, the less this assumption holds. Heterogeneity and integration of existing applications and programming environments are essential.

In settings where the above requirements must be met, the shared information space must be replicated in a way that each group member holds a local copy of (at least part of) the group information. In such a replicated architecture consistency of shared information becomes a major source of complexity, caused by typical properties, of groupware environments like wide-area distribution, heterogeneity, mobility and independent failure of components.

0-8186-5705-7/94 $4.00 © 1994 IEEE

Allthough a variety of approaches deal with this complexity, mainly in the field of databases and file systems [11] [21] many of them do not reflect the specific groupware needs [19]. Therefore we argue for the introduction of a support layer which deals with theses issues in a more Groupware specific way.

This paper describes the design and implementation of COCOON [1], a software layer which supports a shared information space model generic to a wide range of groupware applications. It provides the benefits of a replicated architecture while hiding the complexities inherent in such an approach. COCOON offers a variety of *consistency models*, typical for collaborative scenarios. The following chapters describe the COCOON information-, architecture- and the operational models. The second part of the paper focuses on various consistency models and gives a formal description of their semantics in the context of Groupware environments. The third section describes the implementation of COCOON on the RDO/Smalltalk system. RDO/Smalltalk represents an interface between the Smalltalk programming environment and the ISIS toolkit, developed at Cornell University. The last part of this paper describes the implementation of a simple groupware application on top of COCOON, to illustrate and verify the usefulness of the provided abstractions.

The work described here focuses on the shared information space abstraction of COCOON. For reasons of complexity and genericity COCOON will also provide support in areas like history, undo/redo and persistence. These mechanisms are closely related to the shared information model but are not discussed in this paper.

2 The COCOON Shared Information Space

2.1 The Information Model

In order to provide effective support for the application programmer, the COCOON information model must both support a simple mapping from real-world group information entities to entities of the COCOON model and must reflect state-of-the-art design methodologies and corresponding tools [24]. On the other hand the information model must be as simple as possible, in order to reduce the complexity of the COCOON layer and allow maximum transparency of the replicated architecture.

The COCOON information model is object-oriented. A COCOON object represents a real-world information entity. Attributes hold the entities content. Links represent relationships between objects. Attributes, links and their corresponding access-methods can be written by the programmer or generated automatically by some OO-CASE tool. Fig.1 illustrates an example.

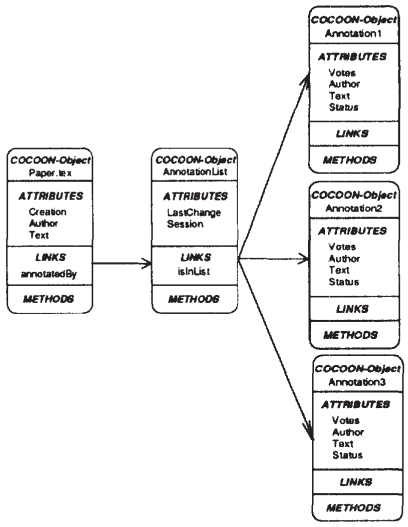

Figure 1: Example of a simple COCOON object net

A document within a group discussion application may be modelled as a COCOON object with several attributes which reflect its status (text, creation date, author). A list of annotations is associated with each document, modelled by a separate COCOON object. This object is linked to the document by the *annotated-by* relationship. Each annotation is itself a COCOON object with attributes reflecting its current state. Annotation objects are linked to the annotation list via the *isInList* relationship.
The result is an object net which represents the real-world information to be modelled. Such an object net is typically expressed in an object-oriented programming language. COCOON does not imply a specific object-oriented programming environment. In fact it

[1] COCOON = COnsistency and COordination in Open Networks

supports access to the (shared) object net from different object-oriented programming languages.

2.2 Shared and Private Information Spaces

A session in COCOON is a cooperative activity which is typically represented by a multi-user groupware application or a set of cooperating single-user applications. The application(s) manipulate a shared information space. Members of a session, usually represented by local agents of the application, have access to the shared information space through their local environment (COCOON and the local object-oriented programming environment).

COCOON is centered around the notion of private and shared information spaces. Information relevant in a session is represented by an object net that can either reside in a private or in a shared information space.
Within a private space, each change to the object net is made locally and invisible to other group members. In a shared information space each change is visible to all group members and changes to the object net can be made by all group members concurrently (subject to some optional coordination policy). A private space can be transferred into a shared information space in the course of a session.

A fundamental concept used in COCOON is that of an unique object identifier. In order to allow uniform access of objects in a shared information space by all local environments, an object must have a session wide, unique identifier. Every object which is placed in a COCOON information space is referenced by such an identifier (OID). Creation and management of object identifiers is currently not part of COCOON, OIDs are specified by the application. Issues in creating and managing these identifiers in a distributed, heterogeneous environment have been specifically addressed in other systems. For details we refer to [23] [16].

COCOON provides five operations to manipulate a private or shared information space :

- `createObj(InfoSp,OID,Obj)` - This operation inserts the local object *Obj* in the respective information space under the unique identifier *OID*. If it is a shared information space, every member of the session has now access to it.
- `deleteObj(InfoSp,OID)` - This operation deletes the object with unique identifier *OID* from the respective information space. The object is no longer accessible in the context of this information space.
- `createLink(InfoSp,OID1,OID2,Linkname)` - This operation establishes a (unidirectional) link between two objects in the respective information space. Objects are denoted by their unique identifiers *OID1* and *OID2*. The name under which this link will be established is denoted by *Linkname*.
- `deleteLink(InfoSp,OID,Linkname)` - This operations deletes a link between two objects in the respective information space. The object from which the relation should be deleted is given by its unique identifier *OID*. The name of the link which is to be removed is given by *Linkname*.
- `changeAttr(InfoSp,OID,AttrName,Obj)` - This operation changes an attribute value of an object in the respective information space. The object is denoted by *OID*, the attribute by *AttrName*. The value of the attribute will be set to *Obj*.

2.3 Operational Model

The COCOON operational model is simple yet generic in order to support a variety of groupware scenarios. It is closely related to the COCOON consistency models (see chapter 3).

A new session and its related shared information space is created by one member of the group usually the coordinator of the group activity. Other group members can join the session, gaining access to the shared information space. The shared information space, which is initially empty, can be set to an initial status, before the group activity starts and group members join. Its up to the application (i.e. the initiatior of the session) to do this or immediately begin with the group activity. In the COCOON model, the initial content of the shared information space is typically constructed by using the *createObj/createLink* operations described above.

During the group activity, changes are made by the application using the five operations provided by COCOON. The visibility of changes to the shared information space in the sense of a *what-you-see-is-what-I-see* [22] property is subject to various consistency policies, which can be specified with each COCOON

operation (see chapter 3 for details). Each update operation results in the change of the object net and a notification to the current set of session members describing the change. This allows the application to take the appropriate action (e.g. redisplay an object within the user interface). Members can leave the session in several ways, loosing access to the shared information space. Private spaces, which are created in the course of the group activity may continue to exist until explicitly deleted by the respective creator.
Another option provided by COCOON is a *disconnect* from the group. The semantics of this operation includes a later reconnect to the session and an automatic transfer of the updates which have occured to the shared information space during the disconnected period. Reconnecting to the group can be done according to several consistency policies (see chapter 3 for details). This corresponds to the needs of mobile and location independent computing which will play a major role in future Groupware settings [20].

Private and public spaces can be merged via replaying the updates to the private space in the public space All updates to the private space can thus be made visible to the other session members.

In order to support history, undo and redo functionality (which is a major source of complexity within Groupware applications), COCOON provides access to the list of updates recently made to the shared information space. This includes manipulation of update operations, removal of redundant or obsolete operations (e.g. createObj(OID,Obj) ... deleteObj(OID)) and checking for conflicts within concurrent histories (occuring if members are temporarily disconnected from the session). To support undo/redo functionality COCOON provides the possibility to replay an arbitrary list of COCOON update operations within a shared information space. Such a list can be constructed by the application according to some specific policy (operation specific, member specific, ...).

Persistence has not been addressed within COCOON so far. There are basically two possible approaches. Making the object net of a shared information space persistent would be the straigth-forward way. Persistent object systems provide support for this purpose [6]. Another approach we see is to make the history of changes to an object net persistent. The object net can thus be reconstructed from scratch. Such an approach would be advantageous within heterogeneous environments and in conjunction with history/versioning schemes. However scalability will be a problem. We will investigate a hybrid approach (checkpointing the object net combined with update lists), as part of our future work.

2.4 Architectural Model

For reasons described earlier, COCOON is deploying a replicated architecture. This means that the content of a shared information space is replicated at the local site of each session member. Each object, referenced within a shared information space, is represented by a copy in the respective local application environment. The mapping between the OID and the local object is done transparently using a local table (object dictionary). Allthough private information spaces can only be accesed by the creating instance, the same approach is used. This eases later transfer into a shared information space.

Update operations are sent to the COCOON server, a central instance which distributes the updates to all session members. On each site a local COCOON agent receives the update operations from the COCOON server, changes the object dictionary and the local copy of the object net respectively. It then notifies the application associated with the respective session. The COCOON server also keeps track of the histories of all shared information spaces including the histories of currently disconnected members. Replication of the COCOON server provides for fault-tolerance and high-availability.

Fig. 2 illustrates the COCOON architectural model. It depicts a scenario with user Peter, Alex and Markus. Markus and Alex have joined a session which is represented by the shared information space *Session1*. User Markus is also member of a another session including user Peter. The respective information space is denoted with "Session2". A central COCOON server is distributing updates and keeps track of the histories. For simplicity reasons this picture does not show private spaces.

3 The COCOON Consistency Models

COCOON provides a number of mechanisms which reflect different models of keeping the local replicas of a shared information space consistent. Theses mechanisms can be divided according to their use in synchronous or asynchronous collaboration scenarios.

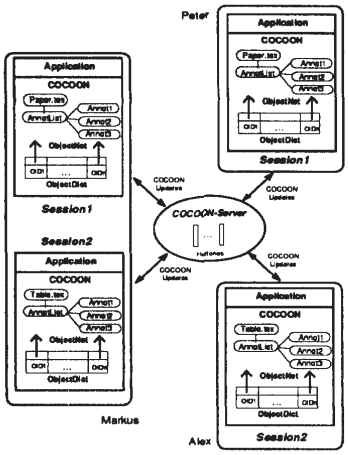

Figure 2: COCOON architecture

3.1 Consistency for Synchronous Collaboration

Referring to the time/space matrix of group collaboration [8], applications in the synchronous domain support activities which envolve interaction at the same time from the same/different location(s). Tools like *Group Editors* or *Meeting Support Systems* are typical representatives for this domain [18]. They provide all group members with a consistent view of the respective group information. Consistent in this case is often synonymous for identical. In general, the term *consistent* is highly application specific depending on the activity to be supported, the information involved and the group structure.

In the COCOON information model, consistency of the shared information space is dependent on the order in which updates are received and applied by the session members. COCOON can deliver updates with several ordering guarantees. Theses can be specified by the application on a per-update basis.

3.1.1 Causal Consistency

In this mode, COCOON guarantees that updates which are causally related are delivered to all COCOON instances in the same order. The following gives a formal description. For further details we refer to [3].

Notation

$C = \{c_1, \ldots, c_n\}$	set of connected sessionm members
$U^{c_i} = u_1^{c_i}, \ldots, u_m^{c_i}$	sequence of updates seen by c_i
u, u_1, u_2	updates to be posted next
$\prec$	partial order of updates (*happened-before* relationship [12])

CC(u)

(1) $\forall c \in C : u \in U^c$
(2) $u_v^{c_i} \prec u \Rightarrow (\forall c \in C : (u = u_k^c \wedge u_v^{c_i} = u_j^c) \Rightarrow j < k)$

This mode is useful for applications, in which a consistent view of the shared information space means preservation of causality in the changes to the object net (e.g. an information structuring and argumentation tool). Another area might be an application, where a coordination policy prohibits concurrent updates to the same part of the object net (e.g. editing the same paragraph). In such an application, the causal mode essentially guarantees that all updates made to the respective part of the object net are received by all members in the same order.

3.1.2 Total Consistency

In this mode COCOON gurantees that updates to the shared information space arrive in the same order at all group members. A detailed description is given in [3]. This will be the most frequently used mode for synchronous collaboration. It gurantees that after an update has been made, all members currently in the session will arrive at the same state of the object net before the next update is visible at any site.

Notation

$\prec \bullet$	Total order according to a first-come-first-serve policy

TC(u)

(1) $\forall c \in C : u \in U^c$
(2) $u_n^{c_i} \prec \bullet\ u \Rightarrow (\forall c \in C : (u = u_k^c \wedge u_n^{c_i} = u_j^c) \Rightarrow j < k)$

The implementation real-time shared objects like pointers or drawings on a sketchpad will typically use this consistency mode.

3.1.3 Atomic Consistency

This mode guarantees, that a set of update operations, which must be applied without interleaving, are delivered atomically to the session members. This can be combined with any other consistency mode. The ordering then applies to the set of update operations to be delivered atomically.

$\mathbf{AC(u_1, u_2)}$

$$(1)\ \forall c \in C : (u_1 \in U^c \wedge u_2 \in U^c)$$
$$(2)\ \forall c \in C : ((u_1 = u_k^c \wedge u_2 = u_l^c) \Rightarrow\ l = k + 1)$$

The atomic mode is useful in settings where different COCOON operations must be executed atomically at all member sites A group calendar application for example [18] has to update each team members calendar in one atomic operation in order to avoid inconsistencies. COCOON currently provides only a rudimentary implementation of this scheme, which supports atomicity for sets of updates referring to the same object.

3.2 Consistency for Asynchronous Collaboration

Asynchronous collaboration is characterized within the time/space matrix as supporting interactions, which take place at the same/different location(s) at different times. Support for these collaboration scenarios reflects two major influences on groupware systems : globalization of teams (e.g. different timezones) and mobility of team members (e.g. temporal disconnection caused by break-down or high cost of the communcaition link). In COCOON we view a group session as consisting of synchronous and asynchronous collaboration patterns, supporting both activities of active/connected and inactive/disconnected session members. Updates are seen by all connected members immediately and will be replayed to disconnected members after they rejoin the session. COCOON provides various consistency models for this scenario.

3.2.1 Weak Consistency

This COCOON model reflects the observation, that certain updates to the object net may only be important to connected session members or until later updates make them obsolete. Note that this differs to the notions of *Weak Consistency* introduced in [13]. Goldings approach corresponds to the *Disconnect-Reconnect-Merge* model described later. There are two mechanisms to support *Weak Consistency* :

- COCOON allows the application to specify the expiration of an update. If the expiration condition is met, the message will not be seen by rejoining session members.
- Certain update combinations are obsolete regardless of explicit expiration. COCOON provides a mechanisms to automatically remove these updates from a given history. `createObj(OID,...)` / `DeleteObj(OID)` pairs and all updates referring to the same OID in between have no effect on the shared information space (analogous `createLink(OID1, OID2, Rel)` / `DeleteLink(OID1, OID2, Rel)` pairs). The same is true for `createLink(OID1, OID2, RelName)` lists with identical *OID1* and *RelName* parameters and for `changeAttr(OID,AttrName,Obj)` with identical *OID* and *AttrName* parameters. They can be removed from the respective history except the last entry.

Expiration of messages can be specified in two ways. A numerical value N assigned with each update triggers expiration after N updates are made to the shared information space (WC1 below). A type value triggers expiration after another update with identical type is received (WC2). Types are specified by the invoker of the update operation. The type mechanism can be augmented with a numerical value N to trigger expiration after N updates with the respective type value are received (the current implementation is somewhat limited in this context).

A typical example for this consistency model is the implementation of real-time shared objects : updating a pointer on a sketch-pad is only necessary for connected members. It is not necessary to keep these updates in the history and replay them for rejoining members.

3.2.2 Initial-Join-Consistency

This consistency model ensures, that latecomers to the session will eventually see a state of the shared information consistent with the changes made within the session so far. The notion of consistency is subject to the expiration policy, specified per update message (see above). In the general case, all updates will be replayed to the joining session member. COCOON ensures, that the join phase is coordinated with other group activities, using a simple coordination policy (updates during then join phase are prohibited). Join

consistency is used in scenarios, where latecomers to a session have to be accomodated.

3.2.3 Disonnect-Reconnect-Join

COCOON does not only support latecomers but also members which temporarily disconnect from a session. In this case consistency means that only the recent updates, made since the respective member has left the group have to be replayed. Analoguous to the *Initial Join* case, theses updates may be subject to an expiration policy. Note, that this consistency model does not preserve updates to the shared information space, made by the disconnected member.
As pointed out above, this form of consistency is important for applications, which support mobile and location independent computing. In these scenarios, components can be temporarily disconnected either deliberately (communication costs) or subject to a communication failure. In many cases it is makes no sense to support further activites of the disconnected session member. In order to reach consistency after a rejoin of this member, it is sufficient to replay the group updates.

3.2.4 Disonnect-Reconnect-Merge

In general, prohibiting disconnected members from updating the shared information space is too restrictive. For many applications this policy will negatively affect group productivity and thus conflict with the basic objective of each groupware application. COCOON provides a consistency model which overcomes this restriction. Updates made by the group and by the disconnected member are both considered part of the shared information space after the rejoin. During the disconnected period, COCOON updates both the group and disconnected member histories. During the join phase, two replay lists are constructed - one which is replayed at the singleton(`replayAtSingletonList`), the other at the group (`replayAtGroupList`) in order to reach a mutual consistent state. The following outlines the algorithm used by COCOON to construct these lists :

```
replayAtGroupList := singletonHistory
replayAtSingletonList := groupHistory
while not empty(singletonHistory) do
    affectedSingletonUpdates :=
    AffectedBy(singeltonHistory,
               first(singletonHistory))
    affectedGroupUpdates :=
    AffectedBy(groupHistory,
               first(singletonHistory))
    if   empty(affectedGroupUpdates)
    then remove(affectedSingletonUpdates,
                singletonHistory)
    else remove(affectedSingletonUpdates,
                singletonHistory)
         remove(affectedGroupUpdates,
                groupHistory)
         (singleton,group) :=
          resolveConflict(
              affectedSingletonUpdates,
              affectedGroupUpdates)
         remove(group,replayAtGroupList)
         remove(singleton,
                replayAtSingletonList)
    fi
done
```

Initially the singleton and group replay lists are a copy of the group and singleton histories. For each update in the singleton history, a list of updates which affect the same object is constructed. The corresponding list is constructed from the updates in the group history. Note that only updates, which affect the same object are crucial. So a conflict occurs only, if the `affectedGroupUpdates` list is not empty. Since the handling of conflicts is application specific, the `resolveConflict()` method must be provided by the application. Its return value is a list of updates to be removed from the singleton and group replay lists in order to achieve consistency. Both replay lists are then replayed at the group and singelton members respectively.
The application discussed in chapter 5 is using this consistency model. It provides a specific `resolveConflict()` method, which implements a *group-has-it-right* policy. Updates to an object, which result in a different object state depending on the order they are applied are regarded as conflicts and the updates made by the singleton will be superseded by the group updates. This policy in conjunction with the DRM consistency model results in one identical object net after the rejoin of a disconnected member while preserving non-conflicting updates on both sides. A formal specification of the *group has it right* policy used in the application and a formal proof of its correct implementation has been carried out but is out of the scope of this paper.

There are a number of further consistency models proposed for collaborative scenarios which are currently not reflected in COCOON [1]. Note that the COCOON approach provides a high degree of flexibility

in integrating these mechanisms (see chapter 4).

4 Issues in Implementing COCOON

4.1 The ParcPlace Smalltalk Programming Environment

As pointed out earlier, COCOON implies an object-oriented programming model. Although it is independent of any specific OO programming language or environment and moreover explicitly supports heterogeneous settings we had to decide on a suitable OO programming system for building the COCOON prototype.

For building the COCOON prototype we chose the ParcPlace4.1 Smalltalk system. We found Smalltalk especially appealing, since

- it provides an object-oriented programming model, which fits closely with modelling real-world cooperative taks.
- it is highly interactive and reuse oriented, which allows a rapid prototyping approach - genericity and usefulness of the COCOON abstractions can thus be evaluated by modelling a number of collaboration scenarios without putting too much effort into application development.
- it supports a very efficient way of implementing user interfaces. To evaluate COCOON it is therefore possible to focus on the application functionality with respect to COCOON abstractions.

4.2 ISIS/News as communication infrastructure

The ParcPlace Smalltalk system itself does not provide any support for cooperation between remote objects. This is however the backbone of every Groupware application. Allthough there are various remote object packages available they do not resemble closely enough the typical interaction patterns inherent in groupware settings nor do they address issues of consistency.

The ISIS system has been used for this purpose within various Groupware settings. ISIS and ISIS/News are described in detail in [4]. ISIS provides a powerful notion of process groups and ordered communication between group members. For implementing the COCOON model, we found the ISIS/News facility most helpful. It nicely resembles the patterns of distributing information within a collaborative task. ISIS/News uses the notion of *news-posters* and *news-subscribers*. Each *subscriber* registers interest in one or more *subjects* . A *news-poster* sends messages to a *subject*. This results in a multicast of the message to all current *subscribers* to the respective *subject* .
The order in which messages arrive at the *subscribers* is subject to various consistency constraints, to be specified by the the *news-poster* . Total ordering guarantees the delivery of messages in the same order at every *subscriber*. Causal ordering guarantees that causal relationships between messages are seen by each *subscriber* correctly. Messages not causaly related can be received in arbitrary order. ISIS/News has several additional features, which are helpful in implementing COCOON :

- The NEWS *backlog* mechanism provides a per subject history facility. Instances subscribing to a subject can specify the amount of history they want to receive from the *backlog* before seeing actual postings. This enables subscribers to see and act on a history of the subject.
- The amount of history received on subscription can be controlled by the subscriber with a variety of mechanisms. Setting a *position marker* e.g. allows a subscriber to temporarily disconnect form the topic and later resubscribe, receiving the history from the position marker onwards.
- The spooling of messages in the *backlog* can be controlled by several mechanisms, providing an expiration of messages in the history based on different aspects.
- Backlogs can be logged to non-volatile storage providing a means to make histories persistent.
- News provides a means to detect, whether a local subscriber/poster is currently disconnected form the service (e.g. network partition) and thus allows appropriate action.
- News can make use of the ISIS long-hawl communication facility. Messages can be sent to subscribers on different LANs over a (slow) WAN connection by using a spooler. The spooler receives messages and delivers them asynchronously to session members in other backbone LANs.

4.3 Integration of ISIS/News into Smalltalk

The ISIS toolkit consists of a number of servers and libraries running in a UNIX environment. Providing an interface from the Smalltalk system to ISIS was a prerequisite for our implementation strategy. This work has been carried out independently of the COCOON activities and resulted in a system called RDO/Smalltalk [7]. The integration of ISIS and ISIS/News into the ParcPlace Smalltalk system faced two major issues :

- Integrating the neccessary components of the ISIS system into the Smalltalk virtual machine.
- Integrating the News model of distributed interactions into the Smalltalk model as smoothly as possible.

Integration of the ISIS run-time library into the Smalltalk virtual machine was done via the C-Interface provided by the ParcPlace system (CPOK). ISIS functionality is thus made available to Smalltalk class and instance methods. Calls can be made bidrectional - from Smalltalk into ISIS and vice versa providing a means to support ISIS upcalls. The low-level details of this interface is hidden behind abstract classes, representing the respective ISIS functionality (e.g. NEWS).
A seamless integration of the ISIS/News *publish-subscribe* model for remote interactions has been accomplished by logically extending the Smalltalk dependency mechanism. The original Smalltalk mechanism only allows for dependencies between local objects. By making an object dependent on a local proxy object (representing a singleton or a group of remote objects), the respective instance is notified of changes on the remote side. This provides the same semantics as in the local case :

- The proxy, which represents the remote instance, has to be created first. It corresponds to a ISIS/News subject. Upates are posted to and received via the proxy.
- Every update of the remote instance (singleton or group) is automatically forwarded to all dependents - the guys delivering updates are *news-posters* the dependents are the *news-subscribers*.
- The ISIS/News service is responsible for delivering the updates in a consistent (totally or causally ordered) fashion.
- This mechanism is augmented by furhter ISIS/News functionality (backlog, long-hauwl connections, ...).

In the case of Smalltalk, the integration of ISIS/News and the programming language can thus be made very elegantly. The mechanism described above is used extensively within the COCOON layer. This use of a Smalltalk specific mechanism within COCOON does in no way conflict with the goal of supporting heterogeneous settings, since the *dependency mechanism* for propagating updates to objects can easily be emulated within other programming environments.

4.4 Implementing the Shared Information Space

Fig.3 illustrates the implementation of the shared information space abstraction within the Smalltalk-ISIS/News environment.

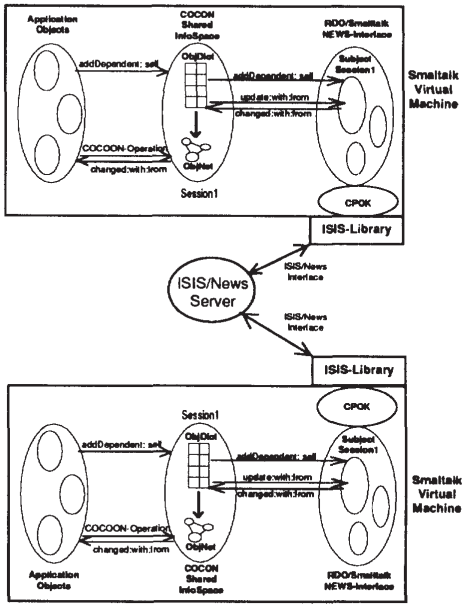

Figure 3: Updating Shared Information with RDO/Smalltalk

An application object creates (or joins) a session by creating a local shared information space object. This object contains the object dictionary and local object

net. The shared information space object creates a subject instance within the RDO/Smalltalk with the session name as identifier (or attaches itself to one if already present). The object dictionary is dependent on the subject instance, the application object is dependent on the shared information space. Creation of a subject results in a *subscribe* call to the ISIS/News server. Calls to the ISIS environment are handled by the ISIS-library, which is accessible through the Smalltalk C-interface (see chapter 4.3).

Invoking a COCOON operation results in a call to the respective method of the shared information space object. The update is posted to the local subject instance via the `update:with:from` message. The subject instance itself calls the respective ISIS/News routine via RDO/Smalltalk (see chapter 4.3).

An incoming update is delivered to the respective subject instance by RDO/Smalltalk. The subject sends itself the `changed:with:from` message, which results in an `update:with:from` messages to all its dependents. The receiving object dictionary is updating its internal table and object net accordingly and notifies its dependents (the application object) with the `changed:with:from` message.

4.5 Implementing the Disconnect-Reconnect-Merge Consistency Model

Fig.4 illustrates the implementation of the *disconnect-reconnect-merge* consistency model.

A member can either explicitly request disconnection from the group or a a network partition is detected. On the member side, this results in the creation of an *UpdateList* within the local shared information space. The `update:with:from` method used to propagate locally invoked COCOON operations writes updates to this list instead of posting them to RDO/Smalltalk. The last update seen from the group is denoted with a postition marker *Pos*, which points to the respective entry in the ISIS/News history list. The current RDO/Smaltalk interface does not support the ISIS/News position marker functionality. It is therfore implemented by COCOON, posting special messages whenever a disconnect of a session member occurs. Both, managing the position marker and the local update list, is transparent to the application.

On the group side things also happen transparently. Updates are propagated to session members via RDO/Smalltalk. The backlog, which holds the changes to the shared information space since its creation is managed and automatically updated by the ISIS/News server.

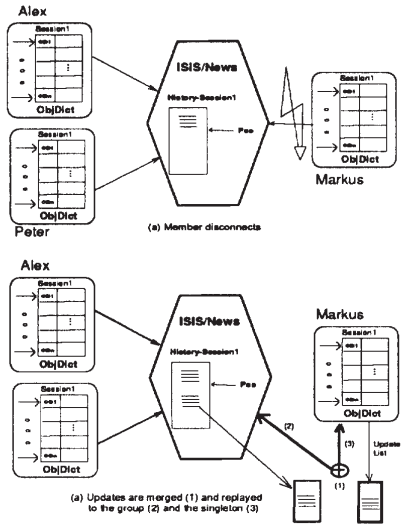

Figure 4: Implementation of Disconnect-Reconnect-Merge consistency

The interesting case is the reentry of the singleton into the group. To freeze the state of the group during the merge, local updates to the shared information space are prohibited. The singleton calls the COCOON reconnect method. This results in a subscription to the respective ISIS/News subject. The subject history from the point where the singleton left is transferred by RDO/Smalltalk transparently. This results in a number of `update:with:from` invocations.

- In the disconnect-reconnect-join case, these update invocations are handled as described in chapter 4.4, changing the local object dictionary and object net respectievely. After the last update from the list has been encountered, COCOON resumes normal (synchronous) session operation.

- In the disconnect-reconnect-merge case, no updates to the local data structures are made in the first place. Moreover a list is created, representing the group history (similiar to the singletons *UpdateList*). Both the group and singletons update lists are now merged according to the al-

gorithm described in chapter 3.2. The result is a pair of update lists, which are replayed at the singleton and at the group respectively, in order to arrive at a consistent state. COCOON makes this transparent for the application.

Since the period in which members are disconnected is usually relatively short, the length of the histories doesn't seem to pose any scalability problems. In the case of latecomers, where history lists can get very big, a hybrid solution may be more scalable. The object net is checkpointed to some central server at fixed intervals. Each checkpoint causes a position marker in the history to be set appropriately. A latecomer can reconstruct the session state from the checkpoint and the list of updates made since the chekcpoint has been written.

Note, that the concept of group and singleton histories also allows a very simple and elegant implementation of additional consistency models (e.g. version trees), undo/redo mechanisms and persistence support. Investigating these issues is part of our future work.

5 Using the COCOON Abstractions :

In order to verify the approach taken in COCOON, we implemented differnt groupware applications. Our goals were twofold : (a) we wanted to build applications which support real-world cooperative tasks and (b) those tasks should include a variety of different collaboration scenarios so that we could use as much of the COCOON mechanisms as possible in order to evaluate their genericity and usefulness. Performance of the COCOON implementation was a further concern.

5.1 An Application to support Code Inspection Meetings

Given the above considerations we implemented a tool to aid code inspection review teams similiar to ICICLE [5]. We do not describe the application in detail here. The following list gives a brief overview of its functionality with respect to the used COCOON abstractions :

- Each session has a READER and several REVIEWERS. The READER initiates the session creating an empty shared information space. REVIEWERS will typically join the session after the READER has set up the initial object net, using the *Initial Join* consistency model. A typical object net occuring in the course of a session is depicted in Fig.1. REVIEWERS can also join at any later point in time.
- The READER is in charge of browsing through the code listing. He also manipulates a grafical object to point to the line currently under review. The code window and pointer objects are implemented as real-time shared objects. Strict WYSIWIS holds for these objects in the context of the session members. Real-time shared objects are implemented as updates to the object net using the COCOON *total ordering* consistency model (in case of updating the code window, where start and end position have to be updated atomically, the *atomic ordering* consistency model is used.
- REVIEWERS are guided through the code listing by the READER as described above. Their task is to make comments the respective lines under review. The proposal of an annotation is made visible to all session members by updating the object net according to the COCOON *Causal Consistency* model. The same COCOON abstractions are used for posting votes on a annotation (REVIEWERS) and deciding an annotation accepted, rejected or deferred (READER).
- Latecomers to a review session are implemented via the *Weak* and *Initial Join* consistency models. Updates to real-time shared objects are subject to the weak consistency constraints. Only the last update will be seen by the latecomer. All other updates are part of the history transferred and replayed by COCOON.
- Each REVIEWER can disconnect from the session (e.g. communication link becomes unavailable or too expensive). Disconnection can be forced if communication with the respective member is no longer possible. In both cases, the disconnected REVIEWER can browse through the code listing, make annotations and vote on his own. He can later reconnect to the session in two ways : a *join* to the session uses the COCOON *disconnect-reconnect-join* consistency model. All updates made from the disconnected REVIEWER during the disconnection period are lost. Updates from the group

are transparently replayed. A REVIEWER can also *merge* with the current session. This maps to the *disconnect-reconnect-merge* model. The conflict resolution method provided by the application uses the *group-has-it-right* policy described in chapter 3.2. This ensures that annotations made by both the group and the singleton will be part of the shared information space after the join while conflicting updates (changes of the code window or pointer object) will be reflected according to the group state.

- The application allows each REVIEWER to make private comments or view additional documents (e.g. compiler listing, man pages) independently from the group. This uses the private information space facility provided by COCOON. Private comments can be made available to the group at any point during the group session by transferring the private space to the shared information space.

5.2 CoNus: an Application to support synchronous and asynchronous collaboration

We implemented an application that provides team members with the ability to visit each other in virtual rooms. If more than one member is present in such a room, informal audio video communication is enabled among them. If an application document is layed on an overhead projector in the user interface metaphor, a shared window system shares this application among the room participants as illustrated in Fig. 5.
We do not describe the application in full detail here. The following list gives a brief overview of its functionality with respect to the used COCOON abstractions

- When the CoNus system is coming up the first time, an electronic secretary initiates an overview. For each user connecting the first time to the system, a private room object is created together with an associated room session. The system distinguishes between tightly and loosely subscribed room sessions. Tightly subscribed means that each change inside a room as for example the creation of a door to another room, the creation of an overhead projector on a table, or the movement of an application document from the table to the overhead projector is tracked by the room session according to the COCOON *total consistency* model. This guarantees e.g., that a document can not be layed on a table that has not been created before. In a second step we will relax the consistency model to use the *causal consistency* model. Loosely subscribed means that changes to that room are not completely tracked by the room session: only users that enter and leave the room are recognized.

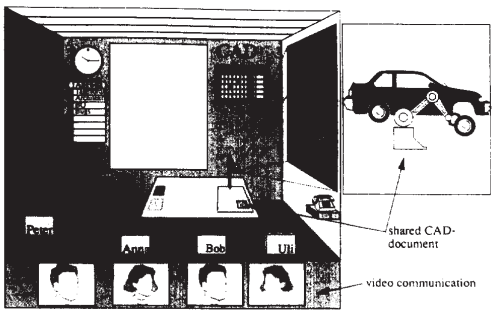

Figure 5: The CoNus Metaphor

- If a user is visiting another user in his room, the tightly as well as the loosely subscribed room sessions track the users disappearence/appearence. There are two variants in preparing consistent replicas of the shared room for the users:
  var a) Loosely subscribed: All the changes that have been made since the last visit of the room will be replayed for the visitor according to the COCOON *disconnect-reconnect-join* consistency model. To reduce message traffic this is appropriate especially if a large number of partners are using the system.
  var b) Tightly subscribed: All the changes have already been performed by the system, because the users session membership remained unchanged. This is appropriate if users frequently visit each other and do not accept the delay for actualizing the room state.

- A user can disconnect from the session (e.g. if he carries his laptop home) in two ways according to var a) and var b):
  a) Before leaving the CoNus-system, he checks out from all concerned rooms with *reconnect-join* consistency model. This updates the rooms

state.
b) No check-out is required for those rooms that have been connected since the last visit.
Thereafter the consistency model switches to *disconnect-reconnect-merge*. Disconnected from the CoNus system, the user is able to work autonomously in private as well as in shared rooms: he can create annotations, documents, room equipments, as well as change the contents of specific objects. The user later merges the session according to the disconnect-reconnect-merge model. The conflict resolution methods provided by the CoNus system are still under development. Possible conflict policies are : *group-has-it-right* or *chief-is-always-right* policy for objects that have been moved around, *solve-conflict-by-originators* policy delegates responsibility to those users that have changed contents of the same objects.

- The *atomic consistency model* is required e.g. if a user enters a room: Between the room change operation delLink (user1, room1) and createLink (user1, room2) no other operations are allowed, because the consistency criterion *each user has an associated room* would be violated.

- The *weak consistency model* is used for real-time-shared objects: The users telepointers and objects that have bee moved around with no change of their part-of-structure (e.g. a document has only be moved on the table).

Allthough the applications show a variety of different collaboration patterns, we were able to verify, that the implementation was very much faciliated by COCOON. The shared information abstraction is generic enough, to support a variety of shared objects typical for a code inspection meeting and the shared rooms scenario. The consistency models map to the basic collaboration patterns in theses meetings. The COCOON implementation on top of the ISIS/News service and RDO/Smalltalk interface proved to be sufficiently efficient even for real-time shared objects. However the conversion of large objects from/into the ISIS message format turned out to be a limiting factor for the scalability of history lists, especially when large objects are envolved. A hybrid approach using checkpoints of the object net in combination with history lists may be a solution in this case. Checkpointing functionality is also important for undo/redo functionality and persistence support. These issues will be addressed in the future. Note that the simplicity of the COCOON information model and its implementation allow a simple and elegant realization of these mechanisms.

6 Conclusion

In this paper we presented COCOON, a groupware support layer, centered around the notion of a shared information space. The concept of a shared information space is inherent in pretty much all groupware applications. Our approach corresponds to this by providing a simple yet powerful and generic model of managing shared information in a collaborative setting. COCOON supports a variety of real-world cooperative tasks. It simplifies the task of a groupware programmer and makes implementation faster and less complex.

COCOON supports a replicated architecture and thus corresponds to major requirements in groupware application design, like heterogeneity of local environments, responsiveness and support for mobile, location independent computing.
The COCOON consistency models for managing shared information are based on an analysis of a variety of collaborative activities. They offer a broad range of options for the groupware programmer, hiding lots of the complexities in dealing with replicated information. In contrast to existing approaches, theses consistency models reflect the requirements of groupware applications, allowing the programmer to focus his effort on application functionality.
Using the RDO/Smalltalk interface to the ISIS/News service proved to be extremly advantageous. It simplified the implementation of COCOON considerably. In fact it turned out that the publish/subscribe model provided by ISIS/News is extremely well suited for the support of groupware scenarios.

Allthough the shared information abstraction is a key concept in groupware applications, there are a number of other abstractions which are used extensively in the support of collaborative tasks. In order to provide a fully generic groupware support layer, our future work will extend COCOON with a number of mechanisms :

- Support for additional consistency models [1] [2].

- Support for coordination of activities, e.g. for work-flow management [15].

- Extension of the information model to support the notion of fragmented objects and applicable consistency models, e.g. integrity constraints or atomic operations [14].
- Support for Undo/Redo of changes to the shared information space.
- Support for Peristence and Histories.

A first investigation showed that the COCOON information model and its implementation via ISIS/News makes support for these things relatively simple. Additionally we will test the strenghts and weaknesses of COCOON by implementing a suite of groupwre applications on top of it.

References

[1] The Association for Computing Machinery. *Proceedings of the 1992 ACM Conference on CSCW - Session on consistency in collaborative systems*, Toronto, November 1992.

[2] W. Babitch. *Software Configuration Management.* Addison Wesley, 1986.

[3] K. Birman. Lightweigth causal and atomic group multicast. *ACM Transactions on Computer Systems*, 8(3), 1991.

[4] K.P. Birman and other. *The ISIS Distributed Toolkit, Version 3.0.* ISIS Distributed Systems Inc., September 1992.

[5] L. Brothers and other. Icicle - groupware for code inspection. In *Proceedings of the 1990 ACM Conference on CSCW*, Los Angeles CA, October 1990.

[6] V. Cahill and other. The Amadeus GRT - Generic Runtime Support for Distributed Persistent Programming. In *Proceedings of the 1993 OOPSLA Conference*, Washington D.C., 1993.

[7] R. Cooper and other. *The RDO Programmers Manual.* ISIS Distributed Systems Inc., August 1993.

[8] C. Ellis and other. Groupware - some issues and experiences. *Communications of the ACM*, 34(1), 1991.

[9] S. Greenberg. *Computer Supported Cooperative Work and Groupware.* Academic Press, 1991.

[10] S. Greenberg and other. Liveware: A new approach to sharing data in social networks. *International Journal of Man Machine Studies*, 34(3), 1991.

[11] S. Jablonski. *Datenhaltung in verteilten Systemen.* Informatik Fachberichte. Springer Verlag, 1991.

[12] L. Lamport. Time, Clocks and the ordering of events in a Distributed System. *Communications of the ACM*, 21(7), 1978.

[13] Golding, R. Long, D. Design choices for weak consistency group communication. Technical Report UCSC-TR-92-45, University of California Santa Cruz, October 1992.

[14] M. Makpangou and other. Fragmented Objects for Distributed Abstractions. In *Readings in Distributed Systems.* IEEE Compouter Society, 1993.

[15] R. Medina-Mora and other. The Action Workflow Approach To Workflow Management. In *Proceedings of the 1992 ACM Conference on CSCW*, Toronto, November 1992.

[16] R. Needham. Naming. In *Distributed Systems.* ACM Press, 1989.

[17] R. Newman-Wolfe and other. Implicit locking in the ENSEMBLE concurrent object-oriented graphics editor. In *Proceedings of the 1992 ACM Conference on CSCW*, Toronto, November 1992.

[18] T. Rodden and other. A survey of CSCW systems. *Interacting with Computers*, 3(3), 1991.

[19] Greif, I. Sarin, S. Data Sharing in Group Work. *ACM Transactions on Office Information Systems*, 5(2), 1987.

[20] M. Satyanarayanan and other. Experience with disconnected operation in a mobile environment. In *Proceedings of the USENIX Symposium on Mobile and Location Independent Computing*, Cambridge, MA, August 1993.

[21] Davidson, S. Garcia-Molina, H. Skeen, D. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3), September 1985.

[22] M. et al Stefik. Beyond the chalkboard - computer support for cooperation and problem solving in meetings. *Communications of the ACM*, 30(1), 1987.

[23] R. van der Linden. The ANSA Naming model. Technical Report AR.0003.01, The ANSA Consortium, Cambridge (UK), February 1993.

[24] Coad, S. Yourdan, M. *Object-Oriented Design and Analysis.* Prentice-Hall, 1991.