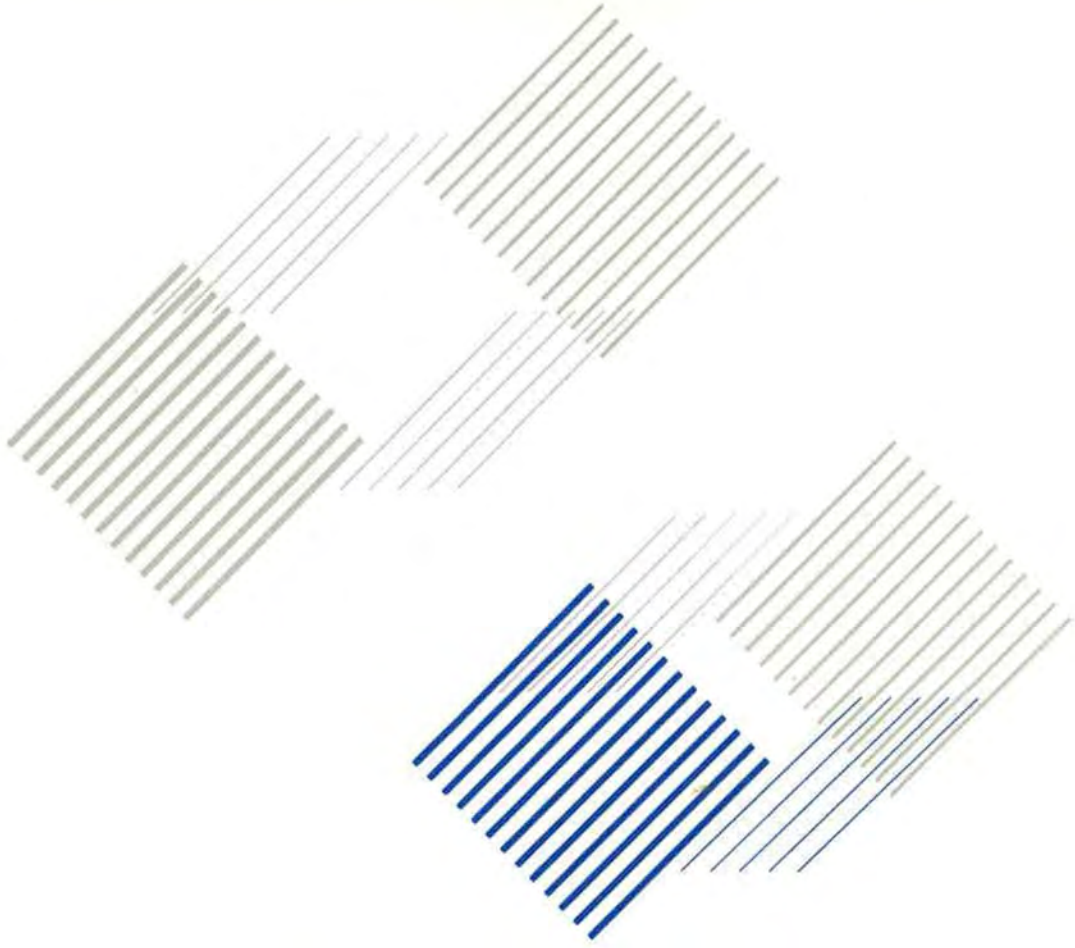
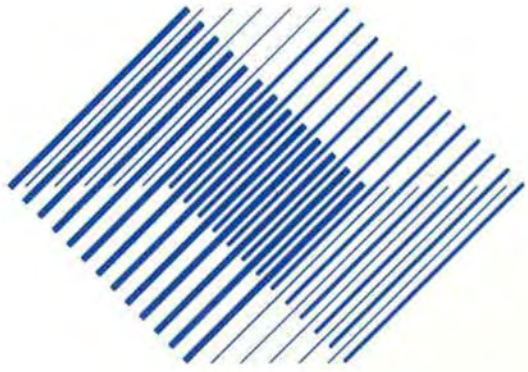


Understanding

The ICAD System



Understanding The ICAD System

By Martin R. Wagner
ICAD, Inc.

Trademarks:

ICAD	The ICAD System, ICAD Design Language, IDL
Shape Data, Ltd.	Parasolid
Sun Microsystems, Inc.	SPARCstation
AT&T	UNIX
Silicon Graphics, Inc.	IRIS
Adobe Systems, Inc.	PostScript
Oracle Corporation	ORACLE
Computervision, a Prime Computer Company	CADDS4X
McDonnell Douglas Systems Integration Company, Manufacturing and Engineering Systems	UNIGRAPHICS II
Bentley Systems Inc., an Intergraph affiliate	MicroStation
Dassault Systemes	CATIA
Autodesk, Inc.	AutoCAD

The trademark information listed above is,
to the best of our knowledge, accurate and complete.

ICAD, Inc.
201 Broadway
Cambridge, Massachusetts 02139
(617) 868-2800

Table of Contents

Chapter 1

About *Understanding The ICAD System*

Who Should Read *Understanding The ICAD System*

Understanding The ICAD System is for all members of the engineering and manufacturing team, including managers, who are considering using The ICAD System. The ICAD System, ICAD's knowledge-based engineering product, is used to develop knowledge-based engineering applications. *Understanding The ICAD System* introduces and defines the terminology of The ICAD System, examines some of the techniques used to develop an application using The ICAD System, and discusses some of the strategies used to integrate such an application into an existing engineering environment.

Understanding The ICAD System answers the following questions:

- What is The ICAD System, and what differentiates it from other computer-aided-engineering design tools?
- What types of engineering problems can be automated using The ICAD System?
- What are the capabilities of The ICAD System and how can they be used to create solutions to complex, large-scale, real-world engineering problems?
- How can an ICAD knowledge-based engineering application be integrated into an existing engineering and computing environment?

In a sense, *Understanding The ICAD System* is a technical overview of The ICAD System. Since the concepts of using The ICAD System are new and unfamiliar to most engineers, the essential features of The ICAD System are not described on a feature-by-feature basis as with most technical overviews, but rather as they apply to a particular application.

We show in detail how The ICAD System is used to create an engineering application that automatically designs mold bases for plastic injection molding. Although the concepts of knowledge-based engineering and The ICAD System are described in the context of a mold-base design, these same concepts are applicable to many engineering domains, and have been used successfully for many different kinds of products in many different industries.

About *Understanding The ICAD System*

Chapters

Understanding The ICAD System is divided into the following chapters:

"Introduction" Discusses the purpose of *Understanding The ICAD System*, gives a brief history of ICAD, Inc., and describes the support that ICAD gives its customers to help them become successful users of The ICAD System.

"Knowledge-based Engineering and The ICAD System"
Provides an overview of knowledge-based engineering and lists some of the benefits realized using such a system. This section includes a definition of knowledge-based engineering, a comparison of knowledge-based engineering with other computer-aided engineering technologies, and a discussion of different kinds of applications that have been developed successfully using The ICAD System.

"Example of an Engineering Problem"
Describes the problem of designing a mold base for plastic injection molding. The rest of the document shows how The ICAD System applies to this problem.

"Modeling an Engineering Problem Using ICAD"
Describes the capabilities of The ICAD System by showing how they are used to develop a mold-base application.

"Integrating ICAD Into an Existing Engineering Environment"
Shows how an application created using The ICAD System can be integrated into an engineering environment that includes various analysis systems and CAD systems.

"Appendix" Includes a glossary of concepts and terminology used by ICAD.

About ICAD

ICAD Inc. develops and markets The ICAD System, an advanced knowledge-based engineering software package for mechanical product and process design automation. At the end of 1989, ICAD Inc. had installed more than 300 ICAD systems in manufacturing and engineering firms throughout the world. The company's list of customers includes a number of leading manufacturing companies in many different industries, such as automotive, aerospace, industrial equipment, consumer products, etc.

In 1984 the founders of ICAD Inc. began the development of The ICAD System under a consulting contract with a major CAD company to provide a system that would automate the design of heat exchanger units for a major industrial equipment manufacturer. The manufacturer recognized that although all heat-exchanger designs were based on similar engineering principles, each new individual design had to be completely redeveloped.

The manufacturer had attempted automating the design of heat exchangers using a traditional CAD/CAM system and a FORTRAN program that added a parametric design capability. As they began programming the design process, they recognized this would require a much greater programming effort than originally anticipated.

ICAD's founders addressed this issue by developing The ICAD System, which uses a knowledge-based engineering approach and object-oriented programming techniques. The company's founders shared the vision that new software technology provided an opportunity to re-think the conventional approach to computer aided design, which is based on computer representation of drawings. Their experience with CAD systems convinced them that to do more than help edit drawings, the system would require a complete model containing all the factors that determine the design, including company standards, procedures, and processes. With such a model, a computer system can integrate the entire design and manufacturing process, constructing correct designs on demand.

The early version of The ICAD System successfully automated the design of heat exchangers. Since that time, many other applications in different industries have been automated successfully as well. ICAD's comprehensive approach to engineering automation has enjoyed market acceptance, enabling the company to grow rapidly.

Learning to Use The ICAD System

ICAD provides you with the support you need to develop successful applications using The ICAD System. We provide various kinds of assistance and training to enable you to change from a newcomer to knowledge-based engineering design to an experienced ICAD Design Language designer. Our goal is capability transfer: we transfer our expertise to you so that you become an effective user of our tools.

- *Intensive training*

  The basic course, *Intensive Introduction to the ICAD Design Language*, includes both lectures and labs. In addition to learning the basics of The ICAD System, you learn techniques that help you select and execute a complex, real-world project in conjunction with your ICAD consultant. By the time the basic course has been completed, many customers have already completed a project with significant real-world savings. ICAD also provides a course on using the ICAD Surface Designer.

- *Expert consultants*

  During training you are assigned a personal consultant to help you plan your application and break through learning barriers. ICAD's consultants are experienced mechanical engineers who have worked extensively on knowledge-based engineering projects.

- *Comprehensive documentation*

  The ICAD System is completely and thoroughly documented. The documentation includes a section on every ICAD Design Language feature including a description of the feature, a syntax diagram, and many complete, working examples of use. In addition, on-line help is available in the ICAD user interface, the ICAD Browser.

- *Extensive customer support*

  Customer support is available via telephone, FAX, or electronic mail to answer any technical question or resolve any technical problem with installation, use of The ICAD System, or use of the hardware platform.

Chapter 2

Knowledge-based Engineering and The ICAD System

What is Knowledge-based Engineering?

Keywords: Knowledge-based engineering (KBE), product model, ICAD product model, rule-based system, concurrent engineering, parametric CAD systems, variational CAD systems, engineering-automation languages, expert systems

Overview

This section introduces the concept of knowledge-based engineering and includes the following topics:

- A definition of knowledge-based engineering.
- A comparison of knowledge-based engineering with the traditional design process.
- An introduction to the product model, including typical engineering rules included in a product model.
- A discussion of some of the benefits realized from using knowledge-based engineering.
- A comparison of knowledge-based engineering with other technologies such as traditional CAD systems, modern parametric and variational CAD systems, engineering-automation languages, and expert systems.

Definition of Knowledge-based Engineering

The ICAD System is a *knowledge-based engineering system*. Knowledge-based engineering, sometimes referred to as KBE, is an engineering method in which knowledge about the product, e.g., the techniques used to design, analyze, and manufacture a product, is stored in a special *product model* (in The ICAD System, the product model is called an *ICAD product model*).

The product model represents the engineering intent behind the geometric design; it captures the how and why, in addition to the what, of the design. It can store product information — attributes of the physical product such as geometry, material type, functional constraints, etc. — as well as process information — the processes by which the product is analyzed, manufactured, and tested. Unlike a CAD model which mostly contains geometric information, a knowledge-based engineering product model contains *all* the information required by the design.

Once engineering knowledge about the product is collected and stored as a product model, design engineers can generate and evaluate new designs quickly

What is Knowledge-based Engineering?

and easily by changing the input specifications for the product model, or modify designs by extending or changing the product model. This frees the engineer from time-intensive, detailed engineering tasks such as repetitive calculations and allows more time for creative design work.

The Traditional Design Process Based on a Well-understood Design Strategy

Consider the design of a crankshaft by an engineering team at an automotive company. The team is given requirements such as horsepower, number of cylinders, torque, and rpm, and is expected to design a new crankshaft that minimizes cost and maximizes fuel economy. Assume that collectively the engineers have a clear and well-understood strategy for designing crankshafts, that is, they have already designed many crankshafts. However, their collective experience is distributed among a group of people with varying ability and covering several engineering domains.

Using traditional design methods in conjunction with CAD technology, the team integrates the input specifications with the constraints and stated goals. All the collective previous experience of designing crankshafts is manually incorporated into the new design. While the CAD system can help design the product geometry, it provides little help for calculating the engineering design attributes such as the required bearing load based on the number of cylinders, horsepower, and torque, or for taking into account other factors such as material constraints, cost considerations, and manufacturability.

Once a preliminary design is created, it is frequently analyzed using one of the many available FEA (Finite Element Analysis) programs and modified accordingly. Finally, the team generates drawings and reports that document the crankshaft design.

Even with all of engineering team's experience, the process of developing a single design is laborious, repetitive, and subject to error. If the input specifications change at any time, the entire process must be repeated. For these reasons, a new crankshaft design typically takes from four to six months to complete.[1]

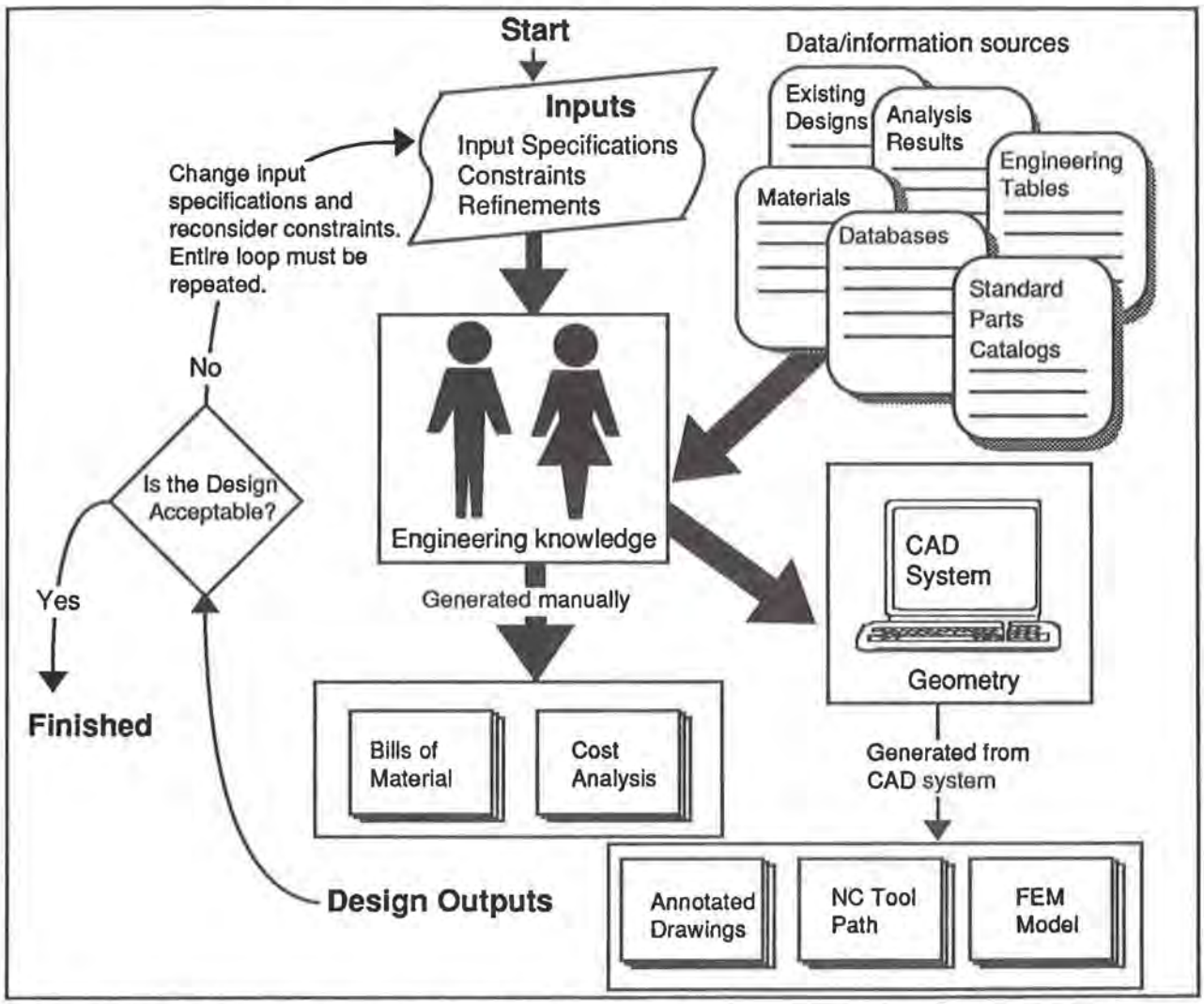
This scenario is typical of the traditional design process, in which engineers use CAD software to create, manipulate, and display geometry-based product-design data interactively. Traditional CAD systems allow engineers to process designs faster than when designs were done completely by hand, but go only part of the way toward providing a completely flexible tool for adapting to changes in a product design. No matter how sophisticated the software tools are, systems that use only geometric data to describe a product are limited in their ability to automate the design and engineering process since they provide no means of capturing the engineering expertise behind the design.

---

[1]According to a current ICAD customer.

What is Knowledge-based Engineering?

The following diagram illustrates this traditional design process. Since the repetitive, detailed engineering tasks are not automated — they need to be done by humans — they create a bottleneck that slows down the design cycle. While the engineers use CAD systems to manage the geometric information and generate drawings, other design knowledge, including catalogs, databases, and engineering standards, must be managed separately without integrated tools. Finally, since this design process is really a loop which repeats until a satisfactory design is found, the effect of the bottleneck is multiplied by the number of times the loop is repeated.



The traditional design process. The design bottleneck – the time-consuming, repetitive parts of the design process – is shown by using heavy lines. Based on inputs and engineering data and information, the engineer generates geometric outputs on a CAD system and manually creates non-geometric outputs such as BOMs and cost analyses.

What is Knowledge-based Engineering?

The Same Design Process Using a KBE System

Suppose this engineering team had previously used The ICAD System to develop a product model of a crankshaft that automates much of the routine and detailed engineering work.[2] If a new request for a crankshaft comes in, a design engineer on the team just feeds the new input specifications directly into the product model. The model asks the engineer some key questions about the application, and then generates a design that incorporates all of the input specifications, meets the constraints, and maximizes fuel economy while minimizing production costs. The design is sent to a FEA automatically, and the design engineer can easily modify the design either by changing the answers to some of the key design questions or by modifying the input specifications.

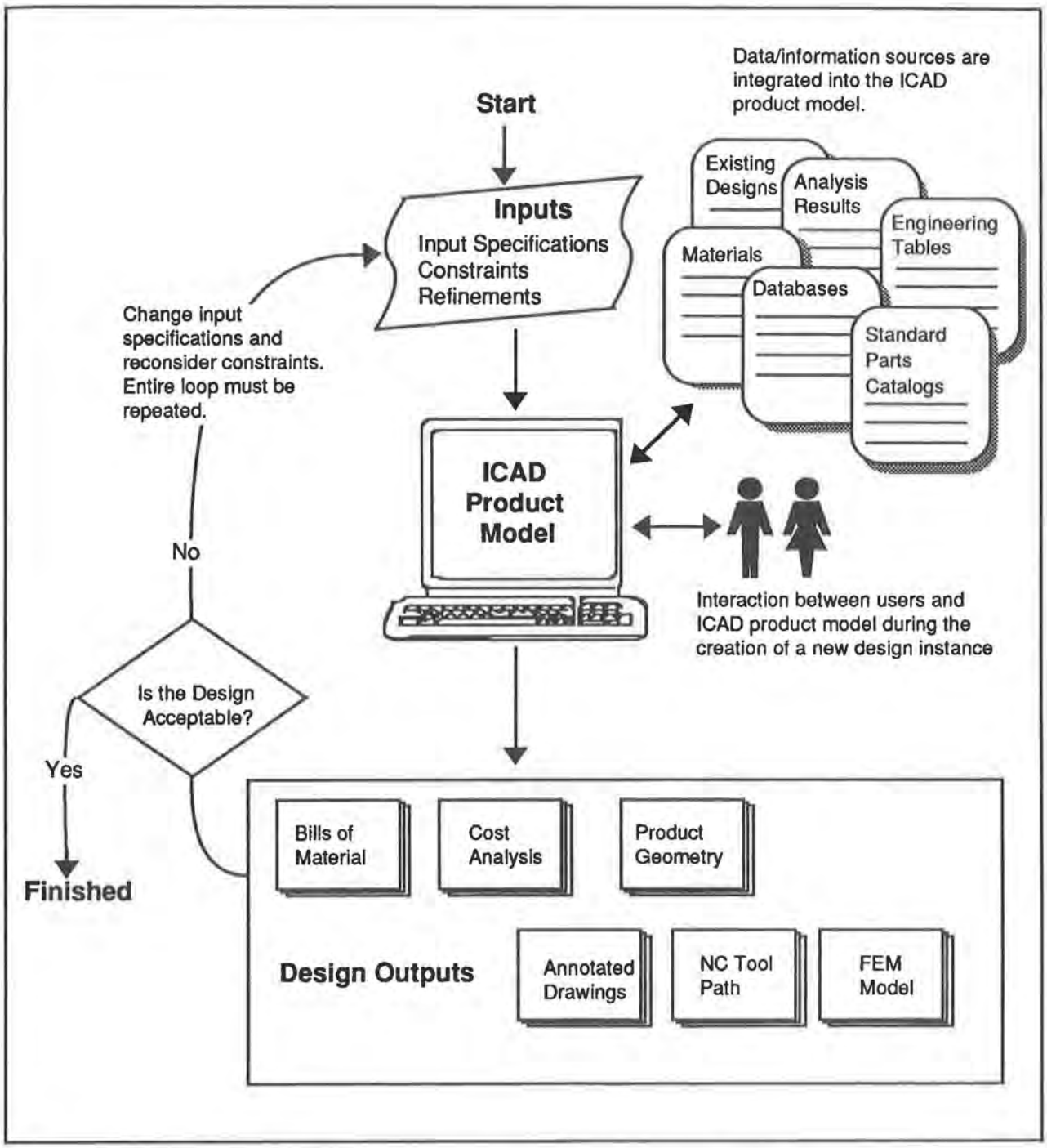
Finally, the result already contains the document set for the crankshaft, e.g., the drawings, reports, and manufacturing plans. A more extensive ICAD product model could also produce a tooling plan. New inputs can be used to regenerate a new design easily. With the product model, generating a new crankshaft design can be done in two days by a single design engineer, as opposed to four to six months with traditional methods.

In this scenario, the bottleneck of the traditional design process — the repetitive tasks that had to be done by an engineer — is eliminated. All such tasks are managed by the product model. The design loop can be repeated efficiently as many times as necessary.

The following diagram illustrates the design process using a product model. The center of the design process is the product model, which manages all the routine engineering tasks. The role of the engineer is to provide the input specifications and make the important design decisions. Catalogs, databases, standards books, etc., are all integrated into the product model. Outputs are all generated automatically by the product model.

[2]The effort involved in developing a product model is discussed later in this section.

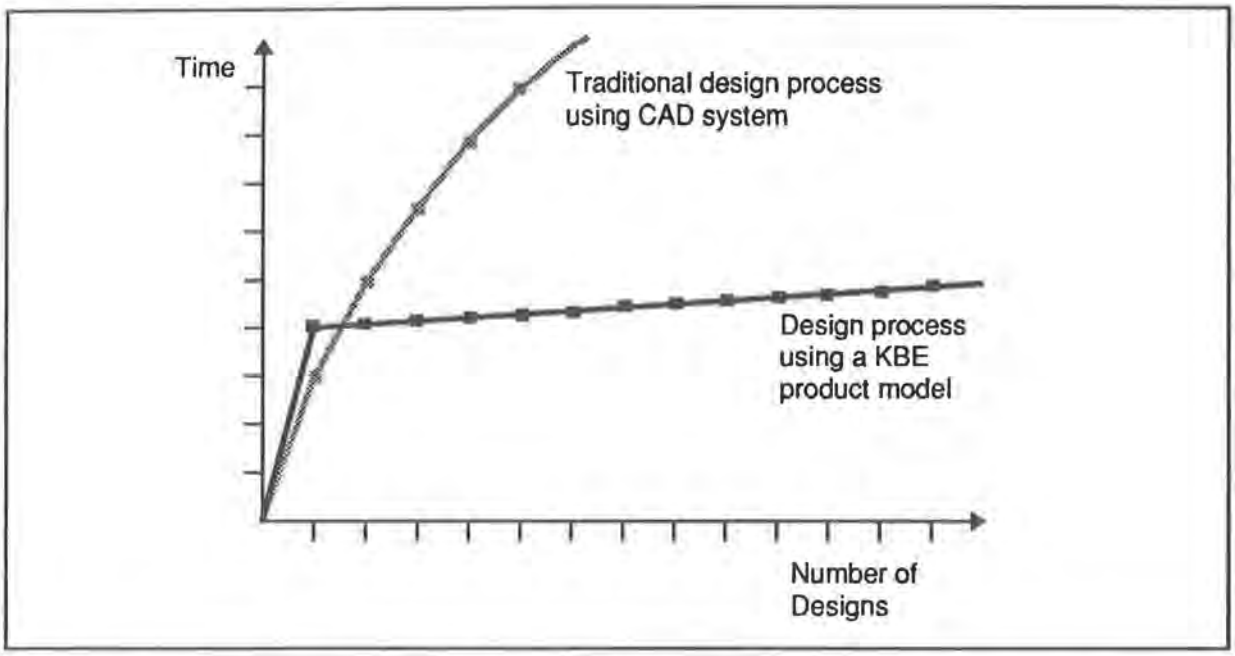
What is Knowledge-based Engineering?



The design process using a KBE product model. The repetitive parts of the design process are managed by the product model, which eliminates the design bottleneck.

What is Knowledge-based Engineering?

Clearly, there is a certain amount of effort involved in developing the product model. In practice, development time varies by application and by required level of detail; typically, it takes slightly more time to develop a product model than it does to create a single design using the traditional design process.[3] The time benefits of developing a product model compared with using traditional design techniques are shown in the following diagram.



Time benefits of using KBE compared with the traditional design process. The squares represent new designs.

Note that even a partially-developed product model shows substantial time benefits since the bottleneck can be widened incrementally.

Using KBE for a Design Process Based on a New Design Strategy

Consider another engineering team that is working on a new, more efficient crankshaft based on some recent theoretical or experimental research. Since this is a new type of crankshaft, they are developing a completely new strategy for

[3]Interestingly, the more complex the design, the *less* time it takes to develop an ICAD product model relative to the time it takes to produce a single design iteration using traditional design techniques.

What is Knowledge-based Engineering?

designing such a crankshaft. There are still many benefits to using a knowledge-based engineering approach even though the design strategy is not well understood.

Using a traditional design process, the team develops an initial design strategy which they use to generate an initial crankshaft design. This first design is not completely successful, so they change their design strategy and produce a new iteration. However, it takes almost as long to produce the second iteration as the first, since most of the calculations for important factors such as bearing width, joint configuration, counterweight layout, etc., have to be completely redone. Finally, after many time-consuming design iterations, the team arrives at the final design.

However, if the team uses a knowledge-based engineering system such as The ICAD System to develop a product model that represents their initial design strategy, the first design can be generated from the product model. Subsequent changes to the design strategy can be made by directly modifying the product model, which can then be used to generate new design iterations. Much of the work done on the previous design iteration can be automatically carried over to the new design iteration. This leads to substantial overall time savings. Likewise, since each design iteration takes less time, many more iterations can be generated before the design is finalized, which leads to a better quality design.

Creating a Product Model by Developing a Rule Base

A product model captures the design strategy required to produce a particular product from a specification — in essence, it is the set of engineering rules used to design the product. For this reason, a knowledge-based engineering system is often called a *rule-based system*, and the product model is often called the *rule base*. This set of rules includes standard engineering rules and experiential "rules of thumb" that constrain the design, information about manufacturing and cost constraints, catalogs of standard materials, parts, and costs, and techniques to maximize quality while minimizing cost and production time. You are not limited to rules involving geometry; in fact, you can incorporate any knowledge about a product design into a product model, including knowledge about the processes used to manufacture the product.

The product model takes an input specification, applies the engineering rules, and generates a product design. This product design can include various outputs, such as reports of engineering results, data for engineering analyses, 3D geometric models that can be downloaded to a CAD system, structured bills of materials, and manufacturing instructions.

What is Knowledge-based Engineering?

Product models often include the following kinds of engineering rules:

- *Rules that automatically create the part's geometry from the input specifications.*

  A rule might relate the diameter of an axle to the torque and length. Also, a length of shaft within an axle could be sized to fit between two load-bearing components whose size and position are based on engineering constraints.

- *Rules that calculate engineering properties about the part.*

  A rule might calculate the maximum load a particular beam can handle based on the shape of the beam and its material. Alternately, the selection of beam size and material could be based on the required maximum deflection.

- *Rules that choose a configuration based on some condition.*

  A rule might choose one configuration for a component within an axle if the axle is above a particular size or weight, and another configuration if the axle is below the size or weight.

- *Rules that optimize cost, performance, and quality.*

  A rule might specify that the material used to construct a crankshaft be the least expensive material whose properties allow it to meet some input constraints.

- *Rules that extract information from external databases.*

  A rule might look up the advertised strength of a material from a material table. Typical applications extract standard parts from catalogs, material properties from material tables, and design features from feature libraries.

- *Rules that communicate with and analyze the results of external engineering analysis programs.*

  A rule might send a particular beam description to an in-house beam analysis program for analysis. Other rules might use the results of the analysis as a basis for changing aspects of the beam design.

What is Knowledge-based Engineering?

There are a few important qualities of rules in a knowledge-based engineering system that make them particularly well suited for typical mechanical design problems:

- *Rules allow the direct implementation of standards.*

  Corporate and regulatory standards and "design rules" form a significant part of common engineering practice. These rules can be implemented directly in the product model, resulting in product designs that always conform to these standards.

- *Rules record design intent.*

  Product models provide a clear reason for every dimension, design decision, and configuration. This information is invaluable to the designer and to those who review the design.

- *Rules do not have to be exact.*

  Many real-world problems do not have exact theoretical solutions, yet "rule-of-thumb" techniques for handling these problems are well developed.

Benefits of Using a KBE System

Our customers have realized many benefits from using ICAD's knowledge-based engineering system. These include:

- *Reducing time to market by automating repetitive designs.*

  Although most new products are variations of current products, each new product design starts from scratch. However, similar products share many of the same routine engineering tasks; they are designed from similar components using similar engineering techniques, and are analyzed and produced in similar ways. Engineers must repetitively perform tedious (and sometimes inaccurate) calculations and look up numerous table and catalog entries. Much time is spent repeating analyses that optimize product designs. A knowledge-based engineering system helps to automate these routine engineering tasks for similar products by representing the knowledge of how to accomplish them as rules in a product model. This frees engineers to spend more time for creative designing.

What is Knowledge-based Engineering?

- *Capturing engineering expertise.*

  Some experience can be lost permanently when the key design people leave the company. A knowledge-based engineering system captures their design knowledge so that it is available in the future.

- *Facilitating concurrent engineering.*

  Typically, different engineering disciplines work separately and sequentially on the same product design. Any change to the design requires each engineering discipline to reevaluate their portion of the design. A product model synthesizes the constraints and engineering rules from all engineering disciplines; it checks that a design change made by one engineering group does not contradict the engineering rules and constraints from the other group.

- *Providing a fully documented design process.*

  In most companies, the "knowledge" of how to design and manufacture a particular process is scattered throughout the organization in the form of drawings of past designs, reports, design notebooks, engineering standards books, and the experience of a few key people. Often it takes much time and effort to reorganize this knowledge for a new design. This can cause many problems, including high engineering and manufacturing costs, slow response to customer requests, and bottlenecks caused by the unavailability of a few key people. A product model supplies a deterministic design process that generates consistent results.

- *Integrating catalogs and databases.*

  Engineers spend too much of their time searching for entries in catalogs and laboriously incorporating the results into their designs. A product model links to catalogs and databases and looks up the appropriate entries automatically.

- *Generating drawings and reports.*

  Another large part of engineering time is spent annotating CAD drawings and writing engineering reports such as bills of materials. A product model can include design rules that capture drawing intent as well as report specifications. These rules allow the product model to generate annotated drawings and reports.

What is Knowledge-based Engineering?

- *Integrating analysis tools into the design process.*

  Frequently, the design engineer uses software analysis tools to assist in solving specific problems. These tools are not easily integrated into the traditional design process since they require input in a specific format. Generating input in that format is difficult and time consuming. A product model can generate the inputs to the analyses routines directly as part of generating a new design. In addition, a product model can read the results of the analysis routines and use them to alter the design, forming a closed analysis loop.

- *Integrating design and manufacturing.*

  In many companies, manufacturing and engineering departments are physically separated and engineering design is often not integrated with manufacturing until the design is essentially complete. Any change to the design needed to meet manufacturing and process-planning constraints can require a long and costly redesign. Also, when engineering design and manufacturing analyses occur sequentially it is difficult to optimize the design for manufacturing, which adds to the cost of the product. A product model incorporates manufacturing constraints as well as engineering constraints, so that the product design is optimized for manufacturing. Both sets of rules can be developed concurrently by the design engineers and the manufacturing engineers, respectively.

- *Improving quality.*

  Under pressure to bring new products to market quickly, quality improvements for new products are often left incomplete. Since a product model accelerates the optimization process, higher quality products can be designed in the same amount of time or less than the time it takes to produce a design using traditional design methods. Increasing the number of design iterations almost always improves the design quality, since the designer now has the time and opportunity to produce "what if?" designs that experiment with the input specifications and design strategy. Additionally, many potential design mistakes are eliminated because of the consistency of generating a design from the same set of rules.

- *Lowering product costs.*

  Unnecessary product costs are often incurred when design changes cannot be made quickly and easily. Design and manufacturing engineering reports, such

What is Knowledge-based Engineering?

as part lists, process plans, drawings and documentation, maintenance, and routing sheets are often generated manually and must be manually changed when design changes are made. A product model automates the production of such reports, thereby lowering product costs.

- *Creating a centralized repository for mechanical design and manufacturing engineering information and experience.*

  A product model can include such information as:

  - Physical data such as geometry, size, location, material, cost.
  - Descriptions of the complex design rules, relationships, engineering standards, and procedures that are used to design and produce the product.
  - Maintenance, quality standards, and life-cycle support.
  - Manufacturing information including manufacturing constraints, manufacturing engineering requirements, tooling and fixtures, cost, process plans, record keeping beyond drawings, and performance specifications.

How KBE Differs From CAD Systems

CAD systems are personal-productivity tools used by designers and draftspeople to create geometry, annotated drawings, and documentation for a single design. A knowledge-based engineering system does not just create a single design; it models the process of generating the design. It is used in all levels of the design process to accelerate the design development.

In addition, CAD systems do not easily take into account non-geometric data. Although some CAD systems are flexible enough to annotate their geometric models with non-geometric information, their ability to interact with that information is very limited. It is difficult or impossible to represent objects that have no geometric components at all, such as processes. Furthermore, CAD systems are usually limited in the size and complexity of the objects that they can handle.

How KBE Complements CAD Systems

CAD systems are increasingly used in conjunction with knowledge-based engineering systems in an integrated engineering environment in the following ways:

- Once a design has been generated from the product model, the geometric information can be transferred to a CAD system for further detailing and

What is Knowledge-based Engineering?

drafting (if necessary), performing finite element analysis, or generating NC programs.

- Parts designed on a CAD system can be used as geometric-constraint specifications to a product model. For example, an existing CAD database may have a complex surface definition for the outer skin of an automobile door. The surface definition of the door can be part of an input specification to a product model that determines the shape and structure of various dependent parts such as the locking mechanism, window movement, and door internal structure.

How KBE Differs From Parametric and Variational CAD Systems

Parametric CAD systems capture geometric relationships between parts. Parts can be scaled easily by altering dimensions. Current parametric modeling systems include geometric features such as holes, slots and pockets that can be positioned relative to one another. Variational CAD systems allow the designer to specify omni-directional geometric constraints in 2D geometry. The capabilities of these technologies, while powerful in their own right, are quite different from the capabilities of a knowledge-based engineering system.

Unlike knowledge-based engineering systems, parametric and variational CAD systems do not automate the design process. They make design changes easier in the traditional process, but they do not replace the traditional design process. There are many other important distinctions between parametric and variational CAD systems and knowledge-based engineering systems:

- Parametric and variational CAD systems are geometry based; they cannot easily handle any other types of engineering knowledge. Our customers' experience suggests that less than fifty percent of the design rules in a product model are related to the geometry. Most of the design can only be described using a knowledge-based engineering system.

- Parametric instances (that is, designs created using parametric systems) are not generic in the same way a product model is generic. In a product model, the entire product configuration can change when input specifications change. This kind of change is impossible to parameterize in a parametric system.

- Parametric instances can only efficiently manage limited complexity (that is, the number of parts and relationships that they can incorporate is limited). In contrast, a knowledge-based engineering system is designed to manage large and complex product designs.

What is Knowledge-based Engineering?

- Although parametric and variational CAD systems can generate outputs, unlike a knowledge-based engineering system they cannot easily integrate an analysis program as part of a closed loop in which outputs are sent to the analysis program and the results are read back into the product model.

- Current parametric and variational CAD systems produce geometric outputs for the most part. A knowledge-based engineering system can produce many different kinds of outputs.

- Current parametric CAD systems are limited to solid modeling and only have limited free-form surface modeling capabilities.

How KBE Differs From Traditional Procedural-Design Languages

Many organizations have tried to implement design-automation programs using macro-programming languages associated with CAD systems (e.g., GRAPL), or standard computer languages such as Fortran or C. These efforts often prove unsatisfactory for a number of reasons:

- They require a high level of programming expertise, since the developers must concern themselves with programming details such as data storage management and data dependencies.

- While they may be adequate for small- to medium-sized tasks, it is too hard to use them to implement large-scale, complex automation projects.

- Once developed, such systems are difficult and expensive to maintain and update.

In contrast, the language used to implement a product model is a state-of-the-art object-oriented computer language that takes care of the storage management, data dependencies, and the procedural "flow" of the program. A product model is created the way engineers naturally describe their design. In fact, many successful ICAD customers had little or no computer programming experience before using The ICAD System.

Product models have successfully implemented many large and complex automation applications, including products in which the final design consists of thousands of parts.

Finally, a product model is relatively easy to maintain due to its modularity, English-like syntax, and declarative structure.

What is Knowledge-based Engineering?

How KBE Differs From Expert Systems

Expert systems attempt to capture design knowledge in an automated decision-making format, generally in the form of procedural "if-then" rules (e.g., "if the size is larger than 30 meters then use two support beams").

Many organizations have tried to implement design automation programs using expert-system shells. However, expert systems have limited use for a design-automation application for the following reasons:

- Typically, expert systems are not designed for mechanical engineering. For example, they do not have the built-in geometric primitives to describe mechanical products. In contrast, knowledge-based engineering systems include built-in geometric primitives for describing complex mechanical products.

- Expert systems work using a collection of procedural "if-then" rules. All engineering knowledge must be put in this format. However, not all engineering rules naturally fit into such a format. In contrast, a rule in a knowledge-based engineering system can take any form that the engineer likes.

- The efficiency of an expert system drops drastically as it gets more complex, since every rule must be checked with every change. To resolve this, expert-system shells allow you to define "meta-rules" that control the flow of operations. However, meta-rules are hard to define and to debug. In contrast, a knowledge-based engineering system is extremely efficient since the system automatically keeps track of which rules need to be evaluated, and only evaluates those rules that are needed.

Summary

- The ICAD System is the leading knowledge-based engineering system in the market today. Knowledge-based engineering is an engineering-design technology in which product and process knowledge about the product is stored in a product model as a set of engineering rules. A product model represents the way the product is designed, and can be used to accelerate the design process enormously by eliminating the bottleneck caused when engineers need to spend time on repetitive engineering tasks.

- A product model includes engineering rules that describe the product's geometry, optimize cost, performance, or quality, extract data from external data bases, etc. A product model takes an input specification, applies the engineering rules, and generates a specific product design.

What is Knowledge-based Engineering?

- Knowledge-based engineering is qualitatively different from all previous engineering-design technologies, and complements these technologies by managing large-scale and/or complex designs with longer design cycles.

Typical Knowledge-based Engineering Applications

Keywords: *semi-custom designs, complex one-of-a-kind products, generative process planning, configuration generation, feature-based design*

ICAD Applications

Although there is no "typical" ICAD application, there are various kinds of engineering problems that are well suited to knowledge-based engineering solutions. Likewise, there is no "typical" industry best-suited to use The ICAD System; ICAD's knowledge-based-engineering systems are installed in a cross-section of industries throughout the world.

The following chart lists some actual ICAD applications and shows both the kind of engineering problem that the application solves and the industry in which the application is used. The remainder of this section describes the kinds of applications best suited to a knowledge-based engineering solution, followed by short descriptions of actual ICAD applications.

	Semi-custom	One-of-a-kind	Generative Tooling	Generative Process planning	Product Configuration
Aerospace/ Defence		Satellite-antenna Reflectors Jet-engine turbine blades	Composite tooling	RAMP project	Electronic packaging
Automotive	Crankshafts Hood design	Exhaust manifold	Layout of die presses	Axles and halfshafts	Option-order processing
Consumer Products	CRT glass Prosthesis	Vending machines	Mold bases		HVAC configuration
Industrial Equipment	Heat exchangers Transformer layout		Punch and die tooling	Pipe-bending and end prep	Generators

Typical ICAD applications in various mechanical and manufacturing industries.

Typical Knowledge-based Engineering Applications

Semi-custom Designs

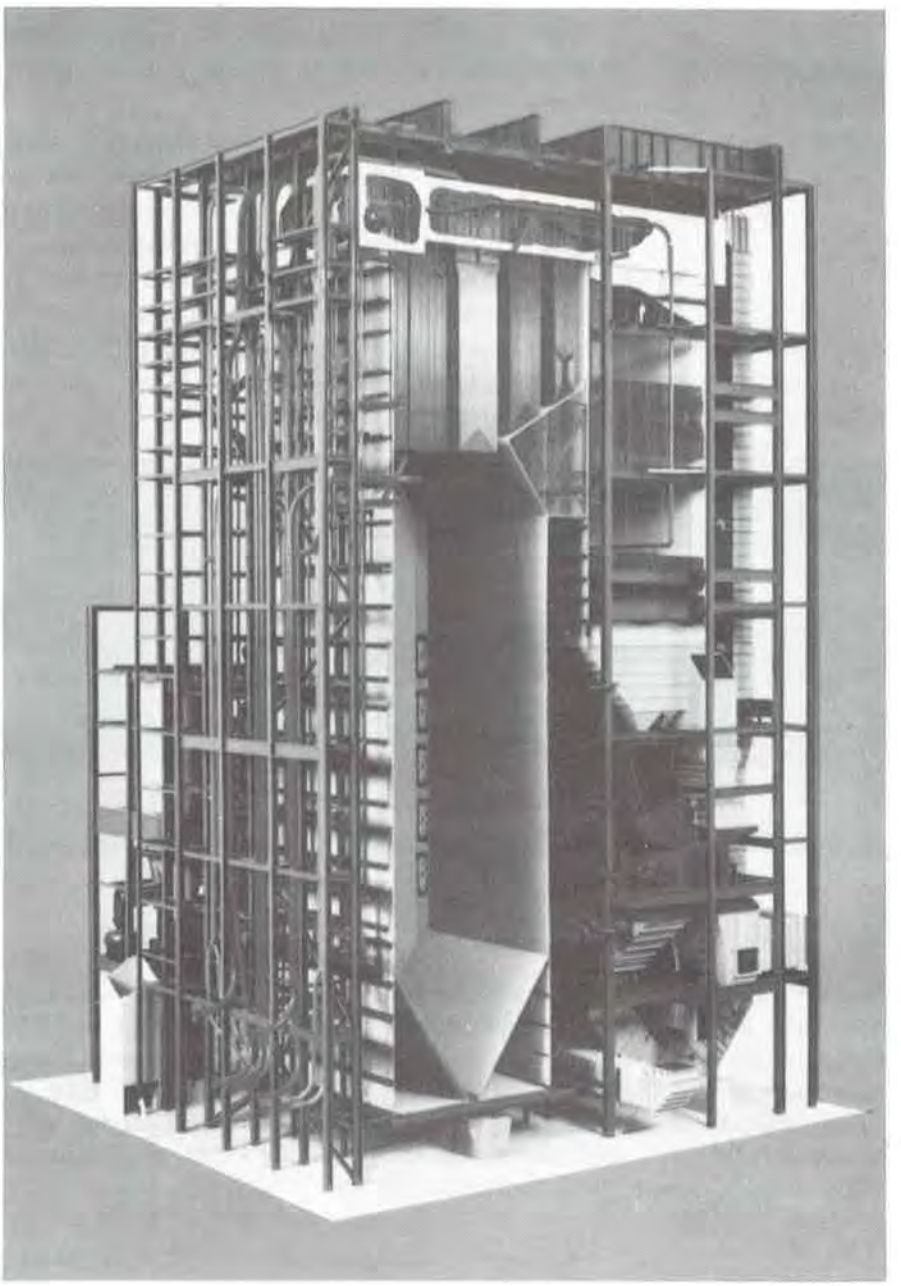
Many products such as gear assemblies, generators, presses, economizers, etc., are customized in order to satisfy particular requirements of each customer. The product is semi-customized since no two products are exactly alike, yet all the different products are very similar and require much the same engineering work to produce a design. An ICAD product model encapsulates the elements shared by all designs, and allows you to produce new designs automatically from input specifications.

- Boiler design

Problem: Although there is a basic design process for boilers which changes infrequently, developing a new boiler design can take a long period of time. It can take between 500 and 2 000 worker hours to create a design for an economizer, which is just one of the boiler's components. This is because each new boiler design is slightly different based on its size, complexity, and sensitivity to different fuels and operating conditions.

Solution: An ICAD product model is used to design various components of boiler systems, including the design of economizers. Rules for designing an economizer include component catalogs for tubes, fins, etc., geometric rules that decide where to place the components, engineering rules for designing and placing the mechanical supports, and manufacturing and costing rules that insure that the resulting design is manufacturable at a reasonable cost. Now an economizer engineer can enter customer requirements and, within hours, have an accurate design including drawings, bill of materials, cost estimate, and an engineering report.

Typical Knowledge-based Engineering Applications



Typical Knowledge-based Engineering Applications

- Prostheses

Problem: While off-the-shelf standard implants are acceptable for most patients, some patients require a customized implant, which is expensive to design.

Solution: Given an input specification from the surgeon (the type of implant, the dimensions of the bone and joint, and the size and weight of the patient) and geometric data from a CAT scan, the prothesis product model selects an appropriate standard implant. If no standard implant is available, the ICAD product model selects the nearest basic implant style and designs the necessary differences to accommodate the patient.

Complex, One-of-a-kind Products

A large, complex product such as jet turbine blades, car doors, antennas, etc., may take several years to design using traditional design techniques. After the design engineer specifies an initial design, other engineering groups must create other aspects of the design sequentially. Using The ICAD System, engineering rules from all groups are stored in the ICAD product model, all changes are automatically kept up-to-date, and design work from all groups can be done concurrently. Many more iterations of the design can be made, improving final product quality.

- Satellite antennas

Typical Knowledge-based Engineering Applications

Problem: The design of a satellite antenna is a highly iterative process. The design engineers must consider tradeoffs between weight, reliability, and volume.

Solution: Using an ICAD product model, design engineers can change design specifications to see how they affect the overall design. The automation of the design allows the company to evaluate many antenna possibilities and reduce the number of expensive prototypes.

The input specifications for the ICAD product model include the number of ribs, the beam diameter, the parabolic focal length, and the offset distance from the parabola axis. The antenna product model uses these specifications to generate a "pre-design" for the antenna which is passed to a FEA program. The refined data from the FEA is brought back into the ICAD product model, which then redesigns the antenna to the more exacting specifications.

Typical Knowledge-based Engineering Applications

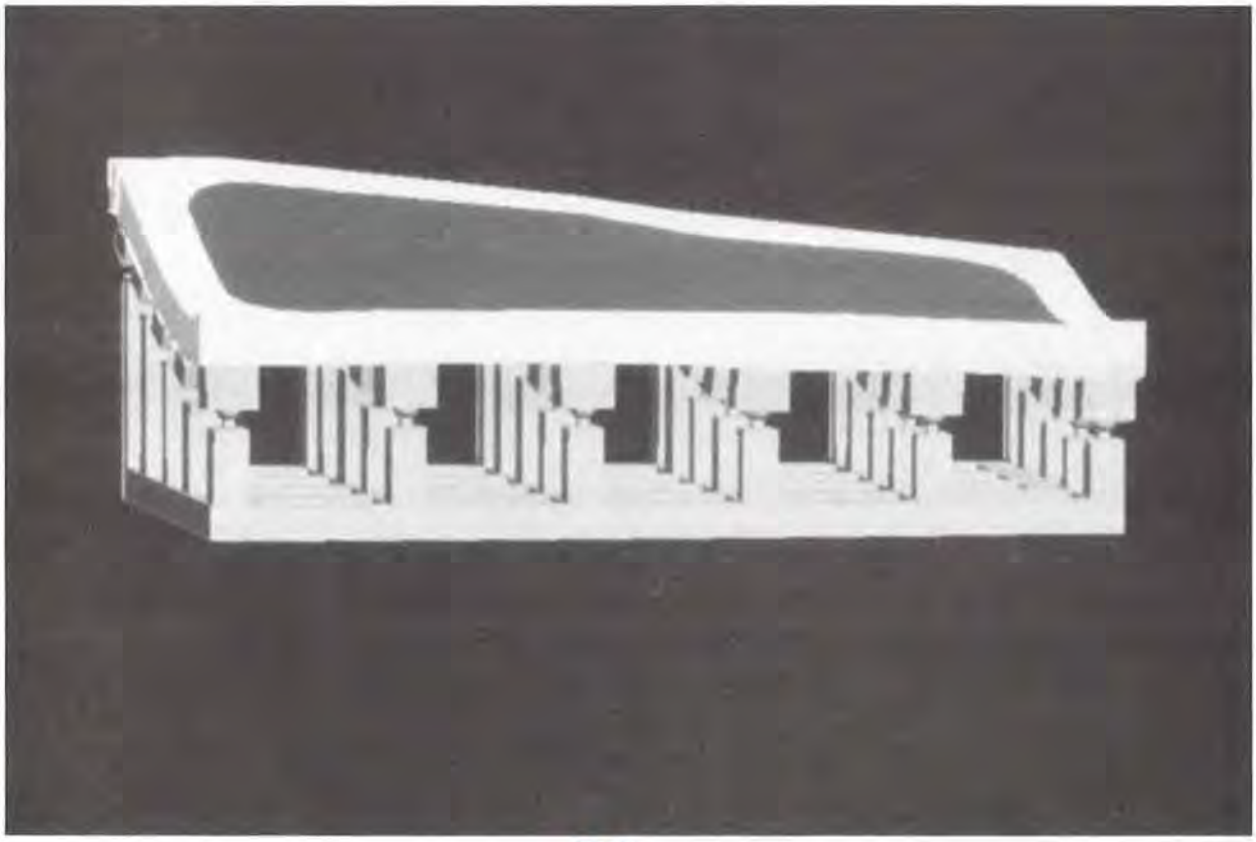
Generative Tooling

A generative tooling application takes an existing part design and designs a customized tool for manufacturing that part.

Composite tools for helicopter blades

Problem: A helicopter with composite rotor blades requires a large number of different tools. Tool engineers must consider varying geometry of the blade, different material types, and material thermal expansion and contraction factors.

Solution: An ICAD product model takes an existing blade design as an input specification, prompts the tool engineer for additional specifications, and uses that to create tools for the blade automatically. The ICAD product model maintains the same standards for each tool design.



Typical Knowledge-based Engineering Applications

- Automated Mold Bases

Problem:	A newly-designed plastic part requires a customized mold base for the injection-molding process. The mold-base designer chooses a standard mold base from a catalog, and then specifies how to customize the standard mold base for the part.
Solution:	An ICAD product model takes the shape of the mold insert and the number of parts to produce, selects a standard mold base, and produces a design for customizing the mold base which includes manufacturing and process data.

Typical Knowledge-based Engineering Applications

Generative Process Planning

ICAD applications can be used to generate detailed process plans for each new design.

- US Navy RAMP (Rapid Acquisition of Manufactured Parts) project

Problem: Typically, it takes the US Navy 300 days to replace a part in one of its ships. This is because there are many different kinds of ships, some as much as forty years old, with too many different kinds of parts to keep in stock. In many cases, even the tools for making the parts no longer exist. Replacement parts must be produced on demand. The objective is to cut the 300 days needed to replace a part down to 30 days.

Solution: An ICAD product model reads a PDES/STEP description of the part, and generates a process plan for building the replacement part. The process plan consists of routing sheets, operator and machine instructions, bills of material, and cost sheets.

Typical Knowledge-based Engineering Applications

Product Configuration

A product-configuration application can be used as a sales tool for order entry. Since the engineering costs for a semi-custom-designed product are unusually high and take an extended period of time, the sales department often must sell a product that has not yet been designed. In such cases, it is difficult to produce an accurate price estimate without going through a costly engineering cycle. A salesperson can use an ICAD product model to produce an accurate estimate at a fraction of the cost.

In addition, since there are often many constraints in a design that make certain configurations difficult or impossible, a salesperson might unwittingly sell a costly or impossible design. An ICAD product model can provide immediate feedback to the salesperson on the engineering and cost consequences of the customer's choices.

- Configuration of press machines

  Problem: Since there are many different applications of press machines, a high percentage of the press machines sold must be semi-custom designed in order to satisfy customer

Typical Knowledge-based Engineering Applications

requirements. Salespeople are often physically separated from engineers, and many simple mistakes and misunderstandings often cause a design to travel between sales, engineering, and the customer several times before the job is finished.

Solution: By giving the salespeople an ICAD product model that designs press machines, they can respond to each customer request more quickly and accurately. The engineers can concentrate on handling special cases and improving the design rather than on generating repetitious designs.

Summary

- There are many types of engineering problems well suited for ICAD solutions. In addition, there are many different industries that use ICAD applications.
- Typical applications include semi-custom design, complex one-of-a-kind products, tooling design, generative process planning, and product configuration.

Example of an Engineering Problem

Introduction

This chapter describes in detail the design and manufacture of mold bases for plastic injection molding, which is a particular engineering domain whose design process has been automated successfully using The ICAD System. The two chapters that follow show how the different capabilities of The ICAD System are used to develop an ICAD product model that automates the solution to some of the engineering problems in this domain. Chapter 4, "Modeling an Engineering Problem Using ICAD" shows how the mold-base design can be represented as an ICAD product model. Chapter 5, "Integrating ICAD Into an Existing Engineering Environment" shows how the ICAD product model can be extended so that it integrates with other systems in an engineering environment.

This ICAD product model is an extensive demonstration of using knowledge-based engineering technology for designing mold bases. We chose this project as a sample ICAD product model for the following reasons:

- The product domain is easy to understand.
- It models a complex and large scale "real-world" problem.
- It includes a generative process planner.
- It integrates many other engineering tools, such as catalogs, analysis programs and CAD output.

In addition, it highlights many of the features of The ICAD System in a simple, easy-to-understand fashion.

While your own application may seem quite different from mold-base design and manufacturing, ICAD's experience with diverse industries shows that the capabilities described in the later chapters are applicable to most engineering domains. We present the mold base as a sample application to show how The ICAD System can model a complex, large-scale, real-world engineering domain. Your own application will use the features of The ICAD System in a similar fashion.

About Mold-base Design

Injection Molding Process

Injection molding is a manufacturing process in which molten plastic is injected into a mold, hardened, and ejected as a finished product. A mold consists of two basic parts.

- The *inserts* contain the *cavities*, that is, the negative casts of the part's shape into which the molten plastic is injected (for the purposes of this discussion, the difference between cavity and core is ignored). Each part cavity is milled out of two inserts that can be separated to remove the part. The mold can contain one or more cavities.

- The *mold base* holds the inserts in place and provides a *runner system* that brings the plasticated polymer to the inserts, a *cooling system* that cools the hot plastic so it can be ejected, and an *ejector system* that knocks out the finished plastic parts.

The ICAD application models the mold base (in particular, an A-series mold base, which is the simplest kind). The mold base consists of two main blocks:

- A stationary block into which the plastic is injected.

- A moving block that contains the ejector plate.

The stationary block is composed of two plates — the clamping plate and the A-plate — that are bolted together. The A-plate is also called the front cavity plate since the cavities for the runners and inserts are milled out of it.

The moving block is composed of the B-plate, a separate support plate for the B-plate, and the ejector mechanism. The B-plate is also called the rear cavity plate since the other half of the runners and inserts are milled out of it.

The ejector mechanism consists of a U-shaped ejector housing that provides the space for the moving ejector plate, and an ejector retainer plate. The B-plate is bolted to the support plate which is likewise bolted to the ejector mechanism.

About Mold-base Design

The following diagram shows a mold base in an open position.

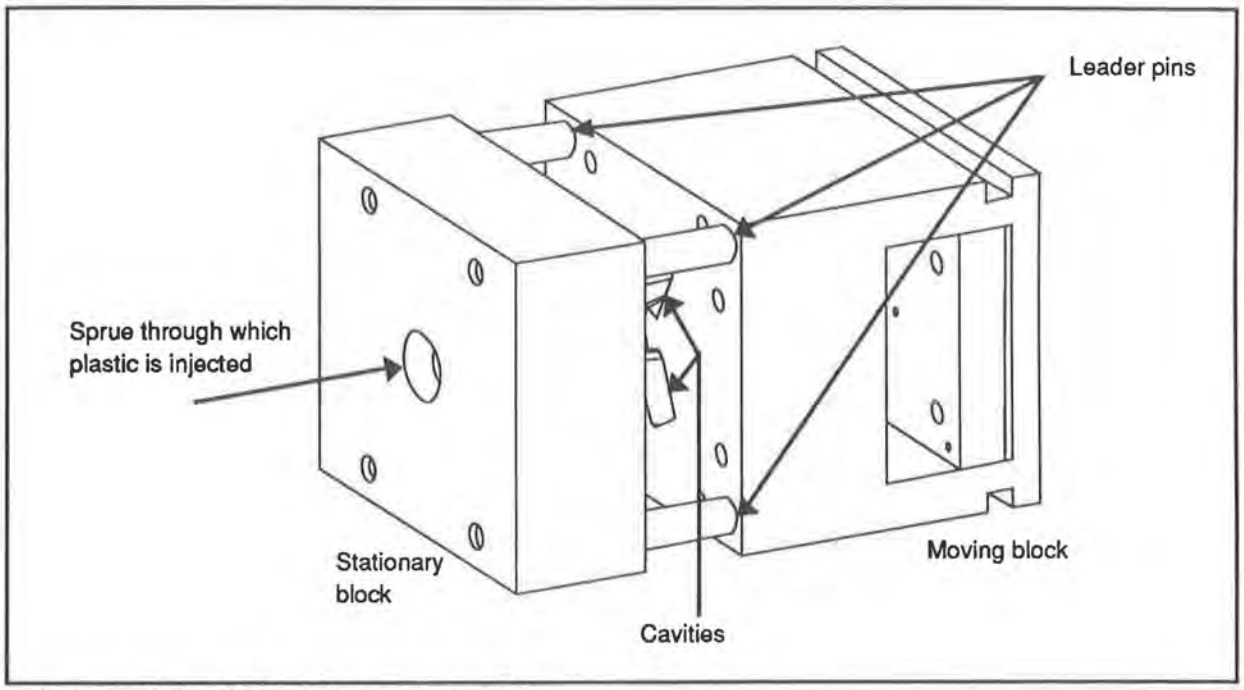
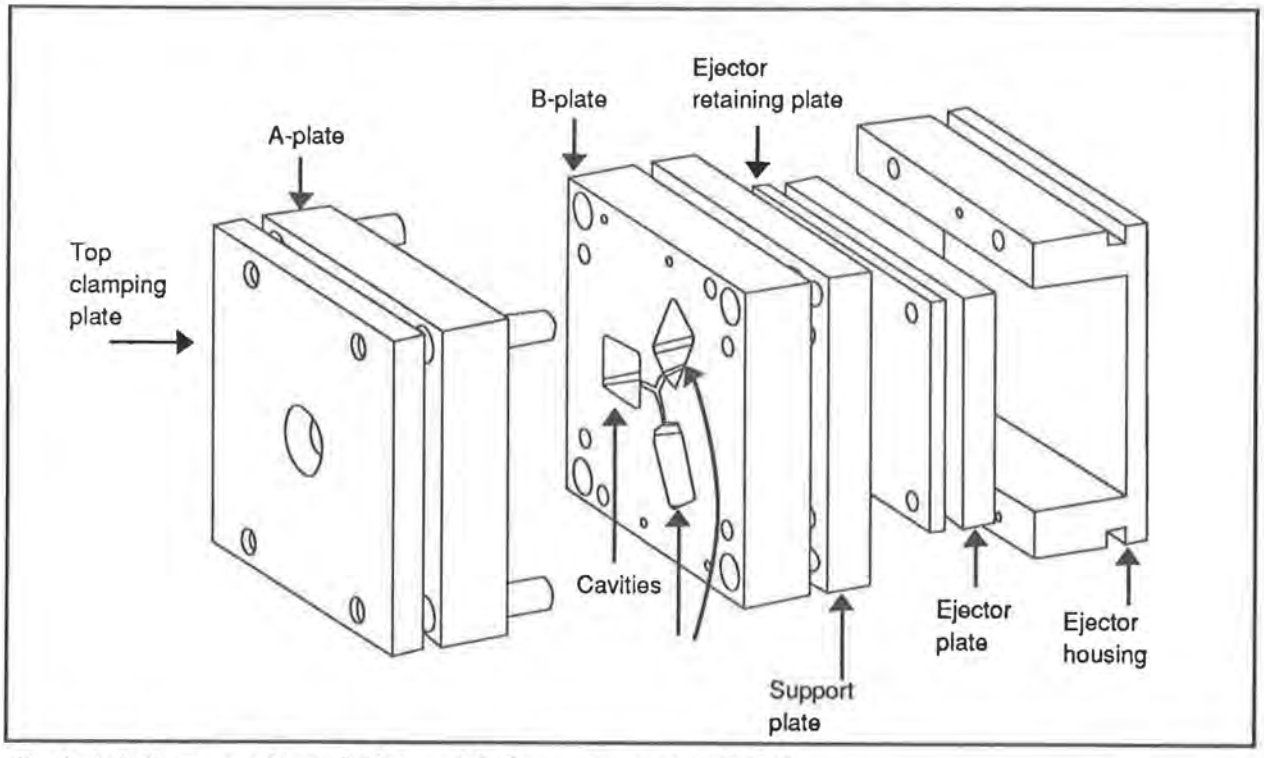


Diagram of a mold base in an open position.

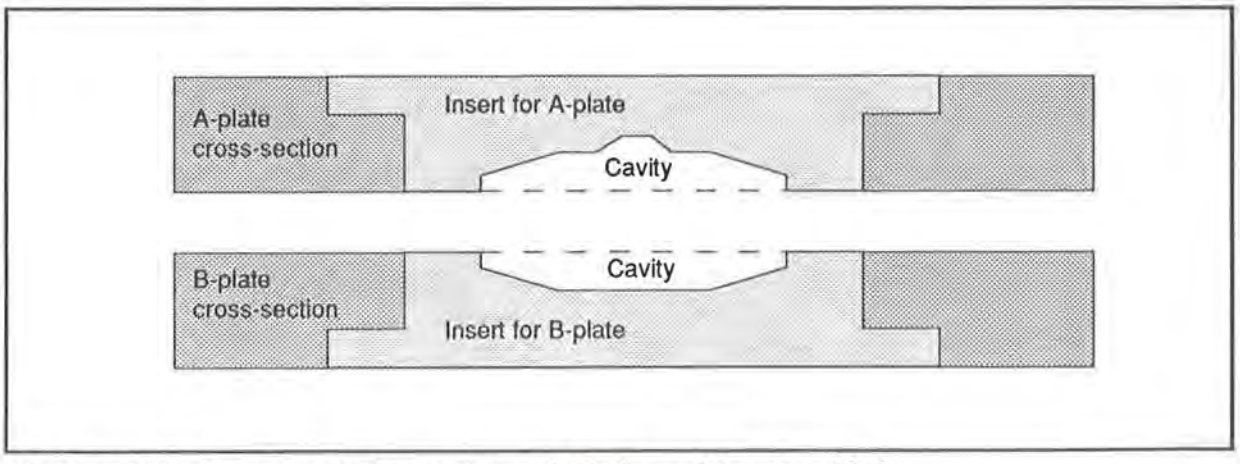
About Mold-base Design

The following diagram shows an exploded view of a mold base.



Exploded diagram of a mold base with the components labelled.

The inserts are separately milled and then inserted into the holes provided for them in the mold base. The inserts contain the cavities into which the plastic is injected, and the *gates* through which the plastic enters the cavities. A cross-section of the A- and B-plates with inserts and cavities is shown in the following diagram.



The A- and B-plates contain inserts from which the cavities are milled.

The mold base and inserts are then placed into an *injection molding machine*, which performs the injection molding. A standard production cycle consists of the following steps:

1. The mold base is closed. The two parts of the mold base are clamped together under high pressure (typically on the order of 75 to 1000 tons). The amount of pressure that can be brought to bear by an injection molding machine is called its *clamp tonnage*.

2. The temperature of the plastic is raised until the plastic can flow under pressure. This process is called *plasticizing the material*.

3. The plastic is injected into the closed mold through the *sprue*, which is a hole into the mold, from which it travels to the cavities via the runners. The amount of plastic that can be injected at one time by the injection molding machine is called its *shot capacity*.

4. The mold base is cooled, allowing the molten plastic in the cavities to solidify. In a *cold runner system*, the plastic in the runner system is also cooled. In a *hot runner system*, the plastic in the cavities is cooled but the plastic is kept plastized so it is not wasted.

5. The mold base is opened.

6. The ejector mechanism knocks out the solidified plastic.

Designing the Mold Base

Mold bases come in standard sizes and configurations that can either be purchased "off the shelf" from any number of different vendors or custom-manufactured. The holes for the inserts and the runner system must be custom-machined, and the ejector-pin placement must be custom-configured.

The mold-base designer must decide which injection-molding machine to use. This generally depends on which machine is available. Larger machines have a larger shot capacity (amount of plastic that can be injected), a faster plasticating rate (rate at which the plastic can be melted and transferred into the mold), and larger clamp tonnage (amount of pressure that can be exerted to hold the mold together while the plastic is injected).

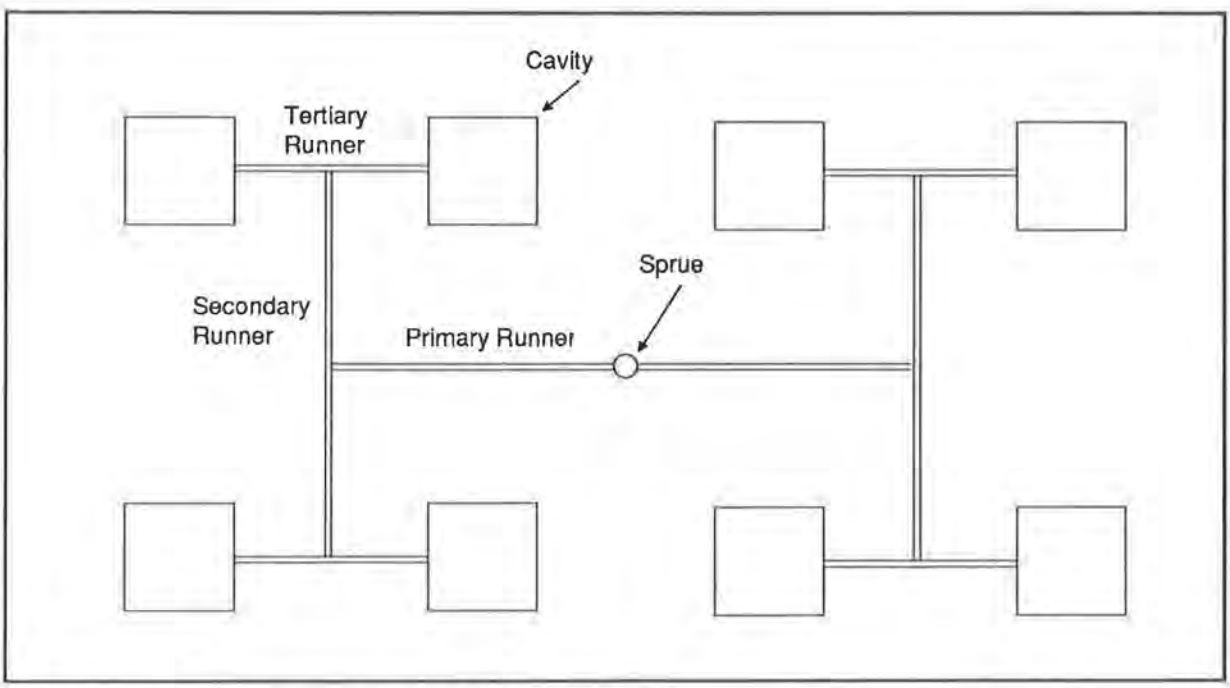
The size of the injection-molding machine determines the maximum size of the mold with which it can be used. In addition to molding larger parts, a larger mold can support multiple cavities, which allows the manufacture of multiple parts with each production cycle. It is always more expensive to prepare a mold base with multiple cavities since additional special-purpose milling is required. However, the production cost and time is lessened since more parts can be made

About Mold-base Design

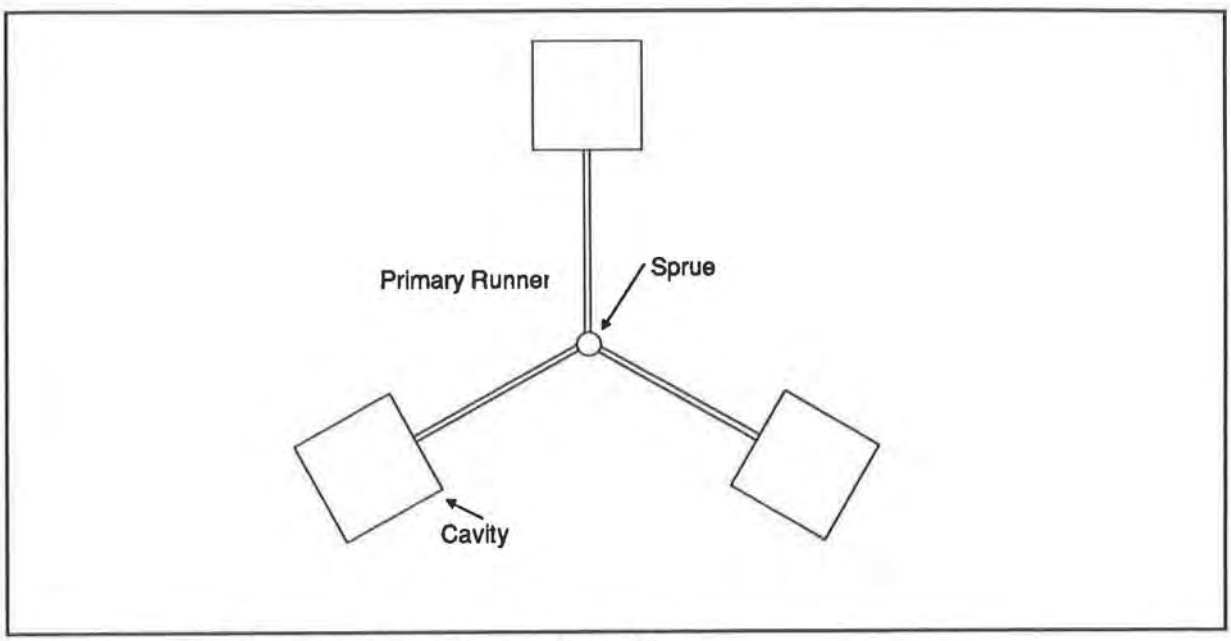
with each cycle. Multiple-cavity molds are generally more profitable for long production runs and are used in the majority of cases.

When multiple cavities are used, the mold designer must configure the layout of the cavities. The cavities should be arranged around the sprue so that each receives its full share of the total pressure through its own runner system. There should be the shortest possible distance between cavities and sprue, equal runner and gate dimensions, and uniform cooling. In a cold runner system, the runners should be as small as possible to minimize the amount of scrap plastic (although the plastic from the runners can often be reused, it is of lesser quality).

The following diagrams show the optimal cavity configurations for an eight- and three-cavity mold.



Optimal cavity configuration for an eight-cavity mold.



Optimal cavity configuration for a three-cavity mold.

About Mold-base Design

There are many other engineering attributes that must be taken into account to design the mold base. Different values for these attributes results in different design trade-offs. A sample of these attributes is listed here:

- *Viscosity,* i.e., resistance to flow. This is a property of the plastic material used whose value must be approximated. There are two techniques for approximating the value, one of which is faster but less accurate. In most cases, the less-accurate approximation is good enough; however, when the injection speed is fast, it is necessary to use the more-accurate, slower approximation.

- *Runner pressure drop.* The mold designer must specify how much pressure can be lost when the plastic goes through the runners before it enters the cavities. This value affects the amount of plastic waste in a cold-runner system.

- *Melt temperature.* This is partially the property of the plastic material, but it can be specified by the mold designer within a given range. The hotter the temperature, the better the quality of the molded part, but the longer the cycle time for producing the part.

- *Mold temperature,* i.e., the temperature to which the mold is cooled to harden the part properly. This is a property of the plastic material. Like the melt temperature, it affects the quality of the molded part.

Summary of Mold Base Specifications and Outputs

The goal of the mold-base designer is to design the most appropriate mold for a given part design. We assume that the mold insert has already been designed and the mold-base designer is given the following specifications:

- The volumetric properties of the part to be molded.

- The plastic material used to manufacture the part.

- The desired production quantity for the part.

- The available injection molding machines.

About Mold-base Design

The designer needs to produce the following:

- Annotated CAD drawings of the mold base, including the cavity layout, runner layout, and cooling layout.
- A bill of materials that lists the costs of each off-the-shelf component, including the standard mold base and the necessary pins, bushings, and screws.
- A summary of the necessary machining costs to machine the inserts and runners in the A- and B-plates.
- Total cost analysis for manufacturing the part, including the cost of preparing the mold base and the production costs.
- Routing sheet, pass details, tool details, machine details, and time standards for manufacturing the mold base.
- An NC path for milling the mold base.

The following chapters show how to develop an ICAD product model that takes these specifications and generates a design which includes these outputs.

Chapter 4

Modeling an Engineering Problem Using ICAD

Introduction

Overview

The chapter "Knowledge-based Engineering and The ICAD System" showed how an ICAD product model can accelerate the design process, save time and improve product quality. This chapter gives a technical overview of an ICAD product model and shows how you create and use one. Components of the mold-base design described in the previous chapter are used to illustrate various aspects of The ICAD System (other examples are used where the mold-base design does not illustrate a particular concept or capability).

It is important to realize that The ICAD System is not just another CAD-like design system except different and better. As shown in the previous sections, knowledge-based engineering is a completely new way of thinking about engineering and requires an understanding of new and possibly unfamiliar concepts, as well as new strategies and design techniques. It's worth learning the new concepts because the payoff is so large.

This section outlines some of the basic concepts upon which The ICAD System is built. The rest of the chapter fills in the details; it shows how the concepts map into the capabilities of The ICAD System, and how these capabilities help you create and use your ICAD product model.

ICAD Concepts

- *ICAD product model*

  An ICAD product model is an engineering description of a set of parts, products, or even a complete product line. The description is as generic as possible, that is, it can generate an appropriate product design for many different sets of input specifications.

- *ICAD Design Language*

  You create the ICAD product model using a specification language developed by ICAD called the ICAD Design Language, or IDL. The ICAD Design Language has an English-like syntax for describing the ICAD product model.

- *Product structure tree*

  An ICAD product model is composed of many different components. The different components represent the different assemblies, parts and features in the overall product. The components are organized into a tree structure called a product structure tree.

Introduction

- *Defpart*

  A *defpart* is a description in IDL of a single component of the product structure tree. A defpart includes engineering rules, or *attributes*, as well as a description of its subcomponents, or *subparts*. The product model itself is just all the defparts that describe the components of the product structure tree. One defpart in the ICAD product model is special since it describes the "root" of the product structure tree.

- *Generating a design instance*

  To generate a design, you supply some input specifications to the root defpart and ask The ICAD System to create a *design instance* of that defpart (that is, a specific version of the root defpart based on the input specifications).

- *Demand-driven evaluation*

  The design instance contains all the aspects of the design that you included in the ICAD product model, including the geometric model, analysis data, bill of materials, etc. However, none of this data is computed until you ask the system for it. This is known as demand-driven evaluation, since The ICAD System only *evaluates* the necessary part of the design instance when you *demand* some information.

- *Mixins*

  To avoid making complicated defparts, The ICAD System allows you to build complex defparts by "mixing together" simpler defparts; the new defpart has the same attributes (rules) and parts (components) as all of the "mixed-in" defparts. ICAD provides many basic defpart primitives that you can use as mixins in your own defparts.

Overview of This Chapter

This chapter includes the following sections:

"Planning an ICAD Product Model"
: Describes how you plan an ICAD product model, including dividing the problem into smaller components, organizing the product structure, and identifying inputs and outputs.

Introduction

"Writing Defparts and Generating Design Instances"
Describes the ICAD Design Language and shows how to create simple defparts, how to use a defpart to generate designs, and how to include engineering rules and inputs in a defpart.

"Writing Engineering Rules"
Shows how you write rules that are included in a defpart.

"Creating the Product Structure"
Shows how you create the product structure of the ICAD product model by including component parts in a defpart.

"Rule Dependencies in the ICAD Product Model"
Shows how to relate engineering attributes that belong to different parts of a product structure tree.

"Modularizing the ICAD Product Model"
Describes the capability of The ICAD System to modularize the design of the ICAD product model, including mixins and generic parts.

"Modeling with Simple Geometric Primitives"
Shows how you incorporate 2D and 3D geometry into an ICAD product model. It also introduces the large set of primitive parts available in The ICAD System.

"Additional ICAD-supplied Geometric Parts"
Introduces some of the other geometric parts that are made available to you as optional packages: curve and surface parts in the ICAD Surface Designer, which is a knowledge-based surface-modeling package, and solid parts in the ICAD Solids Designer, which is a knowledge-based solid modeling package.

"Developing Complex ICAD Product Models"
Describes some of the tools provided by ICAD for writing and testing an ICAD product model, including the ICAD Design Language compiler and the ICAD Browser.

"End-user Interfaces for an ICAD Product Model"
Describes how to use the finished ICAD product model in a production environment.

Planning an ICAD Product Model

Keywords: components, product structure, inputs, outputs

Overview

To get the most leverage out of The ICAD System, it is best to plan the application in advance. This section describes the planning stages of an ICAD product model and includes the following topics.

- Dividing the problem into smaller components.
- Deciding on the basic product structure.
- Choosing appropriate input specifications.
- Determining the necessary outputs.

Dividing the Problem into Smaller Components

A successful ICAD product model is developed by dividing the larger problem into smaller component problems, each of which is more manageable than the problem taken as a whole. While solving the entire problem is the eventual goal, the smaller components provide effective milestones in the development of the final application and usually automate some portion of the entire design. This allows the application to show some quick returns, with incremental returns following over the course of the development.[4] Also, this allows different members of the design team to work concurrently on different components.

For all complex systems, there is a size beyond which solving a large problem is disproportionally more complicated than dividing the large problem into smaller pieces, solving each of the pieces, and integrating the separate solutions. It is best to break down all problems so that they are no larger than this size.[5]

For example, the mold-base product model is divided into several manageable tasks:

- Feature extraction from the IGES input data.

---

[4]The ability to subdivide an ICAD product model into smaller pieces easily is one of the capabilities of The ICAD System that is very hard to do using traditional programming languages.

[5]This is not an inherent limitation of The ICAD System; rather, subdividing complex problems is a generally-accepted problem-solving strategy.

Planning an ICAD Product Model

- Selection of appropriate mold base and configuration of cavities and standard parts.
- Creation of process plan for the mold base.
- Creation of annotated CAD drawings.

The mold base design, itself a component problem, is broken down into sub-problems:

- Determination of the maximum number of cavities, based on the volumetric properties of the part design, the material properties of the plastic, and the type of injection molding machine.
- Configuration of the most efficient cavity layout and runner system.
- Selection of a standard mold-base part from a catalog of mold bases.
- Selection of standard pins, screws, bushings, etc., from catalogs.
- Configuration of the most efficient cooling system.
- Collection and reporting of costs for component parts and machining.

The process plan can also be further divided:

- Determination of form features that must be custom-machined in the mold base.
- Generation of a process plan to manufacture each part of the mold base.

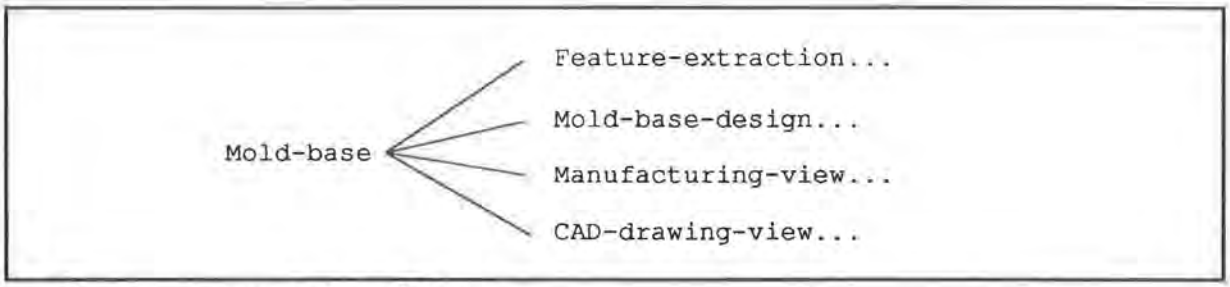
Deciding on the Basic Product Structure

The *product structure* of the ICAD product model represents the way the product model is divided into components, each of which is further divided into subcomponents, etc. A component might represent a physical part or a process.

The first step in developing an ICAD product model is deciding on the organization of the top levels of the product structure, that is, the major assemblies and processes that you are representing in the ICAD product model. Usually this reflects the way that you have broken the problem into components, that is, the first level of the product structure represents the different components of the problem. For example, the product structure of the mold base

Planning an ICAD Product Model

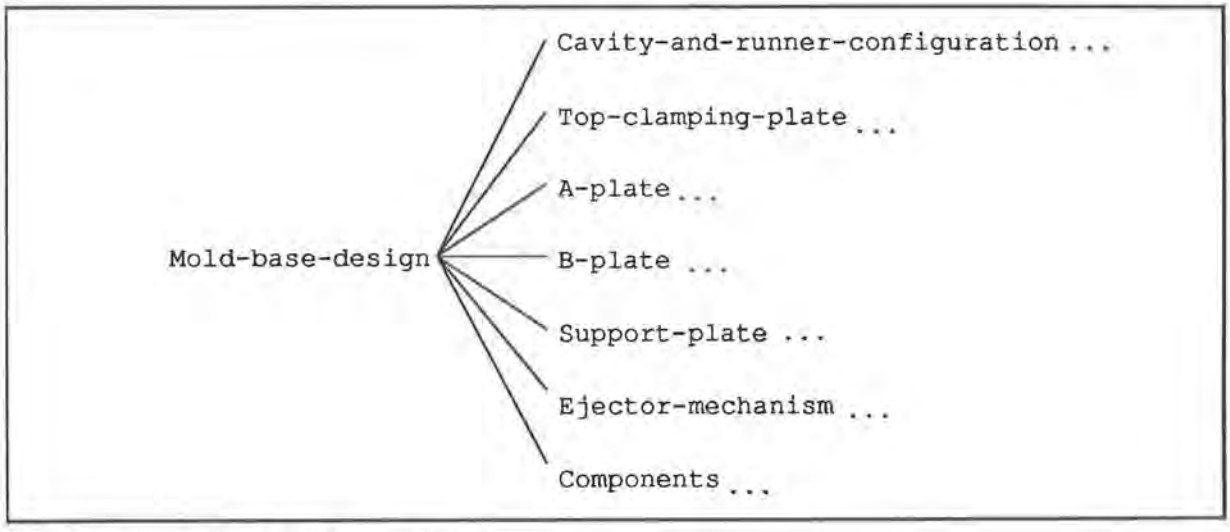
reflects the four main problem components:[6]



Problem components of the mold base.

The most basic component of the mold-base application is the design of the mold base. Its product structure reflects the basic assemblies of the mold base, including the different plates, the cavity and runner configuration, etc.

The following diagram shows the major components of the mold-base design:



Major components of Mold-base-design.

This structure closely matches the design of complex mechanical assemblies in which the overall assembly is divided into subassemblies, which are further divided into subassemblies as needed. Lower levels include standard parts (which can be purchased from a vendor or custom-manufactured), features, and geometric-construction parts (such as a curve, surface, or solid). For a more

[6]Ellipses in the diagram indicate that the terminal components are divided into further components.

Planning an ICAD Product Model

detailed discussion of the product structure, see the section "Creating the Product Structure", page 4-29.

Engineers are concerned with the relationships between the components of an assembly. The ICAD System lets you describe these relationships by allowing a component description to refer directly to another component in a different assembly. For example, even though the plates and the screws that hold the plates together are in different assemblies, you can relate the thickness of the plates to the type of screw needed to hold the plates together.

If different individuals are working on different components, you should decide upon the interface between the different components. For example, the interface between the feature-extraction component and the mold-base design components of the mold-base application is accessed by rules in the feature-extraction component that completely specify the part to be molded. The mold-base-design component need not know anything about the feature-extraction component except how to reference these rules.

Identifying Inputs

By identifying inputs to the ICAD product model, you define the requirements of the design. This means you distinguish between elements of the design that can be accomplished by the ICAD product model using rules, and elements of the design that must be supplied by the design engineer.

Usually the basic problem is represented by an input specification, that is, the design engineer is asked to design a product given some specifications. Clearly, it is logical to include these specifications as inputs to the ICAD product model. Within the scope of the specifications, the design engineer usually has some latitude to decide on other design issues. When developing an ICAD product model, you need to decide how much of that latitude should be left to the design engineer using the ICAD product model, and how much should be built into the rules.

Taking the mold-base design for example, the input specifications include the properties of the part to be molded, the plastic material, the production quantity, and the available injection molding machines. These are all inputs to the ICAD product model. In addition, the ICAD product model prompts the mold-base designer for additional design decisions. For example, when determining the optimal cavity configuration, the ICAD product model lets the design engineer choose whether the cavity layout is optimized for cost or time efficiency.

Identifying Outputs

The purpose of an ICAD product model is to generate outputs. Initially it is more important to be sure that the ICAD product model has the correct content, regardless of the format the outputs will take. The components of an ICAD product model that generate outputs are usually developed towards the end of

Planning an ICAD Product Model

the development process. However, the eventual outputs should still be identified in the planning stages, since they can affect the content of the rest of the ICAD product model. For example, in order to implement a bill-of-materials component, the mold-base design includes cost data in the description of each part. While the bill of materials is developed later, it uses the cost data that already exists in the mold-base-design component.

Writing the Rules

Once an appropriate problem has been selected and divided into smaller components, you need to consider how to best use the capabilities of the ICAD Design Language, described later in this chapter, to model the problem. Most rules are written on a case-by-case basis.

Summary

- There are many strategies that can be used to design an effective product model. Good planning is essential. ICAD consultants will help you plan your initial project.
- Divide the problem into smaller components: each component is easier to solve than the entire problem, and solving a small component gives you immediate payback. The product structure of the ICAD product model is composed of many of these components.
- Identify the inputs to and outputs from the ICAD product model in the planning stages.
- Most components in the ICAD product model can be developed on a case-by-case basis; the important issues to resolve early on are how the different components interact together, and how the components are organized into the product structure.

Writing Defparts and Generating Design Instances

Keywords: *product structure, ICAD Design Language, defpart, attributes, compilation, design instance, instantiation, demanding a value, input specifications*

Overview

This section shows how to describe each component of a product model and includes the following topics:

- How to write *defparts* — descriptions of product components — using the ICAD Design Language.
- How to compile a defpart and generate a design instance.
- How to generate different design instances from the same defpart by specifying inputs to the defpart.

What is a Defpart?

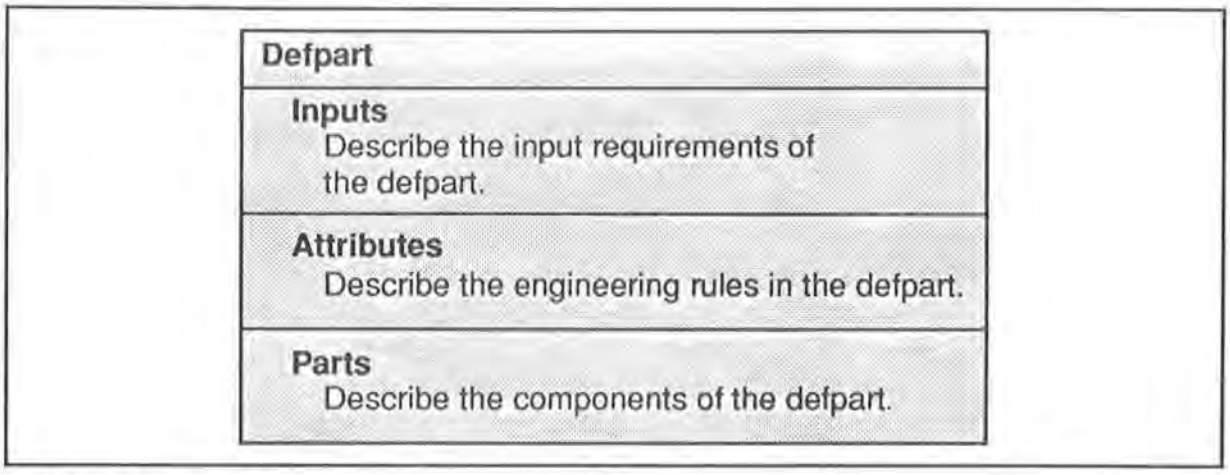
A defpart ("*def*inition of a *part*") is a description of component of an ICAD product model. The ICAD product model is just a collection of defparts. A design instance of the ICAD product model is a collection of defpart instances organized in a product structure tree. Just as you supply inputs to the ICAD product model to generate a design instance, you supply inputs to a defpart to generate an instance of the defpart.

A defpart consists of the following:

- *Input-attributes* are the inputs that must be specified in order to create an instance of the defpart.
- *Attributes* are the engineering rules.
- *Parts* are the components of the defpart; they describe the product structure tree that is created when a design instance of the defpart is generated.

Writing Defparts and Generating Design Instances

The following diagram shows the different sections of a defpart:



Different parts of a defpart.

This section discusses simple defparts which are not related to other defparts in a product structure tree. It shows how to write a defpart using the ICAD Design Language, and how to generate a design instance from the defpart. It also shows how attributes and input-attributes are incorporated into a defpart.

The section "Writing Engineering Rules" describes different kinds of attributes and input-attributes in detail.

The section "Creating the Product Structure" shows how defparts describe the product structure of a design instance by defining parts.

Introduction to the ICAD Design Language

Defparts are written using an engineering specification language provided by ICAD called the ICAD Design Language, or IDL. The ICAD Design Language has many features that make it particularly expressive for describing engineering parts and processes:

- It is an *object-oriented* language. This means that different components of the product are described as separate pieces or objects, that is, a defpart. Each defpart contains engineering rules or *attributes* that describe it.

- It provides facilities for describing complex relationships between defparts. Often the description of one defpart depends on aspects of other defparts.

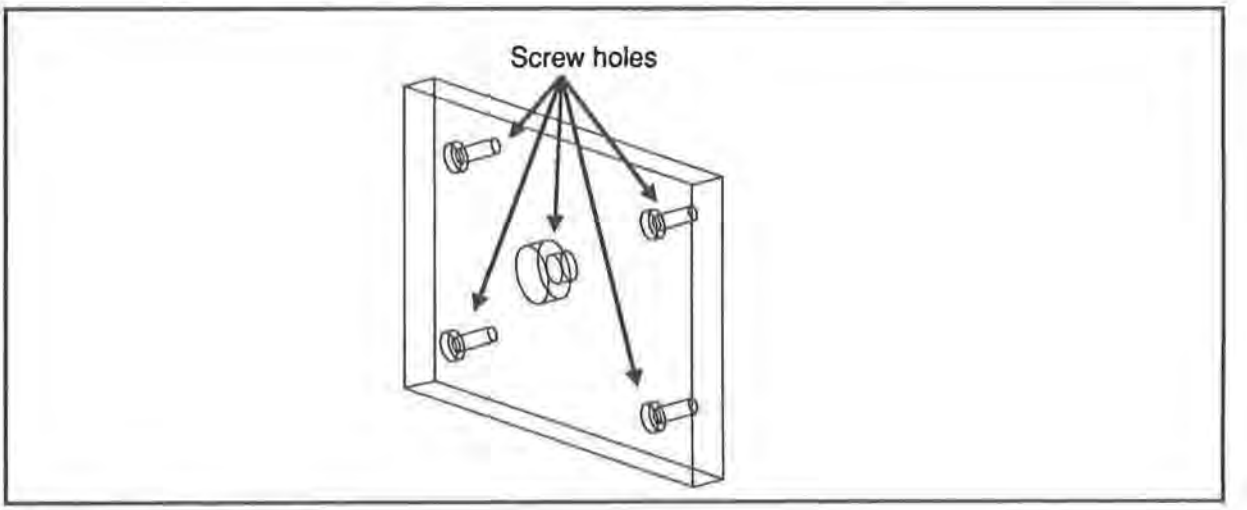
Writing Defparts and Generating Design Instances

- Engineering attributes are described in engineering terms using names that you define.
- The order in which you describe the different defparts is irrelevant since The ICAD System determines the dependencies between defparts automatically. In this way, you can describe the higher-level components first, and then describe the lower-level components on which the higher-level components depend.

A Simple Defpart

This section shows how you can use the ICAD Design Language to describe one component of a mold base, which is a plate. (The mold base includes four plates; see the section "About Mold-base Design", page 3-2 for details.)

A Mold-base-plate defpart describes the geometry of a plate by its width and thickness, as well as the placement of holes for the inserts and screws.[7] The description also incorporates various engineering attributes, such as the plate's maximum expected load and the plate's maximum expected stress (which is a function of the the maximum expected load, the thickness, and the width). The following diagram shows a trimetric view of a plate:



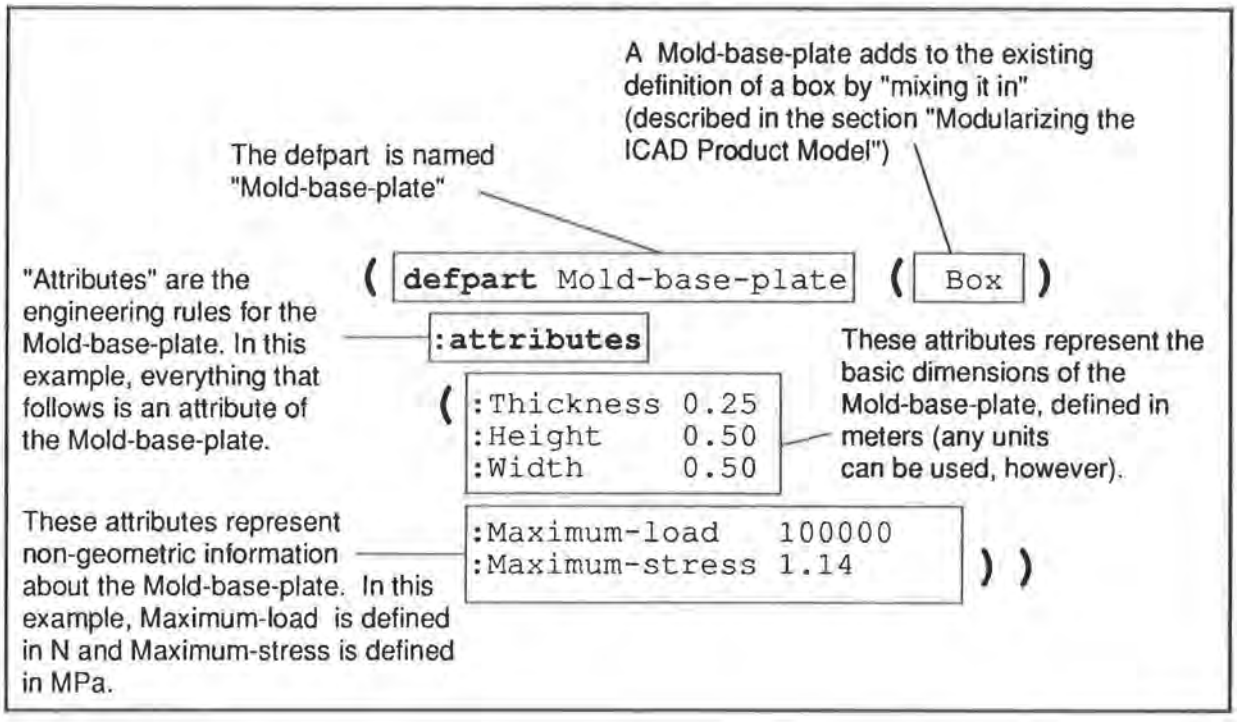
Trimetric view of a plate.

[7]For the purposes of this introductory example, a square Mold-base-plate is assumed.

Writing Defparts and Generating Design Instances

A simplified Mold-base-plate defpart is shown in the following diagram. Throughout the next few sections, this defpart will be rewritten to show more and more capabilities of The ICAD System.

In all ICAD Design Language examples shown in this chapter, names and syntax that are part of the ICAD Design Language are shown in boldface type, while user-defined names and values are shown in standard (roman) type.



The example starts with a parenthesis which indicates that a new defpart is being defined. The word **defpart** indicates that the name and definition of the defpart follow.

After giving the name of the defpart, the example supplies a mixin for the defpart, that is, the name of another defpart whose definition is added to (or "mixed into") the Mold-base-plate defpart. The Mold-base-plate is a kind of a Box, since it mixes in the definition of a box. See the section "Modularizing the ICAD Product Model", page 4-49 for a more detailed discussion of mixins.

Following the ICAD Design Language keyword **:attributes** is a list, in parentheses, of engineering attribute names and values. Engineering rules are usually defined as defpart attributes. In this defpart there are five different attributes. The first three, Thickness, Height, and Width, represent basic

Writing Defparts and Generating Design Instances

geometric information about the part.[8] Maximum-load and Maximum-stress represent engineering information.

Finally, a closing parenthesis matches the opening parenthesis, indicating the end of the defpart.

In general, The ICAD System does not assume any units; you are responsible for maintaining unit relationships. The Mold-base-plate is defined using international units, that is, the lengths are defined in meters (m), the load is defined in Newtons (N), and the stress is defined in Pascals (P).

In this first example, all of the attribute values are constants. Later examples show that the value of some of the attributes are directly related to other attributes. Since we are restricting ourselves to a square Mold-base-plate, the value of the Height must be the same as the value of the Length. Additionally, the Maximum-stress can be computed from the values of the other attributes.

[8]The syntax of the ICAD Design Language requires that attribute names start with colons. The colons are dropped in the text of this document. Names that are part of the product definition are capitalized.

Writing Defparts and Generating Design Instances

This next example of the Mold-base-plate defpart includes rules for calculating the Height and the Maximum-stress.

```
( defpart Mold-base-plate (Box)
  :attributes
   ( :Thickness 0.25
     :Height (the :Width)
     :Width 0.50
     :Maximum-load 100000
     :Maximum-stress
      (div (* 0.2874
              (the :Maximum-load)
              (^2 (the :Width))
              1.0E7)
           (^2 (the :Thickness)))))
```

The rule for computing the Height says that its value is the same as the Width.

"**The**" means get the value of the attribute. This is called *referencing* the attribute.

Notice that order of the attributes in the defpart is irrelevant.

The rule for computing the Maximum-stress uses a simple formula for stress:

$$\frac{0.2874 * \text{Maximum-Load}^2 * \text{Width} * 10^6}{\text{Thickness}^2}$$

Since the values of Width, Thickness, and Maximum-load are used in the rule for calculating the Maximum-stress, we say that Maximum-stress *depends on* Width, Thickness, and Maximum-load.

At first glance, this rule looks somewhat strange because the ICAD Design Language uses a *prefix notation* to represent expressions. That is, the operator comes first, followed by each operand. Each expression is written inside a set of parentheses. For example, the expression `5*6` is written `(* 5 6)` in prefix notation. **div** means divide and **^2** means raise to the power of 2.

Generating a Design Instance from a Defpart

Once you have written a defpart, The ICAD System provides a way to generate design instances from the defpart. The first step is to *compile* the defpart, that is, The ICAD System translates the defpart into a form the computer can understand. Once the defpart is compiled, you can ask The ICAD System to produce a design instance of the defpart. This is called *instantiating* the defpart.

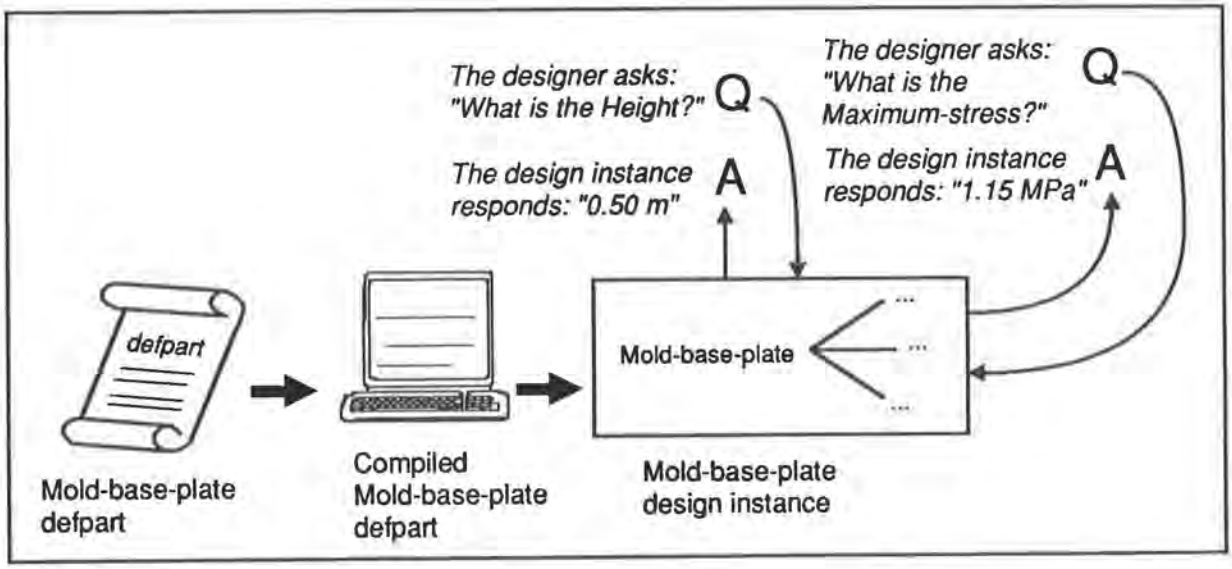
The design instance knows how to evaluate the rules in the defpart to produce results. For instance, the previous defpart contains a rule that says the value of the Height is the same as the value of the Width. The design instance knows how to calculate the value of the Height based on that rule. When you ask the

Writing Defparts and Generating Design Instances

design instance for its Height, it evaluates the rule and responds with 0.20.[9]

Note that the design instance does not evaluate any rules until you ask for their values. This capability is called *demand-driven evaluation*. Demand-driven evaluation minimizes the number of calculations that The ICAD System performs, that is, only those calculations necessary to compute the results that you request are performed. When you ask for the Height, for example, the design instance evaluates the rule for the Height and Width (since Height depends on Width); it does not evaluate the more complex rule for Maximum-stress.

The following diagram shows how you produce a design instance from a defpart. First you compile the defpart, then you ask The ICAD System to create a design instance from the compiled defpart. Finally, you can ask the design instance to evaluate rules built into the defpart.

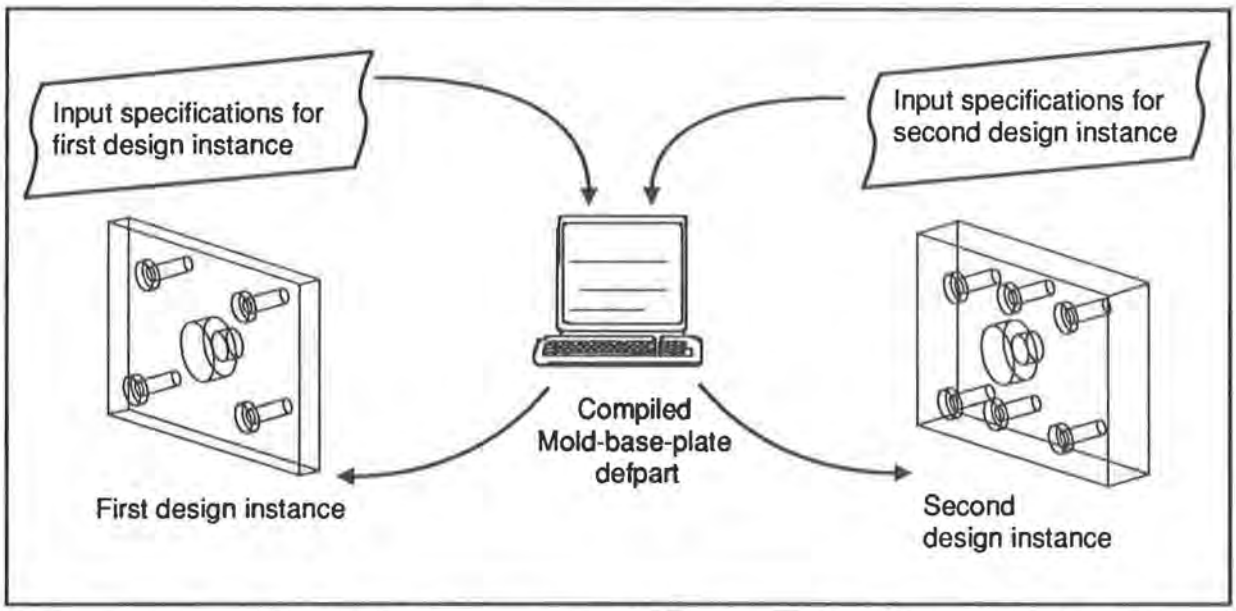


[9]Geometric attributes of a design instance are commonly demanded by asking The ICAD System to draw a picture of the design instance.

Writing Defparts and Generating Design Instances

Including Input Specifications in a Defpart

Defparts can contain *input requirements* as well as engineering rules. Input requirements are like engineering rules except that their values are not part of the defpart; The ICAD System expects you to supply values when the compiled defpart is used to generate a new design instance. The values of the inputs can substantially change the characteristics of the product design since many engineering rules depend on other values. If some values are inputs, then values of the dependent engineering rules change as the inputs change.



Different input specifications to the same compiled defpart generate different design instances.

For example, the previous Mold-base-plate defpart has a fixed Width, Thickness, and Maximum-load. A more interesting defpart, shown below, describes a plate of any size with a varying maximum load. Width, Thickness, and Maximum-load are now inputs, that is, you supply them when you create a new design instance.

Writing Defparts and Generating Design Instances

The following example shows this new Mold-base-plate defpart. You must supply the values of the Width, Thickness, and Maximum-load when you create a new Mold-base-plate part design. Notice that the definition of Maximum-stress is the same as before. However, since the value of Maximum-stress depends on the values of the inputs Maximum-load, Width, and Thickness, its value changes as these inputs change.

```
(defpart Mold-base-plate  (  Box  )
:inputs
  (:Width :Thickness :Maximum-load )
:attributes
  (:Height (the :Width)
   :Maximum-stress
   (div (* 0.2874
           (the :Maximum-load)
           (^2 (the :Width))
           1.0E7)
        (^2 (the :Thickness))) ))
```

"Inputs" are attributes whose values are supplied by the end user when a new design instance is generated.

The definition of Height and Maximum-stress are the same as before.

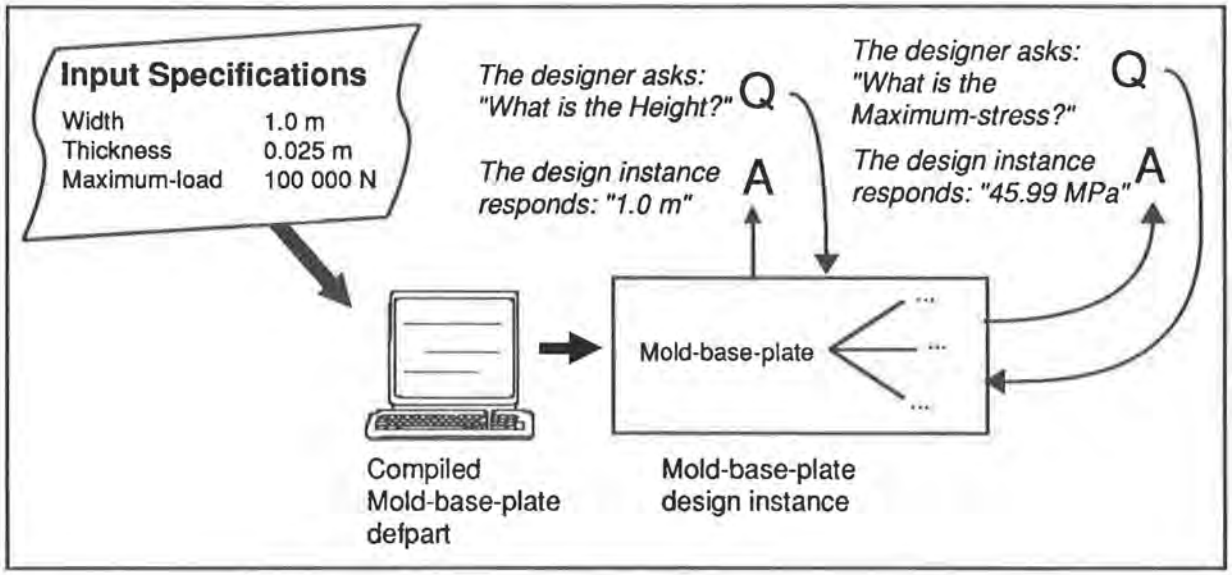
Creating Different Design Instances from the Same Defpart

With this new defpart, you must supply values for Width, Thickness, and Maximum-load when you generate a new design instance. For instance, you could supply the following set of inputs to the defpart:

Width	1.0 m
Thickness	0.025 m
Maximum-load	100 000 N

Writing Defparts and Generating Design Instances

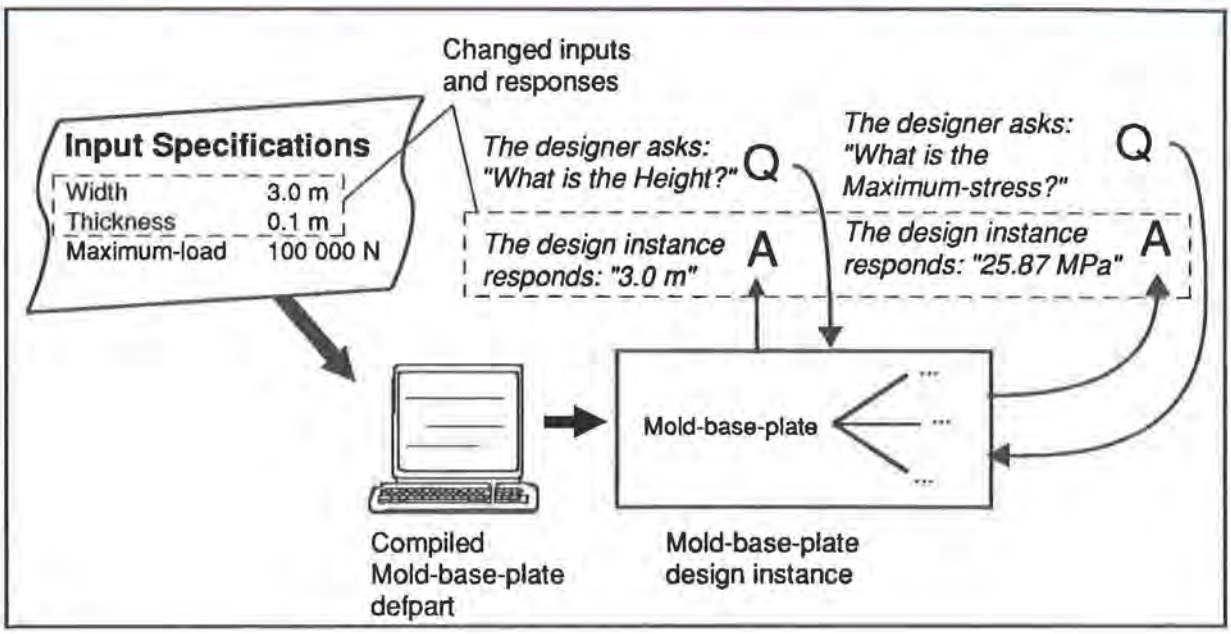
If you ask the design instance for its Maximum-stress, it evaluates the rule for Maximum-stress using the input values.



Another set of inputs creates a new design instance which has a different value for Maximum-stress. The following inputs to the Mold-base-plate defpart, for instance, produce a new value for Maximum-stress:

Width	3.0 m
Thickness	0.10 m
Maximum-load	100 000 N

Writing Defparts and Generating Design Instances



This shows how an ICAD product model can be used to do "what-if" analysis. In this simple Mold-base-plate defpart, you can investigate how the Maximum-stress changes as the input specifications, e.g., the Width, Thickness, and Maximum-load, change.

Summary

- ICAD product models are divided into components. Defparts describe each component.
- Defparts contain both engineering rules and input requirements for each component. These rule encapsulate the engineering knowledge about the component. Defparts are written using the ICAD Design Language.
- To use a defpart, you compile it, and then generate a design instance from the compiled defpart. You can ask the design instance to evaluate any of the rules in the defpart. The results depend on both the input specifications and the rules.

Writing Engineering Rules

Keywords: attributes, rules, conditional attributes, catalogs, trials, remote evaluation, demand-driven evaluation, unbound values, inspectors, cached values, input-attributes, choice-attributes

Overview

This section shows how you include engineering rules in a defpart and includes the following topics:

- Attributes as engineering rules.
- Descriptions of different kinds of attributes.
- Evaluation of the value of attributes in a design instance.

Attributes

The rules in a defpart are called *engineering attributes*, or just *attributes*. Attributes define a part's physical dimensions, its location in space, its economic traits, its material properties, and any other kind of information that is required. In fact, you can model any engineering property that describes the product using attributes.

The previous section showed how some simple engineering rules are represented as attributes. It also showed how some attributes are represented by inputs, that is, their values are supplied when a new part design is generated.

This section shows some of the more complex ways to define engineering rules as attributes.

Attributes as Constant Values

For simple attributes, you supply the value when you write the defpart. For instance, one attribute of the mold base is the Polymer-melt-temperature. The following defpart fragment defines this attribute and specifies its value as 300 Celsius.[10]

```
:Polymer-melt-temperature 300
```

[10]A defpart fragment is a piece of a defpart. We sometimes show defpart fragments instead of the entire defpart to simplify this document.

Writing Engineering Rules

The following fragment defines an attribute called Material. Its value says that the Mold-base is made out of AISI 4130 steel. In this case, the value of the attribute is a name, not a number.[11]

```
:Material 'AISI4130-steel
```

Attributes Whose Value Depends on Other Attributes

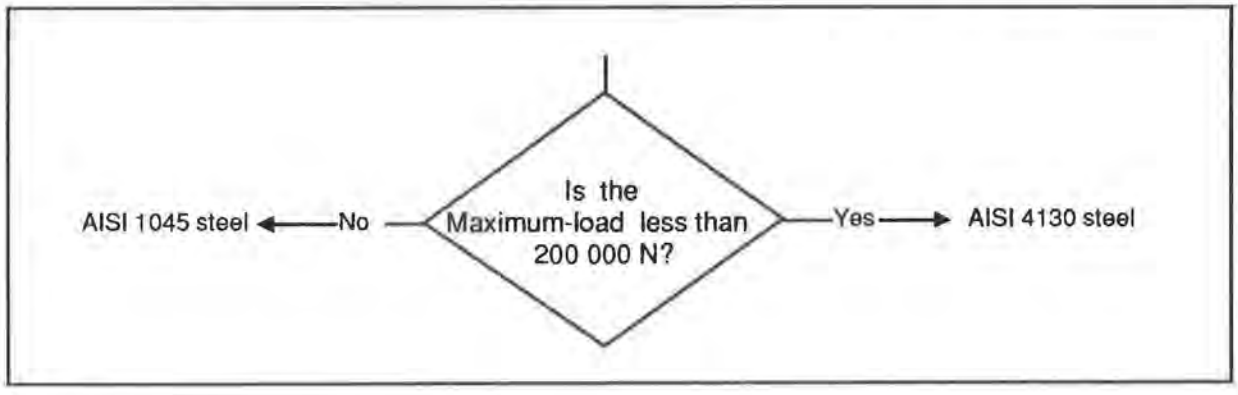
More interestingly, you can define the value of attributes relative to other attributes. In the Mold-base-plate defpart, for instance, the value of Maximum-stress is based on the Thickness, Width, and Maximum-load. As another example, the rule that calculates the Required-melting-temperature in the mold base is defined as "multiply the Polymer-melt-temperature by 1.05". This is written as follows in the ICAD Design Language:

```
:Required-melting-temperature (* 1.05 (the :Polymer-melt-temperature))
```

Conditional Attributes

The value of an attribute can be defined *conditionally*, that is, its value is one of several different choices depending on a test condition.

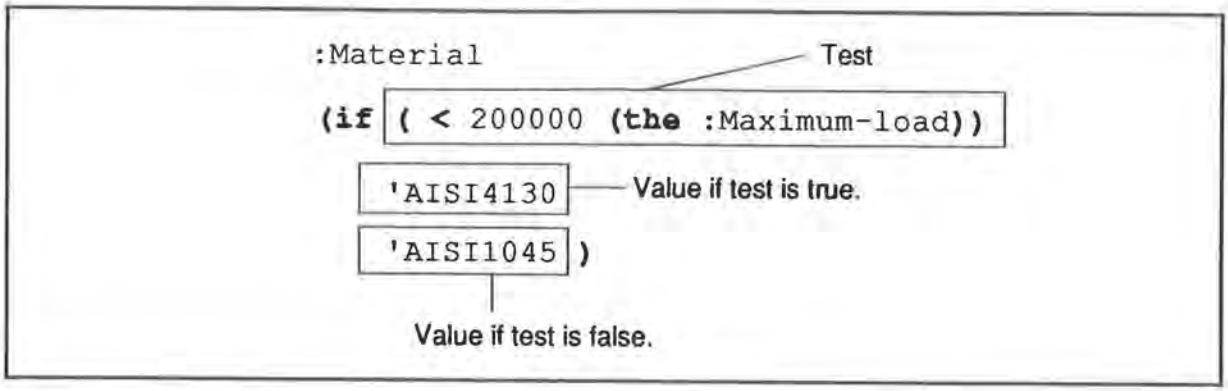
The mold-base plates, for example, can be constructed from two different materials, AISI 1045 steel or AISI 4130 steel. The AISI 1045 steel has better stress characteristics than the AISI 4130 steel, but it is also more expensive and should only be used when the maximum load requires it. The value of the Material attribute is calculated using the following rule: If the Maximum-load is less than 200 000 N use AISI 4130 steel, otherwise use AISI 1045 steel.



[11]The quote character indicates that AISI4130-steel should be interpreted literally, that is, the computer should not try to translate it into a number.

Writing Engineering Rules

You express this in the ICAD Design Language as follows:



Input-attributes

Input-attributes represent the input requirements of the defpart. You supply a value for each input-attribute when you generate a new design instance. In the previous section we saw how Width, Thickness, and Maximum-load can be supplied as inputs to the Mold-base-plate. You must give a value for Width, Thickness, and Maximum-load whenever you make a new Mold-base-plate design.

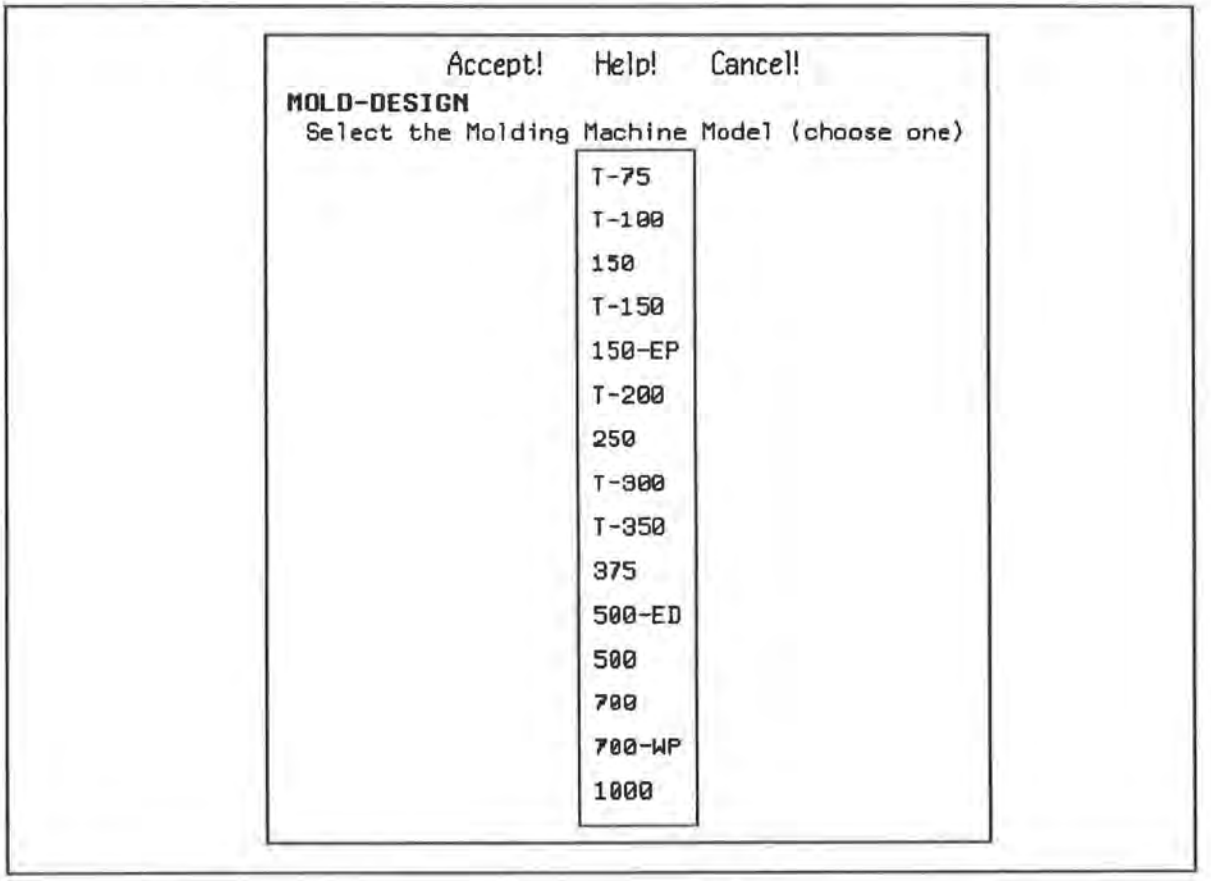
Choice-attributes

Defparts can also contain *choice-attributes*. The values of choice-attributes are not part of the engineering rules. Instead, you supply the value of a choice-attribute when prompted by The ICAD System.

Choice-attributes are like inputs in that you supply their values when the design instance is generated. However, unlike inputs, you do not supply their value when the design instance is first created. Instead, the design instance prompts you for the value of a choice-attribute while it is computing an engineering rule that depends on the value of the choice-attribute. Thus, you do not have to supply values for all choice-attributes, just the ones that are needed for the design instance.

Writing Engineering Rules

The mold base, for instance, requires that you choose an injection molding machine from a list of possible injection molding machines. This list is calculated from a catalog, so only certain machines are valid choices. The rule that describes the choice-attribute insures that only valid choices are presented. The following diagram shows the pop-up that appears to the end user when a value for the choice-attribute is required.



Writing Engineering Rules

Other Kinds of Attributes

There are many other ways to write engineering rules in The ICAD System:

- *Catalog lookup*

  The value of the attribute can be computed by looking it up in an on-line catalog. For example, the cost of each screw used to hold the plates together can be determined by querying a catalog or database.

- *Engineering trials*

  Each value in a series of values can be "tried" as the value of the attribute. The first value that successfully passes a test is used as the value of the attribute. For example, there is a set of possible kinds of steel that can be used for the body of the plates. The least-expensive steel is tested first as the value for the plate material. If it is not strong enough, the next-expensive steel is tested, etc., until one is found that is strong enough for the job.

- *Remote evaluation*

  The value of an attribute in a defpart can actually be computed by another engineering package, and then used in the ICAD product model. For example, the value of the Mold-flow-result attribute in the Mold-base is calculated by a remote flow-analysis package.

Writing Engineering Rules

Evaluating the Value of Attributes

Rules that are used to calculate the value of attributes are defined in the defpart. The actual values of the attributes are associated with design instances of the defpart, that is, after you generate a new design instance from the defpart you can evaluate the value of some of its attributes.

Demanding the value of an attribute causes the rule that defines the attribute to be evaluated. In addition, all attributes on which the rule depends are also demanded and evaluated; however, no other attributes in the ICAD product model are evaluated. This insures that only those calculations that are required to compute the desired result are executed. This feature of The ICAD System is called *demand-driven evaluation* and is a main reason that The ICAD System can manage large designs with thousands of parts effectively.

Consider the Mold-base-plate defpart (this is the same Mold-base-plate defpart discussed in the previous section):

```
( defpart Mold-base-plate    ( Box  )
  :inputs
  (:Width :Thickness :Maximum-load )
 :attributes
  ( :Height (the :Width)
    :Maximum-stress
    (div (* 0.2874
            (the :Maximum-load)
            (^2 (the :Width))
            1.0E7)
         (^2 (the :Thickness))) ))
```

Notice that the Height depends on the Width, that is, in order to evaluate the Height The ICAD System must first evaluate the Width. Likewise, the Maximum-stress depends on the Maximum-load, the Width, and the Thickness.

Writing Engineering Rules

Consider a design instance of the Mold-base-plate with the following input specifications:

Width	1.0 m
Thickness	0.025 m
Maximum-load	100 000 N

The ICAD System provides a tool called an *inspector* in which the value of all the rules in a design instance are shown. When the design instance is first generated, the inspector contains the following information:

Thickness	*Unbound*
Width	*Unbound*
Height	*Unbound*
Maximum-load	*Unbound*
Maximum-stress	*Unbound*

All of the attributes are *unbound*, that is, none of their values have been evaluated by the system yet.

When you ask the design instance for the value of an attribute, that attribute is *demanded* by The ICAD System, and any other attribute upon which that value depends is also demanded. For instance, since Height depends on Width, when you ask for the value of Height, the value of Width is also calculated.

After Height is requested, the inspector looks like this:

Thickness	*Unbound*
Width	1.0
Height	1.0
Maximum-load	*Unbound*
Maximum-stress	*Unbound*

Both Width and Height are calculated.

Writing Engineering Rules

After an attribute is calculated, its value is automatically *cached*, that is, it is stored in the memory of the computer. This means that the first time an attribute is demanded, The ICAD System computes its value from the appropriate rules and then remembers the value.[12] The ICAD System never has to recompute that attribute; subsequent times the attribute is required, The ICAD System just returns the cached value. This is much more efficient than recalculating the value each time it is required.

For example, when you ask for the value of Maximum-stress, The ICAD System automatically demands the values of Maximum-load, Thickness, and Width, since Maximum-stress depends on these other attributes. The ICAD System finds that Maximum-load and Width have not yet been calculated, so it calculates those values. The value of Thickness has already been calculated, so it returns the cached value without recalculating it.

After Maximum-stress is requested, the inspector looks like this:

Thickness	0.025
Width	1.0
Height	1.0
Maximum-load	100 000.0
Maximum-stress	45.99

[12]Unlike most programming languages, The ICAD System takes care of the order of evaluation and data dependencies for you. That is, The ICAD System always knows whether an attribute depends on the value of another attribute.

Writing Engineering Rules

Summary

- Attributes represent the rules in the product model.
- Attributes can be simple values, arithmetic expressions defined relative to other attributes, conditional values, catalog entries, trial values, or values returned after executing another computer package.
- You supply values for input-attributes when you create a new design instance.
- You supply values for choice-attributes when prompted by the design instance.
- The values of attributes are computed on demand. Only those attributes that you request are computed. Once computed, the value is cached.

Creating the Product Structure

Keywords: product structure tree, parts, children, root, leaves

Overview

This section shows how to create the product structure of an ICAD product model and includes the following topics:

- The product structure tree and its components: root, parts, branches, and leaves.
- The meaning of different levels of the product structure tree in a mechanical assembly.
- The relationship between parts in a product structure tree and defparts.
- Inputs in a product structure tree.
- Varying the product structure tree using the inputs.

The Product Structure Tree

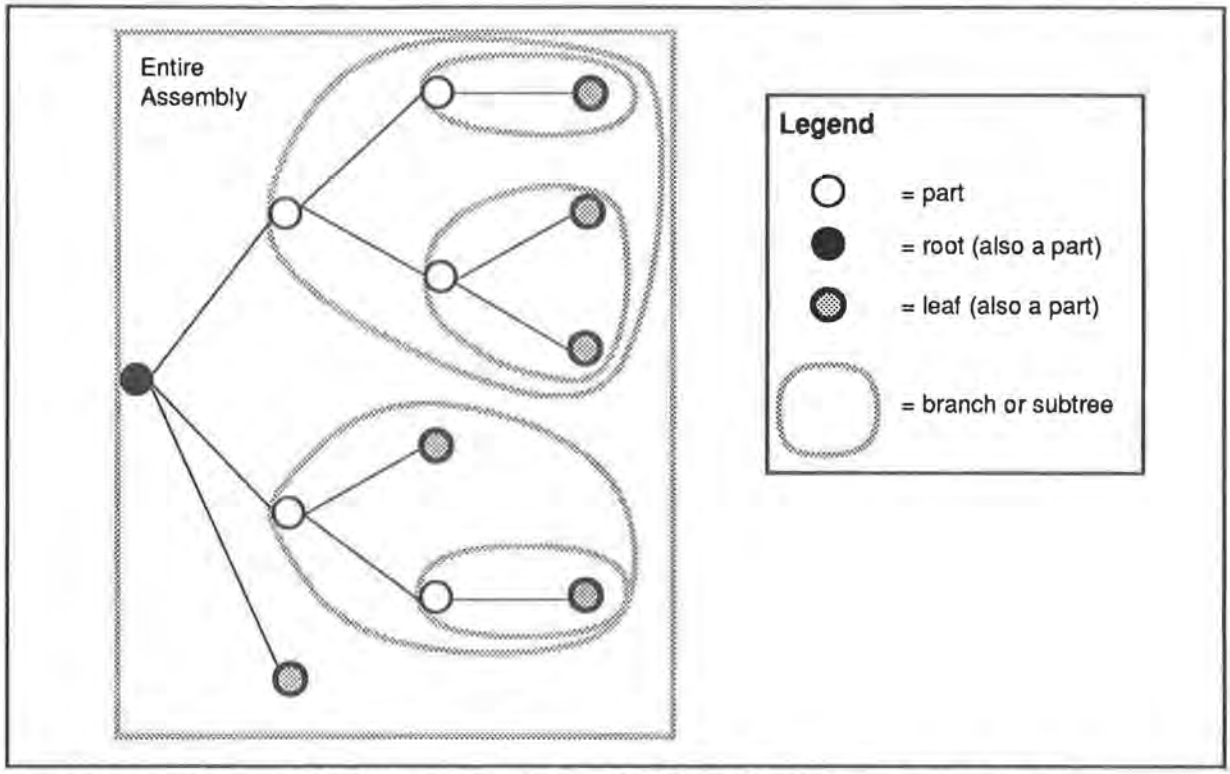
A design instance of an ICAD product model is organized into a tree structure called the *product structure tree.*

A tree organization naturally reflects the way that design and manufacturing engineers think about product design. It provides an unambiguous way of addressing every rule in the ICAD product model, which allows complex relationships to be set up between rules (see the section "Rule Dependencies in the ICAD Product Model", page 4-45). Tree organizations also provide a natural modularity to the product design. Complex assemblies are divided into subassemblies. Each subassembly can be designed separately, and each is further divided into components that can be designed separately.

A product structure tree links the components of a product design together. The *root* of the product structure tree is a design instance of the defpart that represents the entire product structure. The root branches out into more detailed levels which are called *children* or *parts* of the root. For example, a branch might represent a subassembly of the product assembly. Each child can have additional children. The terminal components of the tree are called *leaves*; these are parts that have no children.

Creating the Product Structure

The following diagram shows a basic tree structure with the root, branches, and leaves indicated. Each circle represents a part in the tree, which is a design instance of a defpart.



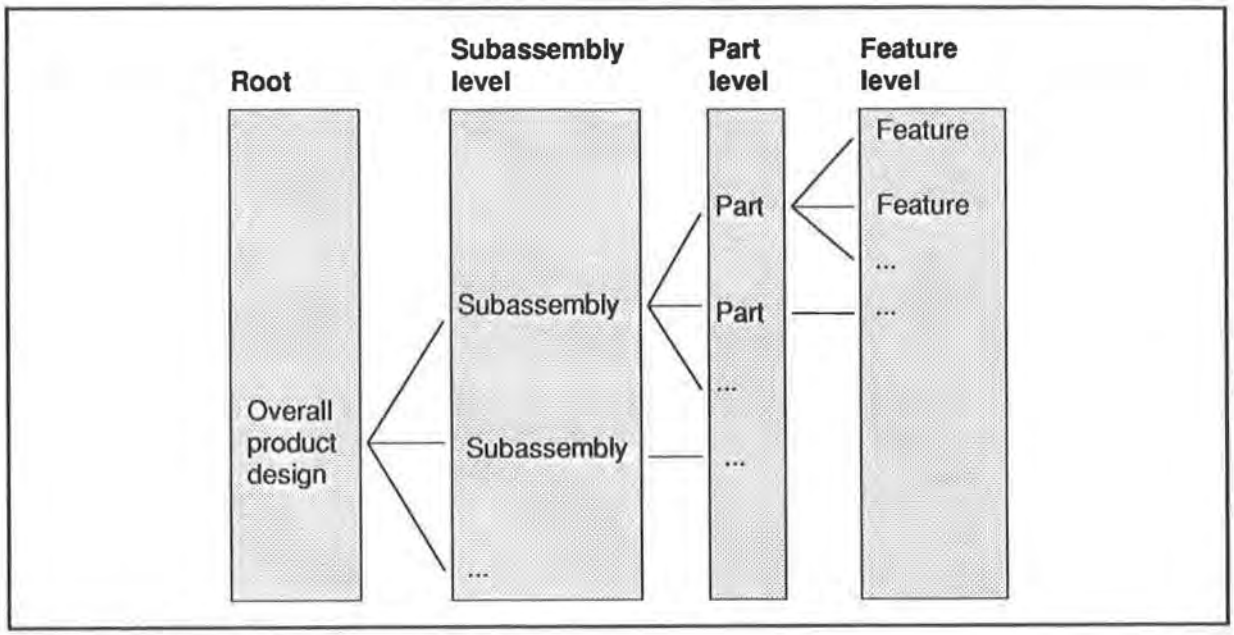
Basic tree structure. Each circle represents a part in the product structure tree, that is, a design instance of a defpart.

Levels of the Product Structure Tree

It can be helpful to think of the different levels of the product structure tree as roughly representing a number of conceptual levels in a mechanical assembly. The root represents the entire design instance, which is divided into the design of the part. This overall design decomposes into subassemblies. Subassemblies are typically built from component parts (e.g., screws, holes, etc). Parts are often divided into their component design features. The lowest level (leaves) of the product structure tree usually represents the actual geometric structure of the mechanical part (see the section "Modeling with Simple Geometric Primitives", page 4-56).

Creating the Product Structure

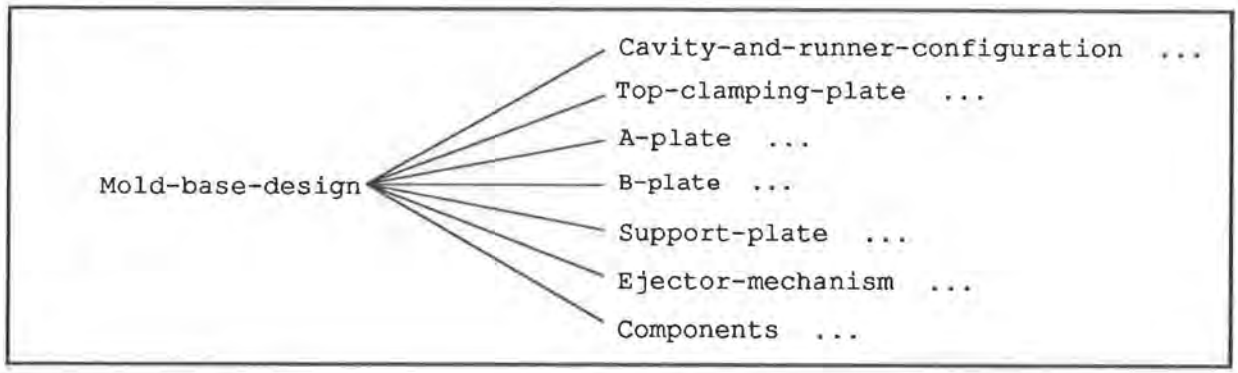
This is shown in the following diagram. Note that product structure tree of your ICAD product model may not follow this diagram exactly. Sometimes there are many levels of subassemblies before the component part level, and in some cases component parts are not divided into design features.



Different levels in a product structure tree.

Creating the Product Structure

If we look at the product structure tree of a design instance of the mold base, we find that the mold base is subdivided into five subassemblies. A subassembly represents the Cavity-and-runner-configuration, the Top-clamping-plate, the A-plate, the B-plate, the Support-plate, the Ejector-mechanism, and the Components (that is, screws, pins, and bushings). The following diagram shows the root of a mold base design instance with its children. Note that the complete product structure tree is not shown. The ICAD System generates part designs for the product structure tree one level at a time; the ellipses show that there may be lower levels of the product structure tree that have not been generated.[13]



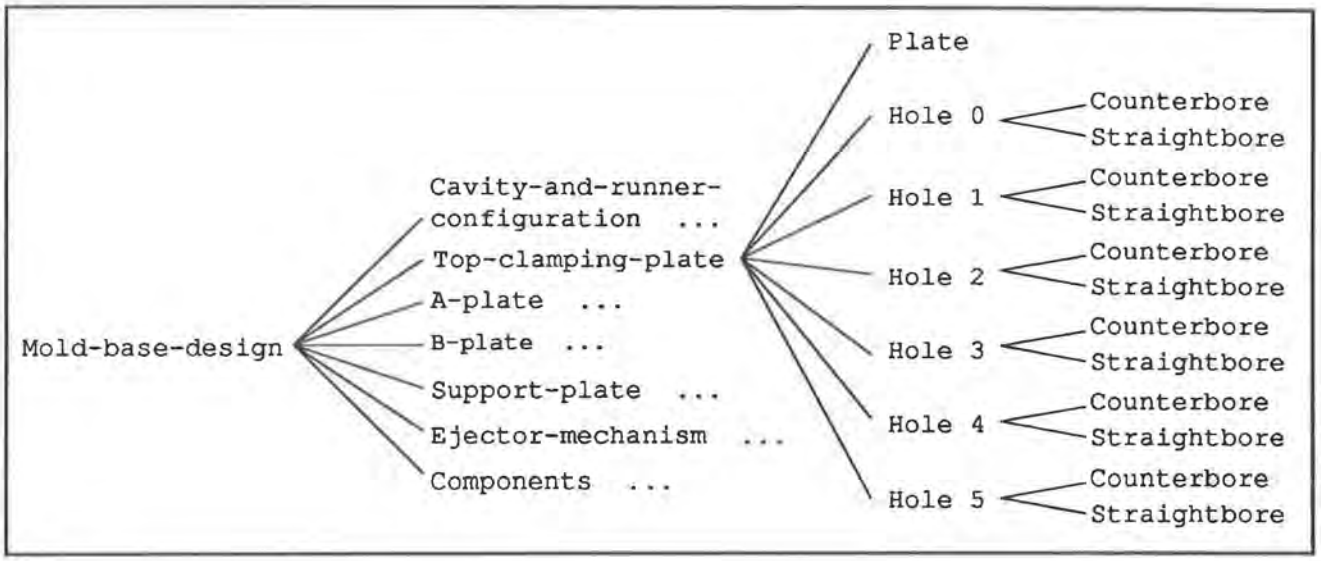
If we want to look at one of the children of the mold base in detail, we can ask The ICAD System to expand that portion of the tree. In this case, we ask to expand the Top-clamping-plate, and we find that the Top-clamping-plate has six children. One child, the Plate, represents a box with the same dimension as the Top-clamping-plate. The five other children represent holes in the Top-clamping-plate.

The Plate terminates the product structure tree; it directly represents the geometry of the product model. However, the Holes are subdivided into further children. Each Hole is composed of two features, a Counterbore and a Straightbore.

[13]This is another example of demand-driven evaluation.

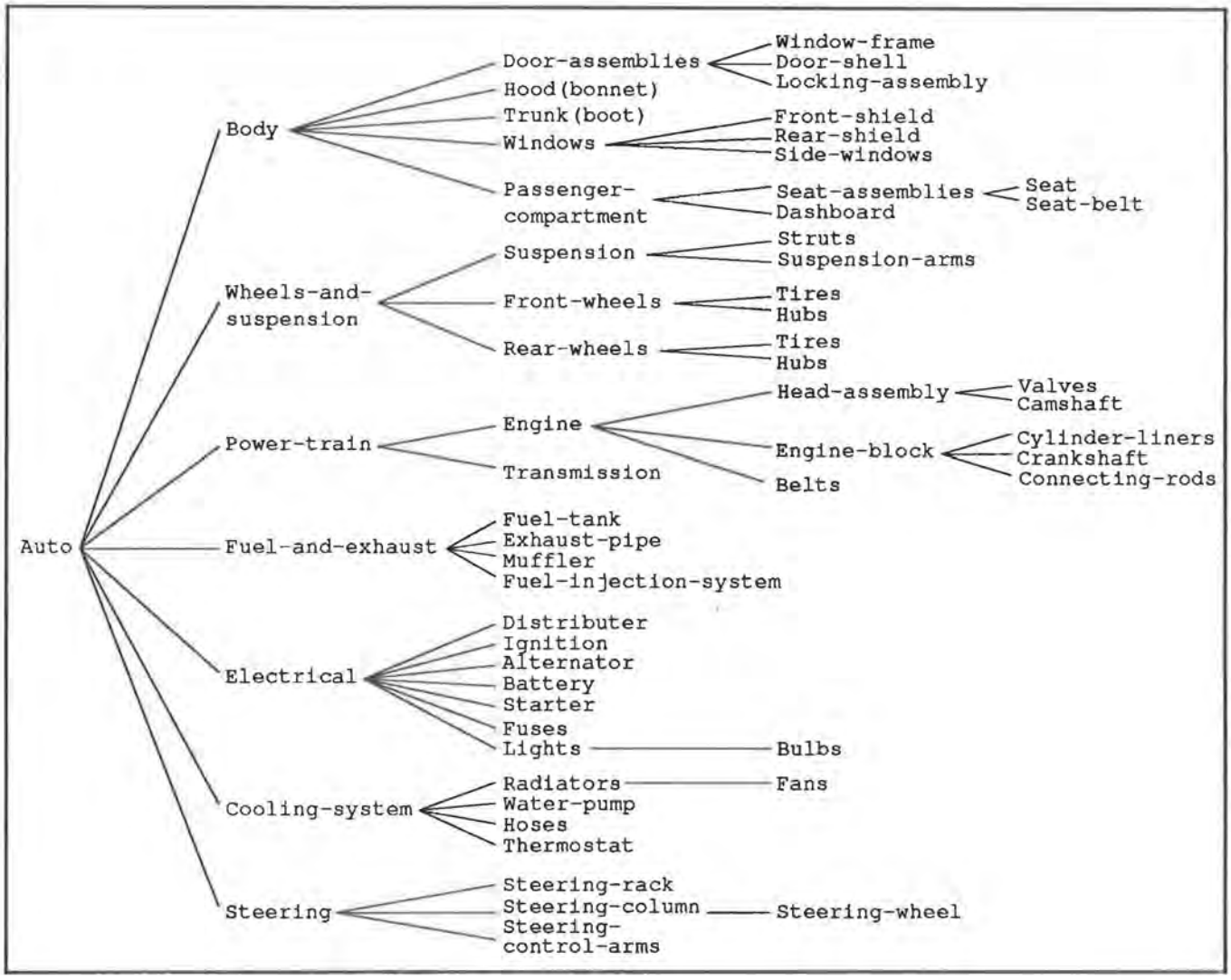
Creating the Product Structure

The following diagram shows the product structure tree for a Top-clamping-plate:



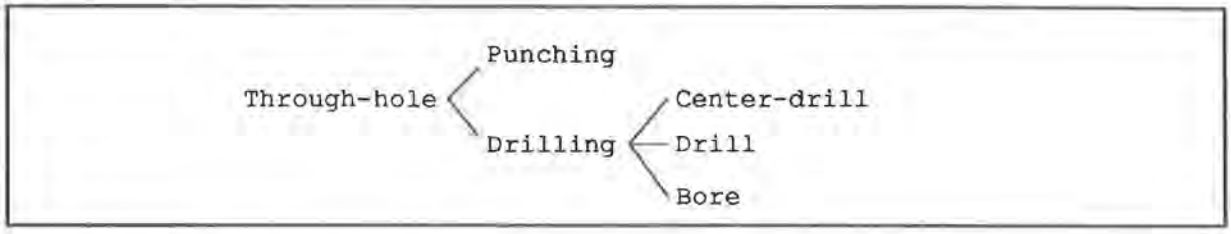
Creating the Product Structure

Other products have different product structures. A simplified product structure for an automobile, for instance, might look something like this:



Creating the Product Structure

A product structure tree often has components that represent process information. This product structure tree of a Through-hole, for instance, includes components that represent manufacturing processes such as punching and drilling:



Product Structure Trees and Defparts

Each part of the product structure tree is a design instance of a defpart. The product structure tree links the design instances into an overall design instance. Thus, the product structure tree consists of a tree of design instances, not defparts.

The root of the product structure tree is a design instance of the defpart that describes the overall ICAD product model. The name of the root in the product structure tree is the same as the name of the defpart that describes it.

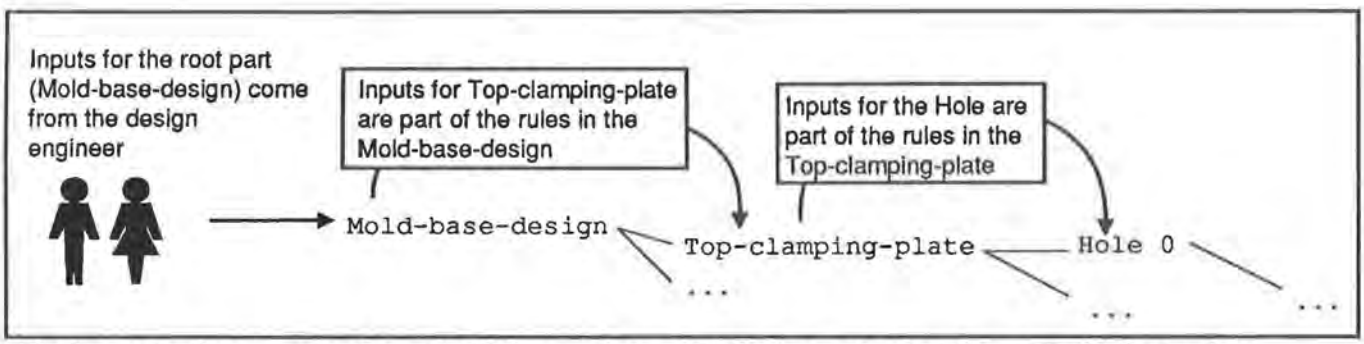
The root describes how it is divided into components. It does this by including the following items in its defpart:

- The names of each of the components. For example, the components of the mold-base are named Cavity-and-runner-configuration, Top-clamping-plate, A-plate, B-plate, Support-plate, Ejector-mechanism, and Components.
- The components *type*, that is, the defpart that defines the child. For example, the Top-clamping-plate, A-plate, B-plate, and Support-plate are all types of Mold-base-plate, that is, they are all defined by the Mold-base-plate defpart.
- The input specifications for each child. Since each component is a design instance of a defpart that can have input-attributes, the necessary inputs for each child must be specified in the definition of each child.

Notice that the mechanism of supplying inputs to a root defpart is quite different from the mechanism of supplying inputs to a lower-level defpart. You supply the inputs to the root defpart when you create the new design instance. However, the inputs to a lower-level defpart are part of the rules of the parent defpart. For example, when you create a new Mold-base-design instance as the root of the product structure tree, you supply the inputs to the new Mold-base-

Creating the Product Structure

design part. The inputs for each of the components, however, are specified in the rules for the Mold-base-design. This mechanism continues down the tree. For example, the inputs for each component of the Top-clamping-plate are specified in the rules for the Top-clamping-plate, etc. This is shown in the following diagram:



The inputs to the root come from the design engineer, but the inputs to the other parts in the product structure tree come from their parents.

Creating the Product Structure

The following is a simplified defpart that represents the mold base. It shows how the components of a design instance and their inputs are specified as rules in the defpart that creates the design instance.

```
(defpart Mold-base-design ( Basic-mold-data )
  :inputs
    (:Part-to-mold
     :Production-quantity )
  :parts
    ((Cavity-and-runner-configuration
       :type Cavity-layout )
     (Top-clamping-plate
       :type Mold-base-plate
       :Thickness 0.03
       :Number-of-holes 5)
     (A-plate
       :type Mold-base-plate
       :Thickness 0.05
       :Number-of-holes 5)
     (B-plate
       :type Mold-base-plate
       :Thickness 0.05
       :Number-of-holes 8)
     (Support-plate
       :type Plate
       :Thickness 0.03
       :Number-of-holes 8)
     (Components
       :type Components )
     (Ejector-mechanism
       :type Ejector-mechanism) ) )
```

Mold-base-design "mixes in" Basic-mold-data, which includes many of the design details.

The input requirements of the Mold-base-design

Following the "Parts" keyword are the components of the Mold-base-design.

The first component is the Cavity-and-runner-configuration. Its type is Cavity-layout, that is, it is a design instance of the Cavity-layout defpart.

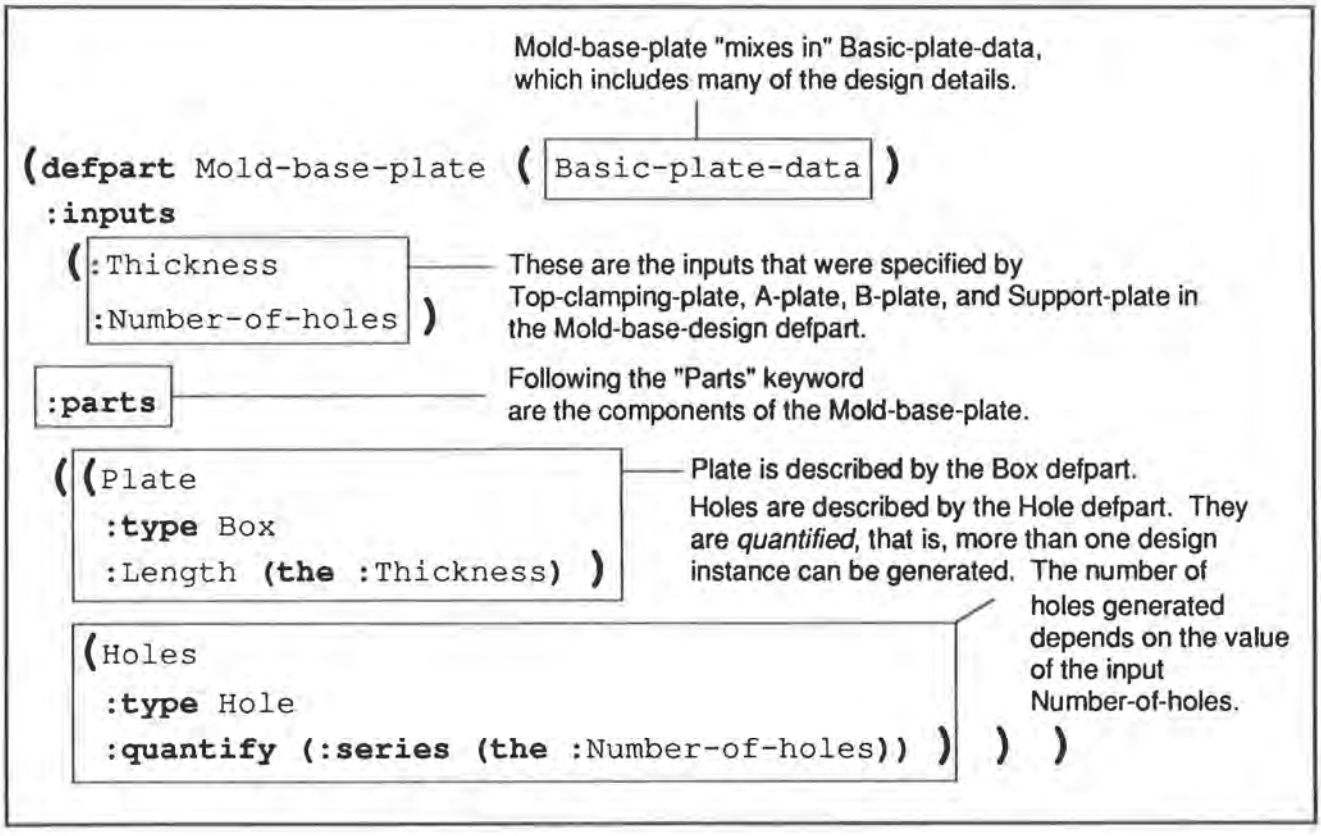
The second component is the Top-clamping-plate. Its type is Mold-base-plate. Since the Mold-base-plate requires inputs to generate a design instance, the inputs Thickness and Number-of-holes are specified here.

The third, fourth, and fifth components — A-plate, B-plate, and Support-plate — are also kinds of Mold-base-plates. The inputs to each component are different, however.

The sixth and seventh components — Components and Ejector-mechanism — have the same name as the defparts that describe them.

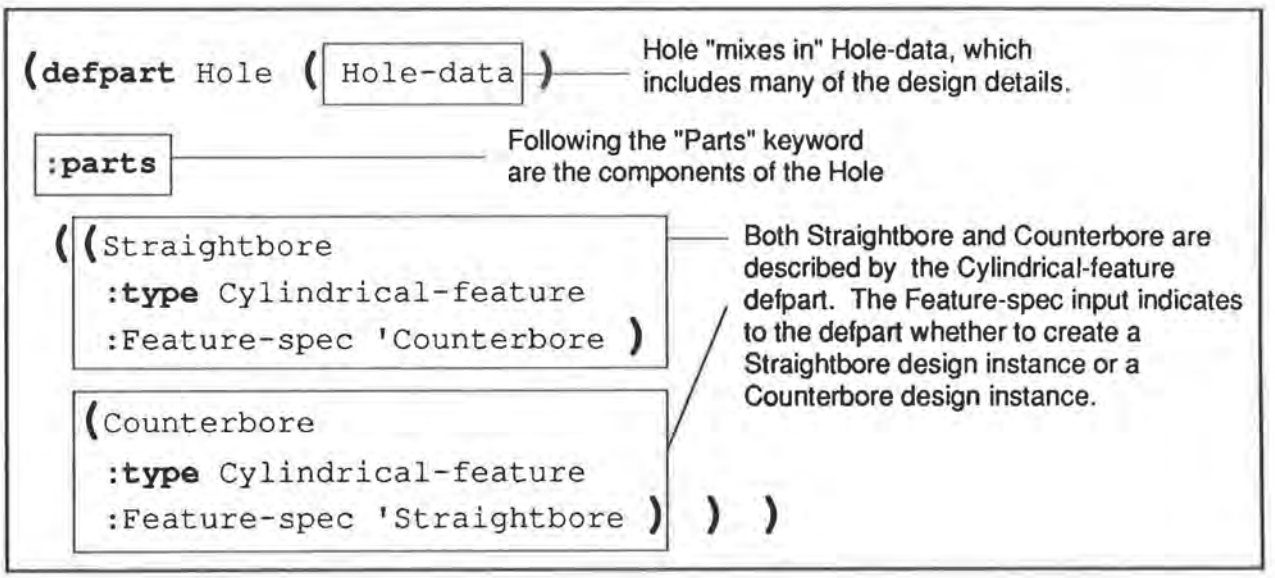
Creating the Product Structure

The Top-clamping-plate, A-plate, B-plate, and Support-plate are all described by the Mold-base-plate defpart. As we saw earlier, each Mold-base-plate has six children.



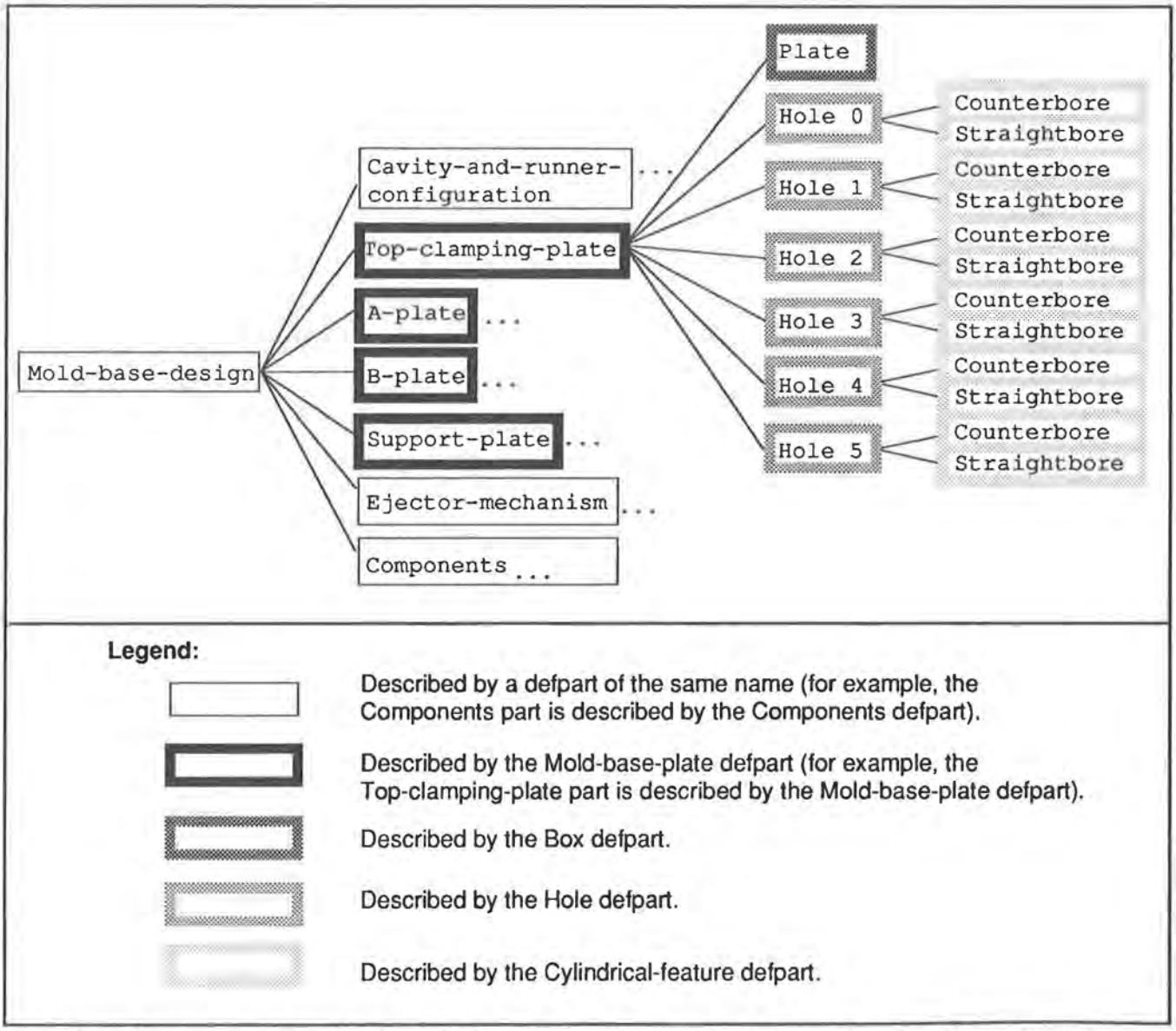
Creating the Product Structure

Each Hole is composed of a Straightbore and Counterbore, therefore the defpart that represents a hole includes rules that specify the Hole's children.



Creating the Product Structure

The following diagram shows how the product structure tree for a design instance of the Mold-base relates to the defparts that represent each part in the product structure tree. Note that there are many more parts in the product structure tree than there are defparts.



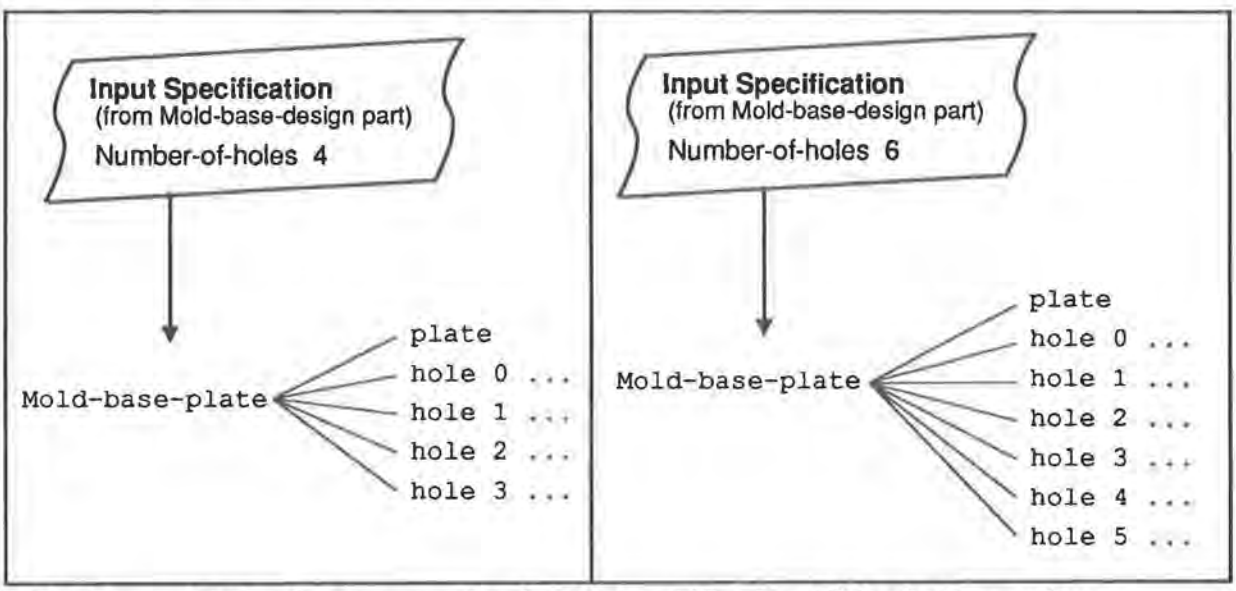
Partial product structure tree for the Mold-base-design showing the defparts that describe each component.

Creating the Product Structure

Varying the Product Structure Tree Based on the Inputs

The product structure described by a defpart is not fixed; its configuration is based on the input specifications.

For instance, one of the inputs to the Mold-base-plate defpart is Number-of-holes, that is, the number of holes in the plate. The value of this input changes the number of children the Mold-base-plate has. If Number-of-holes is four, then Mold-base-plate has five children — the box-shaped plate and four holes. If Number-of-holes is six, then Mold-base-plate has seven children — the box-shaped plate and six holes.



A Mold-base-plate has five components when the input Number-of-holes is four, and seven components when Number-of-holes is six.

In this example, the basic structure of the Mold-base-plate doesn't change drastically since the inputs to the Mold-base-plate only changes the number of holes in the plate.

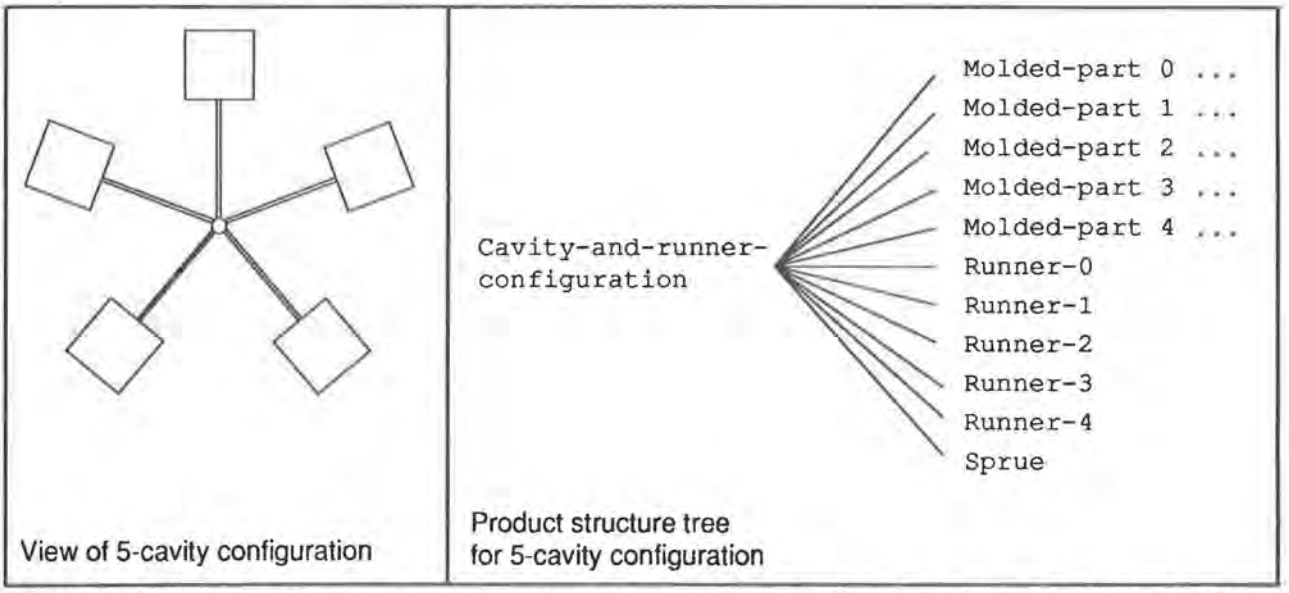
The Cavity-and-runner-configuration part of the Mold-base design instance illustrates a more dramatic structural change in the product structure tree. As discussed in the section "Design of a Plastic Injection Mold Base", one of the more complex aspects of a mold-base design is the layout of the cavities and runners. In the Mold-base product model, there are different defparts that represent different types of Cavity-and-runner-configurations. For instance, there are one-cavity configurations, two-cavity configurations, four-cavity configurations, etc. The actual type of Cavity-and-runner-configuration is chosen by The ICAD System when the product structure tree is generated.

Creating the Product Structure

The rules that the ICAD product model uses to chose a configuration can be summarized as follows: The type of injection-molding machine (selected by the design engineer) determines the maximum number of cavities that can be molded. The Cavity-and-Runner-Configuration part designs a different configuration for each of the possible number of cavities (from one to the maximum). The cost of each configuration increases as the number of cavities increases (since it costs more to mill additional cavities). However, the cost of production decreases as the the number of cavities increases (since you can produce more parts with each cycle). A cost-per-part attribute can be calculated by the following expression:

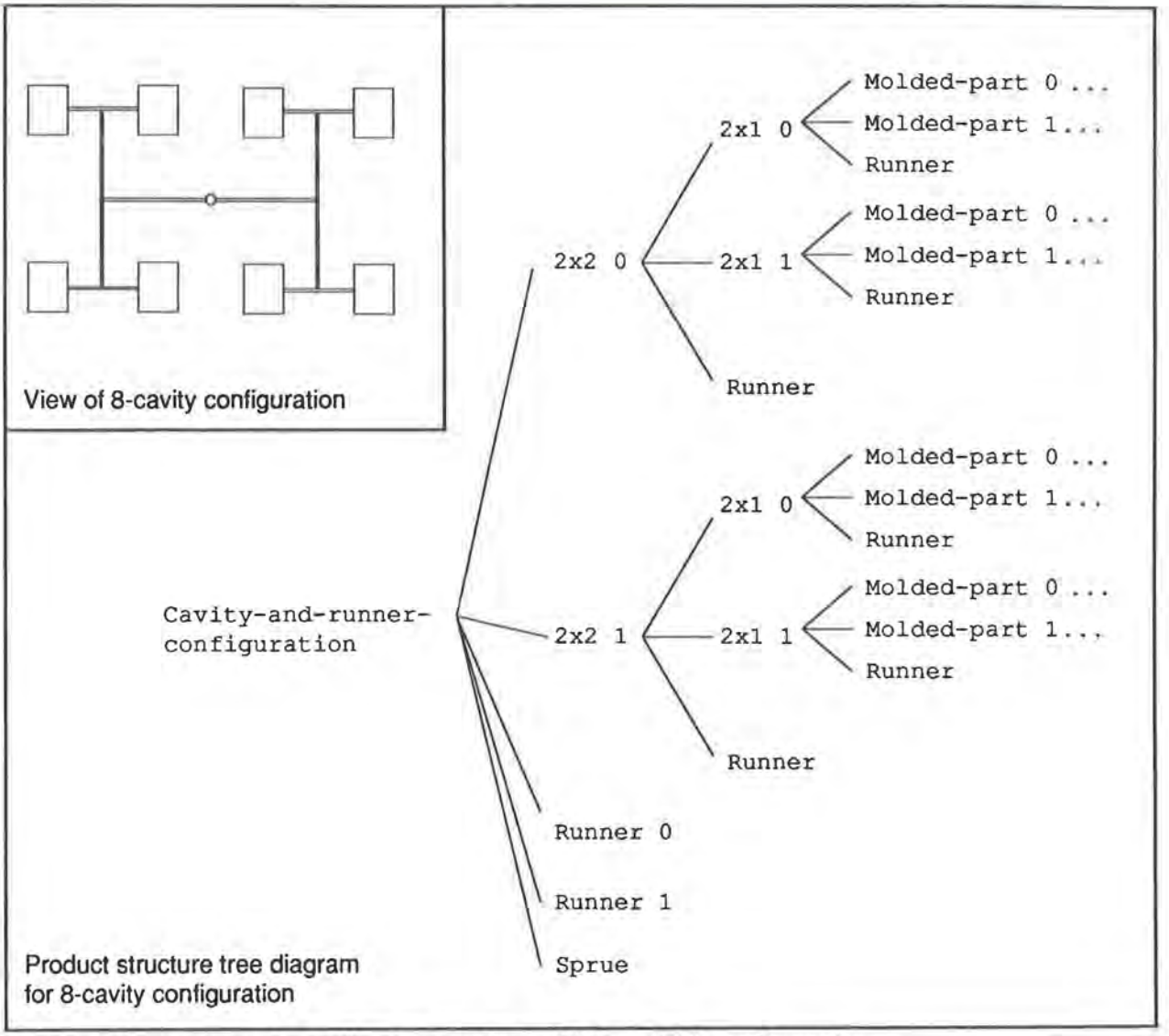
$$\frac{\textit{Cost-of-manufacturing-mold-base} + \textit{Total-production-cost}}{\textit{Total-number-of-parts-to-mold}}$$

The configuration that results in the least-expensive cost per part is chosen as the final configuration. The product structure tree reflects this final configuration.



Cavity-and-runner-configuration for 5-cavity mold base. This configuration is optimal for 200 000 parts to mold.

Creating the Product Structure



Cavity-and-runner-configuration for 8-cavity mold base. This configuration represents the optimal configuration for 3 000 000 parts to mold.

Summary

- The components of a design instance are organized into a product structure tree, which naturally represents the subdivision of a mechanical product into assemblies and subassemblies. The overall product is the root of the product structure tree. Product components are parts in the product structure tree.

Creating the Product Structure

- Different levels of the product structure tree correspond to different levels of the design. The root represents the overall product, which is typically divided into subassemblies. Subassemblies are composed of component parts. Parts are built out of features. The lowest level is the geometric primitives provided by ICAD.

- Each part in a product structure tree is described by a defpart. Defparts can include components by specifying the name of the component, the kind of part the component is (i.e., the name of the depart that defines it), and the inputs to the defpart that defines it.

- The product structure of a design instance can change as the input specifications to the defpart change.

Rule Dependencies in the ICAD Product Model

Keywords: *referencing, symbolic referencing, simple referencing chain, complex referencing chain, inherited values, descendant attributes, part-whole defaulting*

Overview

This section shows ICAD Design Language techniques for relating engineering attributes that belong to different parts of an assembly and includes the following topics:

- Describing relationships between components.
- Symbolic referencing between attributes in the same defpart.
- Symbolic referencing between different components of a product structure tree.
- Passing data between a component in the product structure tree and its descendants.

Relationships Between Attributes

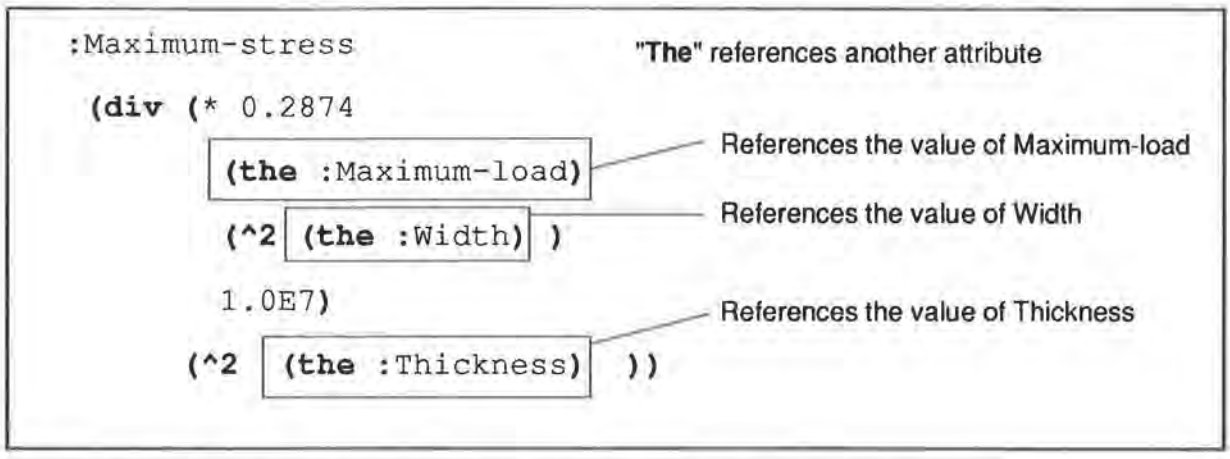
Most engineering is concerned with relationships between different components and attributes of an assembly. In a mold base, for instance, the size and shape of the runners depend on the temperature of the molten plastic, as well as the distance the plastic has to travel to reach the cavities. These relationships are described in a defpart by having the rule that defines one attribute refer to the dependent attribute. This is known as *symbolic referencing*, since the rule refers to the dependent attributes by name.

Symbolic Referencing Between Attributes in the Same Defpart

When a rule refers to an attribute defined in the same defpart, it only needs to give the attribute name.

Rule Dependencies in the ICAD Product Model

In the Mold-base-plate defpart shown earlier, for example, the value of the Maximum-stress depends on the value of the Maximum-load, Width, and Thickness, which are all referenced using simple referencing chains:



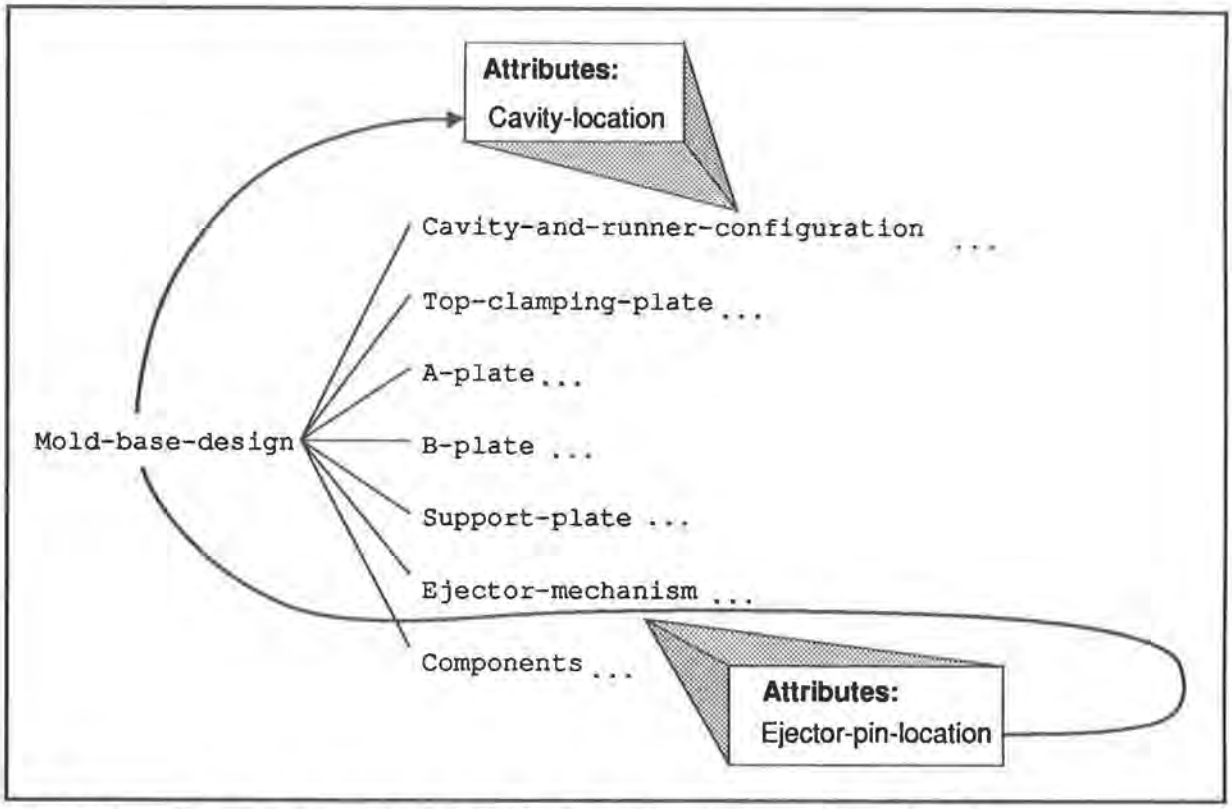
Symbolic Referencing Between Attributes in Different Defparts

One of the benefits of a product structure tree is that it allows any attribute anywhere in the tree to refer to any other attribute elsewhere in the tree. You refer to the attribute symbolically by describing the relative location of the attribute in the tree.

In the mold base, for instance, the location of an ejector pin in the ejector mechanism depends on the cavity layout. The location of each cavity in the cavity layout is defined as an attribute called Cavity-location in the definition of the Cavity-and-Runner-Configuration part. The attribute that defines the location of each ejector pin is defined as an attribute called Ejector-location in the Ejector-mechanism part. Using symbolic referencing, the definition of the Ejector-location can depend upon the value of the Cavity-location even though the two attributes are defined in different parts of the tree.

Rule Dependencies in the ICAD Product Model

The following diagram shows the relative location of the two attributes in the product structure tree:



The Ejector-pin-location attribute in the Ejector-mechanism part depends on the Cavity-location attribute in the Cavity-and-runner-configuration part.

Symbolic Referencing Using Descendant Attributes

It is very common for a component to require attribute values from components above it in the product structure tree. The normal mechanism for passing such data down the product structure tree is input-attributes, that is, a component can pass data to its subcomponents by supplying inputs to each subcomponent.

The ICAD Design Language provides another mechanism for passing data down a product structure tree called *part-whole defaulting.*[14] With this mechanism, a defpart can declare some of its attributes as *descendant.* This means that the values of those attributes can be accessed by another attribute that is lower in the product structure tree. Part-whole defaulting makes it easy to define a rule whose value is propogated through the entire product structure tree, and is thus available to any descendant of the part using a simple referencing chain.

[14]One of the two inheritance mechanisms implemented in the ICAD Design Language. The other, kind-of inheritance, is described in the next section.

Rule Dependencies in the ICAD Product Model

Summary

- Attributes can reference other attributes defined in the same defpart using simple referencing chains.
- An attribute in one part of the product structure tree can access the value of an attribute in another part of the product structure tree using a complex referencing chain.
- An attribute in a top-level part in the product structure tree can pass its definition down the tree to all of the descendants of the part using part-whole defaulting.

Modularizing the ICAD Product Model

Keywords: generic parts, kind-of inheritance, modularization, mixins

Overview

This section discusses some of the features of The ICAD System that are used to modularize the design of the ICAD product model and includes the following topics:

- Benefits of modular design.
- Components in the product structure tree.
- Generic parts.
- Kind-of inheritance.

Benefits of Modularization

Modularization means you describe the ICAD product model using small, easy-to-understand defparts instead of using large, specialized defparts. A modular ICAD product model minimizes the number of different defparts that are necessary as well as the size of each defpart.

A modular ICAD product model is robust and easy to maintain. This is because the component defparts of a modular ICAD product model are small, easy to understand, easy to test, and easy to use.

ICAD provides three strategies for building modular ICAD product models:

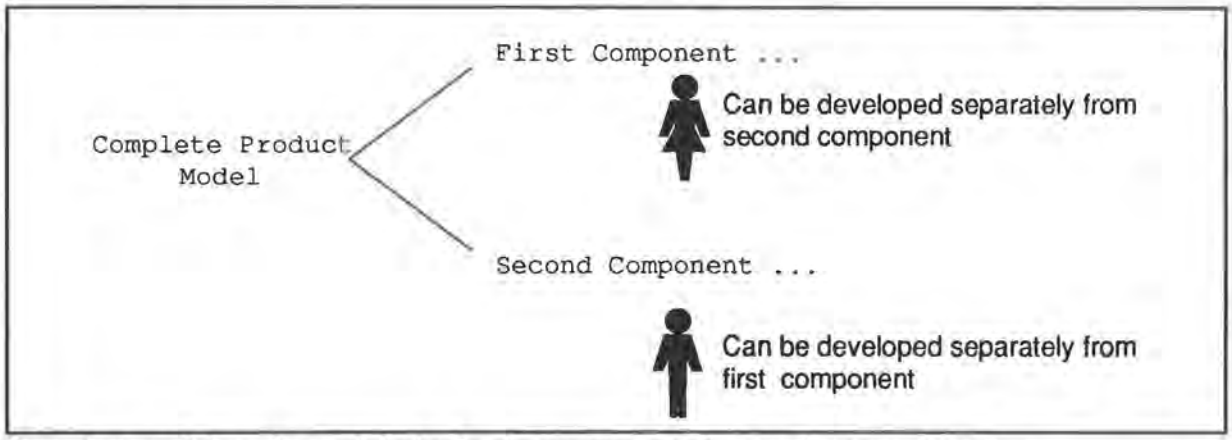
- Using the product structure tree to divide the ICAD product model into separate components.
- Using *generic parts*, defparts that can be used many times in different components of the same ICAD product model.
- Using *mixins*, which allows new defparts to be written using the definition of an existing defpart.

You use all three of these strategies to implement the component problems isolated in the planning stages of the application (see the section "Planning an ICAD Product Model", page 4-4).

Modularizing the ICAD Product Model

The Product Structure Tree

The product structure tree provides the product design with a natural modularity. Each assembly in the product design can be developed separately — in fact, they can be developed by different engineers at different times. It is easy to integrate separate assemblies; you simply define them as children of the same defpart. Likewise, each assembly can be further subdivided, with each piece being defined separately and integrated as parts of the assembly. You can subdivide an assembly to any extent that you like; The ICAD System puts no limit on the number of possible levels in a product structure tree.



A product model can be modularized using components in a product structure tree.

Generic Parts

A *generic part* is a defpart that is developed as a stand-alone component, that is, it is not defined relative to any other part in the product structure tree. This allows you to use the generic part throughout the product structure tree. Typically, a generic part includes a large number of input requirements.

A generic part often represents a component that is commonly used in the assembly. The individual design instances of each component may differ, e.g., they may be a different size, shape, material, etc., but the component is basically the same in all places.

For instance, the mold base uses a number of different kinds of screws throughout the assembly. The screws differ in diameter, length, and type of head (phillips or slotted). While it is possible to write a different defpart for each type of screw used in the Mold-base, this increases the number of defparts you need to write for the ICAD product model, making the ICAD product model harder to maintain.

A better solution is to define a generic screw whose common properties are constant and whose differing properties are specified by inputs. Such a screw is

Modularizing the ICAD Product Model

generic since the same defpart can be used whenever a screw is needed. Inputs for a generic screw include the type of head, the screw-head diameter, and the screw length.

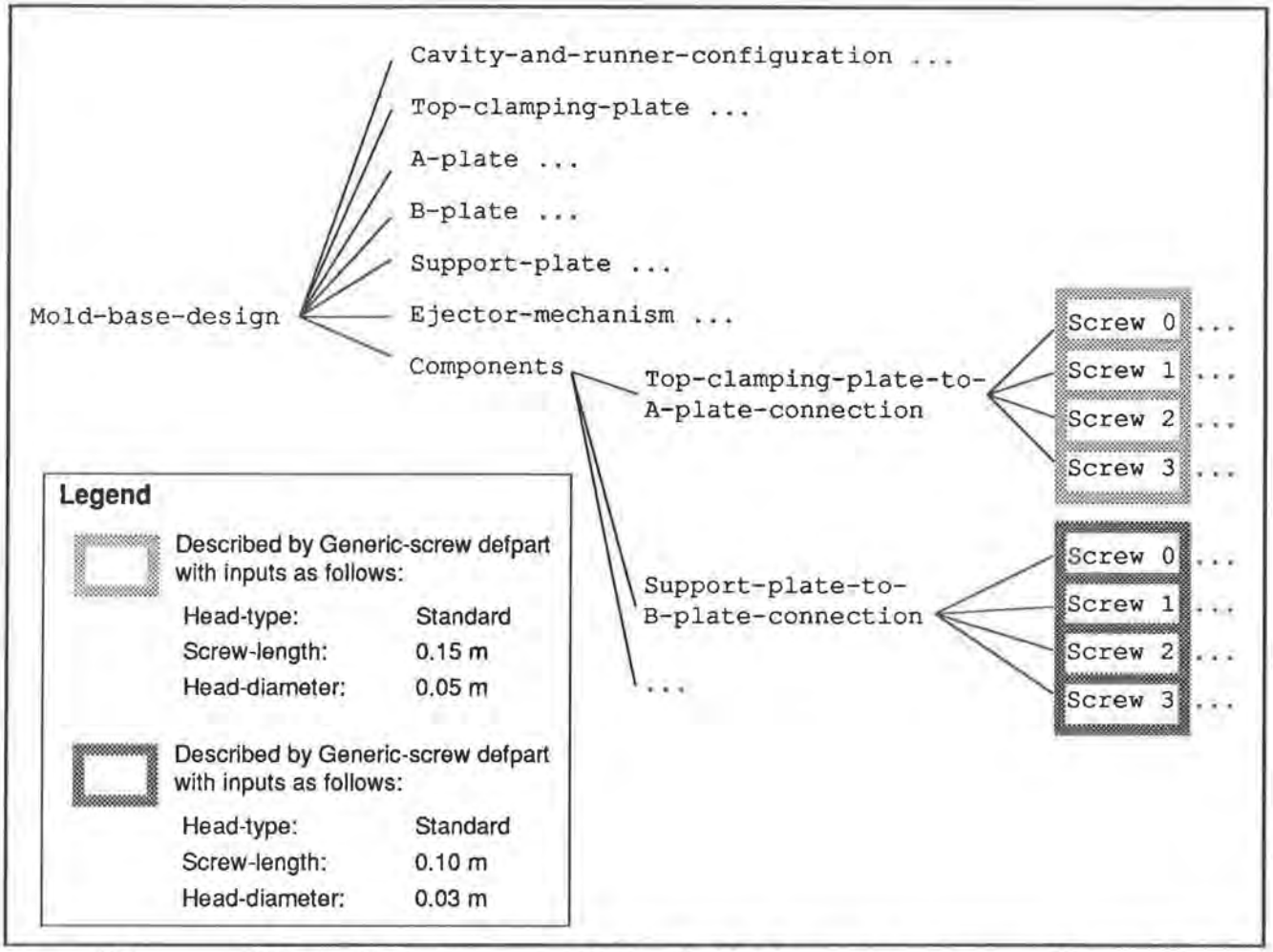
The following defpart describes a generic screw:

```
( defpart generic-screw ( box )
  :inputs
  (:head-type :screw-length :head-diameter)
   :parts
   ((screw-head
     :type screw-head
     :head-radius (half (the :head-diameter))
     :head-type (the :head-type) )
    (screw-body
     :type screw-body
     :head-radius (half (the :head-diameter))
     :head-length (the :screw-length) )))
```

By changing the inputs, you change the kind of screw.

Modularizing the ICAD Product Model

In the Mold-base, one type of screw is used to fasten the Top-clamping-plate to the A-plate, and another kind of screw is used to fasten the B-plate to the Support-plate. The lengths and diameters of the screws are determined by a rule that sums the thickness of the plates. However, the same generic screw defpart describes both types of screws.



The mold base uses different kinds of screws, each described by the same generic-screw defpart with different inputs. In this product structure tree, the screws used to hold the Top-clamping-plate to the A-plate are different than the screws used to hold the Support-plate to the B-plate.

Each application has its own set of useful generic parts. Generic parts for a holding tank, for instance, include beams, bolts, and braces. Generic parts for an engine assembly include connecting rods and belts. These are all components that are used in different forms throughout the assembly.

Modularizing the ICAD Product Model

Mixins

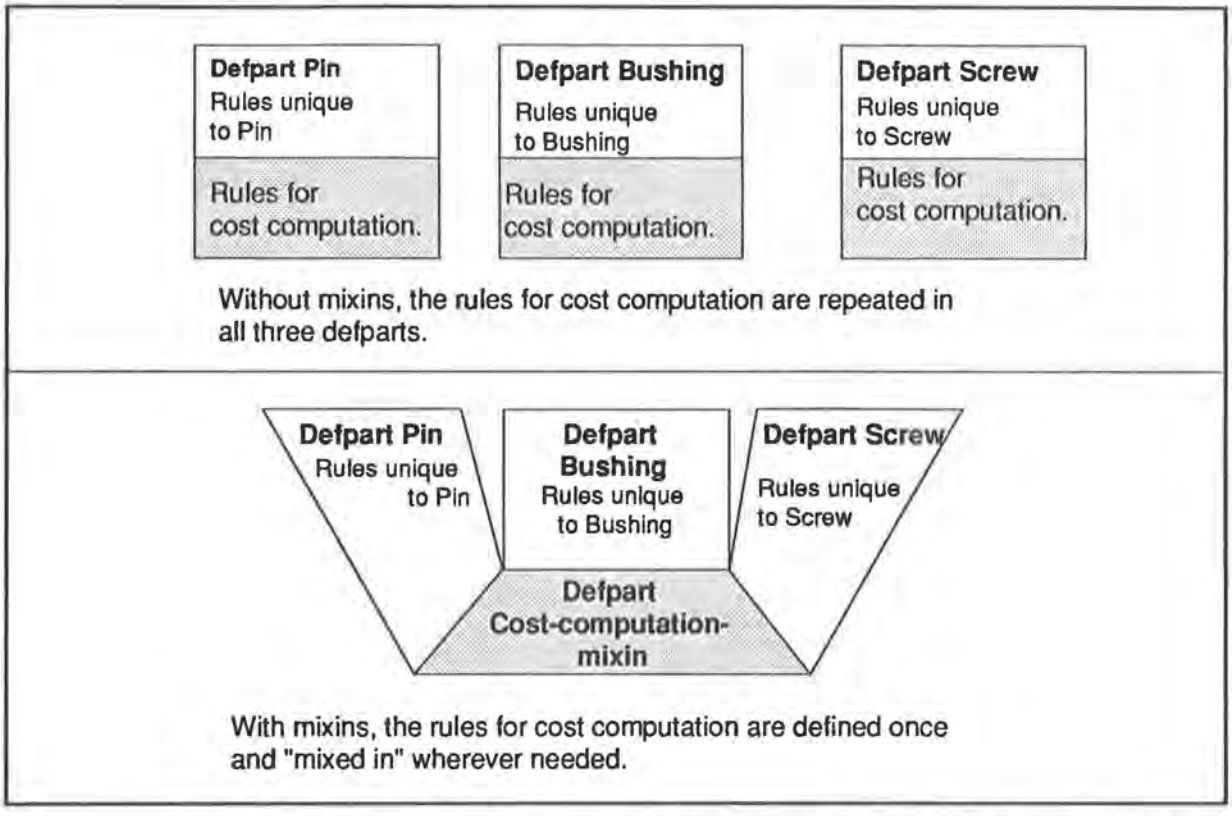
Defparts are rarely created from scratch. Instead, their definitions usually build upon pre-existing definitions. *Mixins* allow a new defpart to use the definitions (that is, the rules and component) that are already described by an existing defpart. You can mix the definition of an existing into the new defpart.

This capability allows you to create a new defpart that is just like an existing defpart except for some small differences. Only those characteristics that are different need to be specified. We say that the new defpart *inherits* some of its definition from the existing defpart. This is known as *kind-of inheritance*, since the new defpart is a "kind of" existing defpart.[15]

[15]One of the two inheritance mechanisms implemented in the ICAD Design Language. The other, part-whole inheritance, was described in the previous section.

Modularizing the ICAD Product Model

In the mold-base application all defparts that represent component parts of a mold (such as pins, bushings, screws, etc.) must include rules for computing cost information. In most cases the technique for computing the cost is the same. Rather than repeat the same rules in each defpart, you can create a Cost-computation-mixin defpart and mix it into the defparts for the pins, bushings, screws, etc. This is shown in the following diagram:



Often mixins are used to divide the information that would normally be in a single defpart into separate defpart modules. A single defpart that includes hundreds of attributes would be tiresome to write, hard to read, and difficult to maintain. Instead, the different attributes are divided into logical categories, and the attributes in each category are defined in separate defparts.

Although the Mold-base-plate defpart discussed earlier is relatively simple, it includes two different categories of rules: geometric rules (e.g., the rules that describe the Thickness, Width, and Height), and engineering rules, (e.g., the rules that describe the Maximum-load and Maximum-stress). Using mixins, we can write a Mold-base-plate-geometry defpart that describes the geometry of the Mold-base-plate and a Mold-base-plate-stress defpart that describes the load and stress on the plate. Then we can mix the two together to form the Mold-base-plate defpart.

Modularizing the ICAD Product Model

Summary

- The ICAD Design Language includes tools for modularizing the ICAD product model design. A modular design is easier to write, clearer to understand, and simpler to maintain, and thus accelerates the process of developing the ICAD product model. The planning stage of the ICAD product model usually identifies most of the modules.
- The product structure tree provides a basic modularity to the product design.
- Generic parts are defparts that model commonly-used component parts in the product model. Using generic parts simplifies the ICAD product model.
- Mixins allow a new defpart to inherit definitions from an existing defpart. This allows defpart definitions to be modularized.

Modeling with Simple Geometric Primitives

Keywords: geometric primitives, positioning, orienting, box, length, width, height, 2D wire-frame primitives, 3D volumetric primitives, local coordinate system, bounding box

Overview

This section shows how you incorporate 2D and 3D geometry into an ICAD product model and includes the following topics:

- ICAD-defined geometric primitives.
- Positioning and orienting geometric parts.

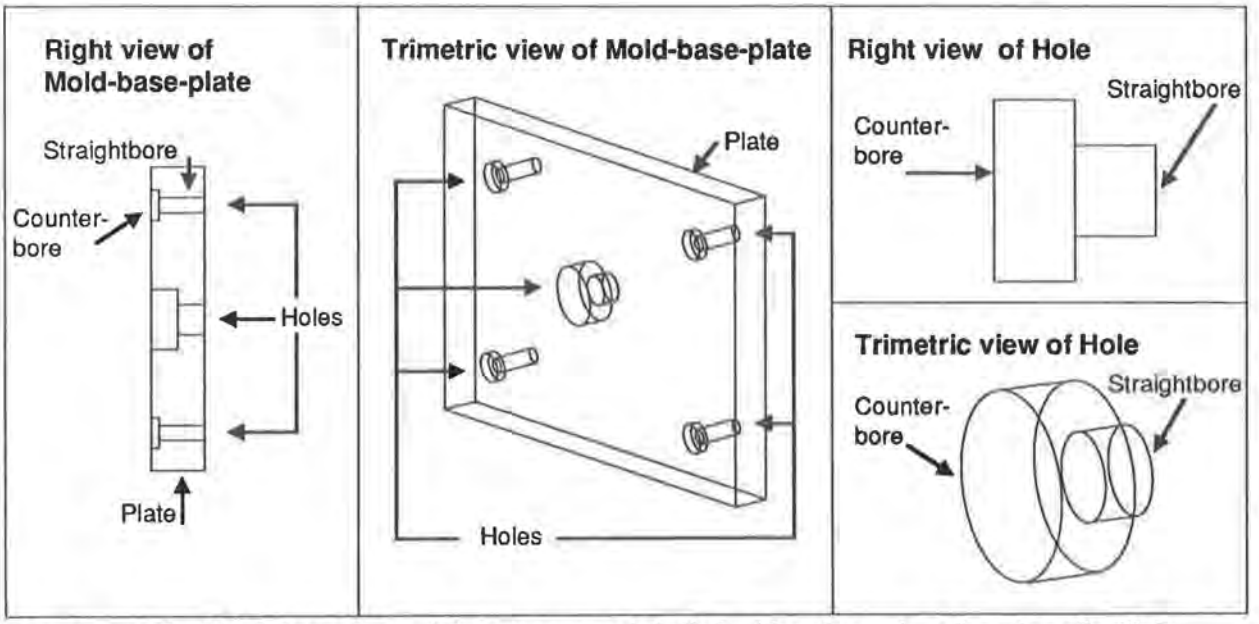
Geometric Primitives

Geometric primitives provided by ICAD represent the physical shape of objects. The set of primitives can be used by different applications in many different industries. ICAD's focus has been to provide primitives for mechanical-design applications.

Mixing an ICAD-provided primitive into your own defpart gives it geometric meaning. This implies that the part can be drawn in The ICAD System or transferred to a CAD system. Usually the leaves of a product structure tree represent the actual geometry of the product.

Modeling with Simple Geometric Primitives

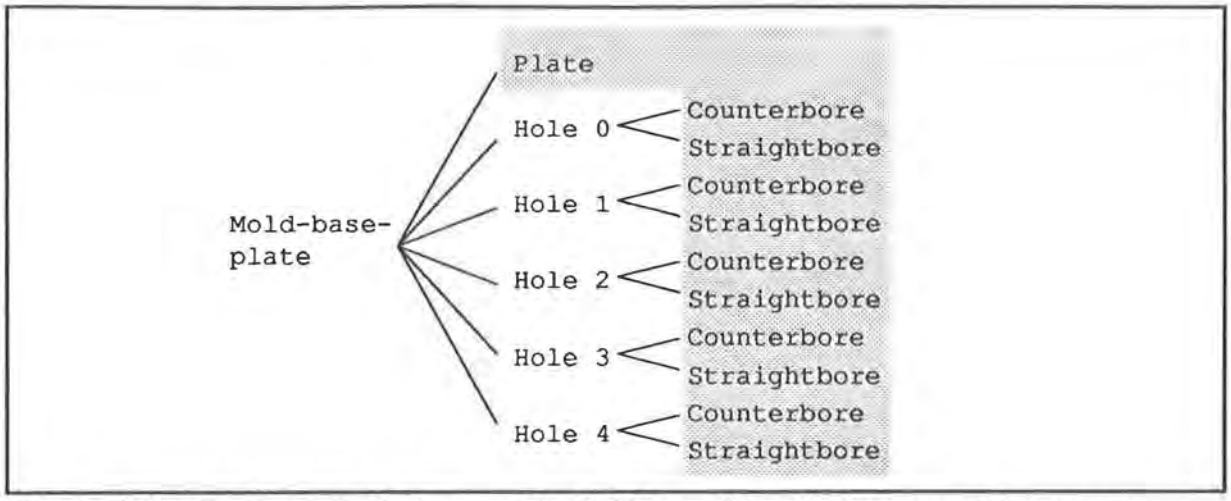
In the mold base application, the leaves of the Mold-base-plate in the Mold-base-design are the Plate, Counterbores and Straightbores. The Plate is a kind of a Box (that is, it mixes in the ICAD-provided Box defpart), which gives it a box-like geometric shape. The Counterbores and Straightbores are kinds of Cylinders (that is, they mix in the ICAD-provided Cylinder defpart) which gives them cylindrical kinds of shapes.



Drawing of the Mold-base-plate with components labelled. Also shown is a drawing of a Hole, which includes a Counterbore and Straightbore. The different parts are modeled using ICAD-provided geometric primitives.

Modeling with Simple Geometric Primitives

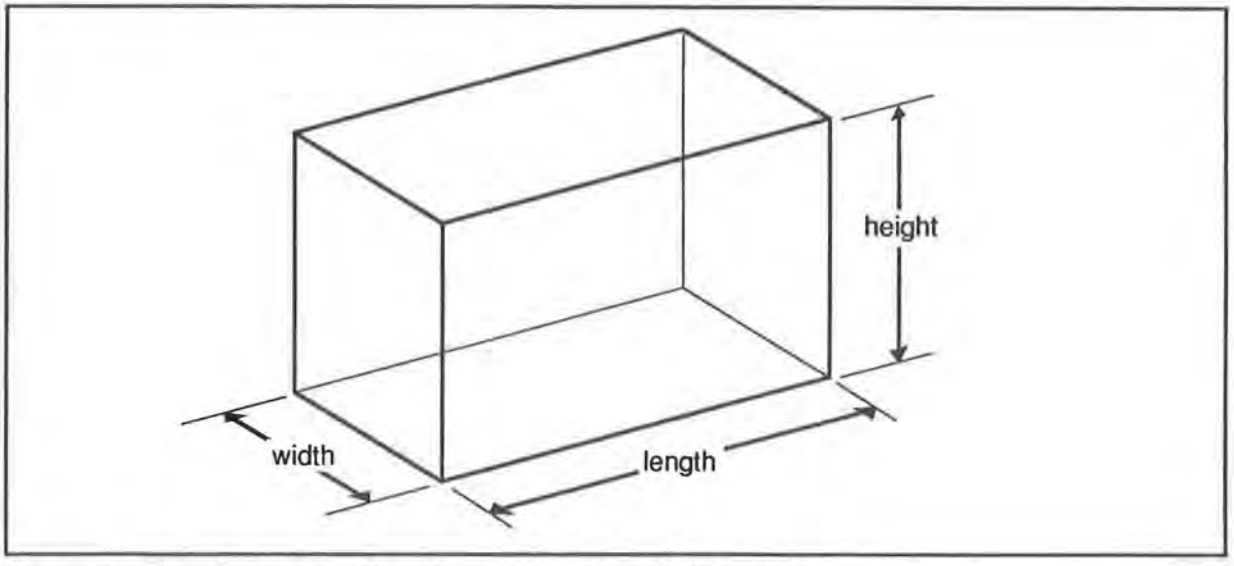
The following diagram shows how the leaves of the product structure tree represent the part's geometry.



The shaded area represents the leaves of the Mold-base-plate, which represent the geometry of the part.

The Box

The most basic ICAD-defined primitive part is a box, which is described by its length, width, and height.

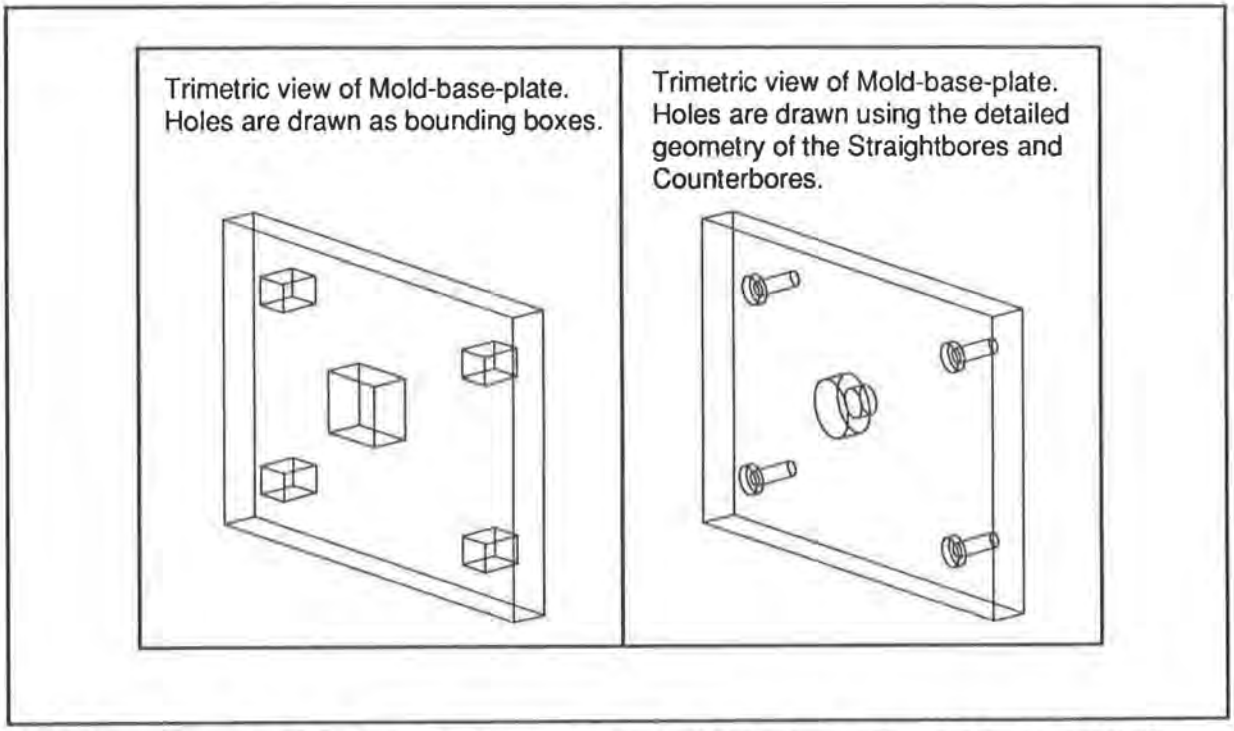


Box with major dimensions (length, width, and height) labelled.

Modeling with Simple Geometric Primitives

Often the higher-level parts of the product structure tree are kinds of boxes. This allows them to model the geometry of the "bounding box", which is a box that encloses the geometry of the leaves. By drawing the bounding box, you can roughly see the geometry of the assembly without having to draw the detailed geometry of the leaves.

In the mold-base application, the geometry of the Hole assembly is a bounding box that includes the geometry of the Counterbore and Straightbore.

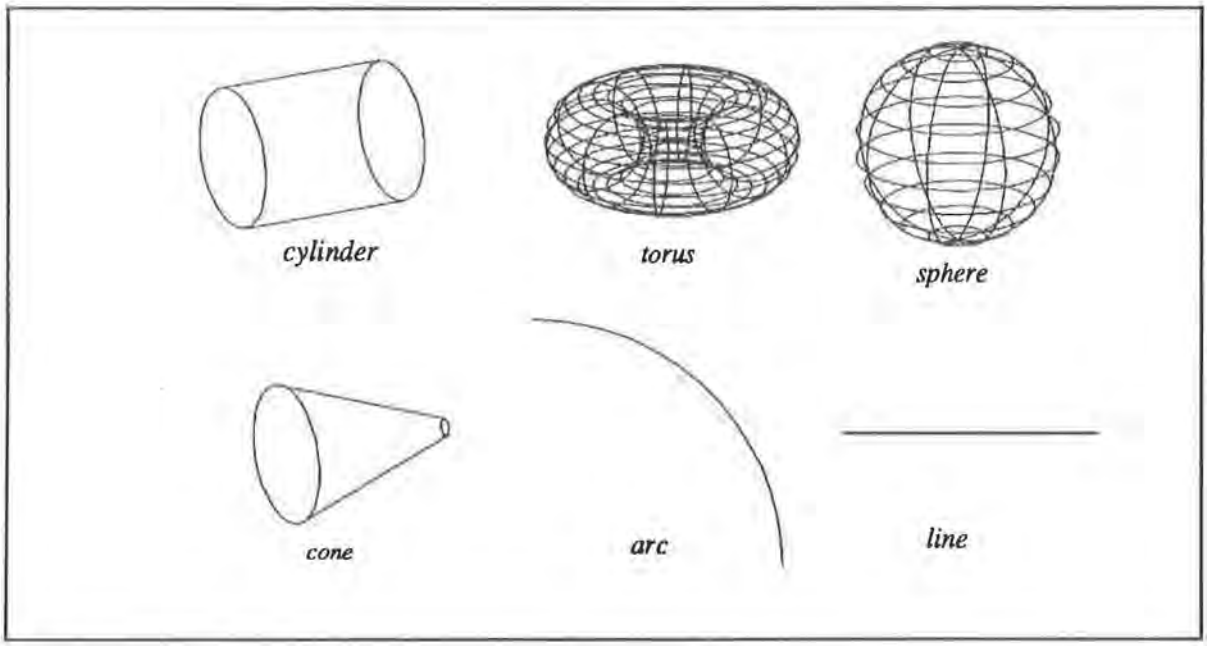


Mold-base-plate. On the left, just the bounding boxes of the holes are drawn. On the right, the detailed geometry of the holes, that is, the Straightbores and Counterbores, is drawn.

Modeling with Simple Geometric Primitives

Other Geometric Primitives

ICAD provides many other commonly-used geometric primitive parts. The standard 2D wire-frame primitives include lines, circles, arcs, ellipses, points, and polylines. 3D volumetric primitives include boxes, cones, cylinders, cut cylinders, projected polygons, spheres, and tori. ICAD also provides standard high-level parts, including a route pipe through a set of points, and a pipe elbow between two straight pipes.



Examples of 2D and volumetric primitives provided by ICAD.

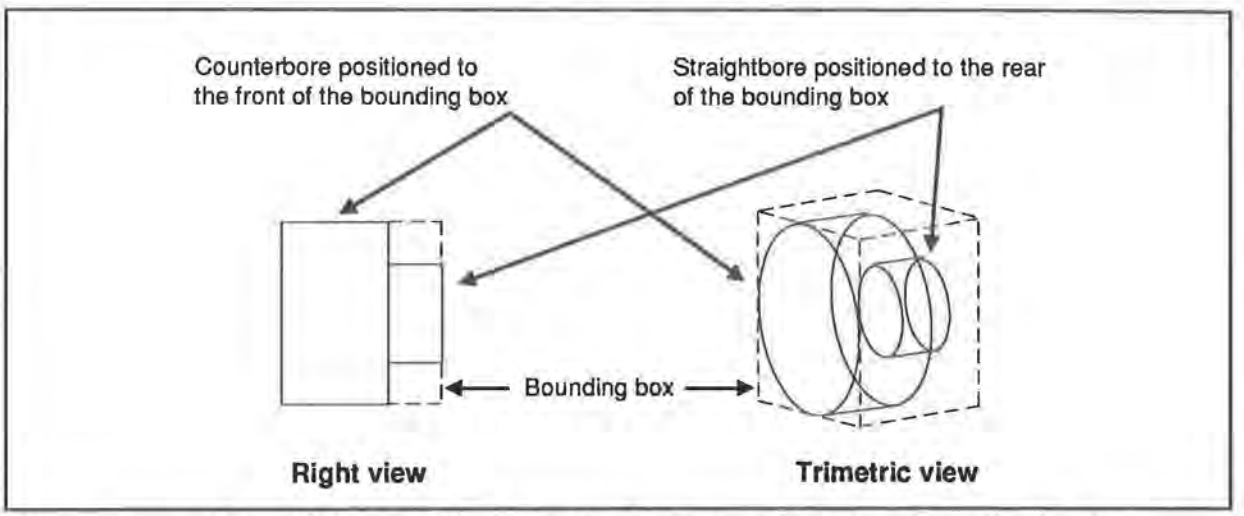
Orientation and Positioning of Geometric Parts

Parts are always positioned and oriented relative to the assembly of which they are children. This allows you to design an entire assembly without having to consider the position and orientation of the subassembly in the overall assembly. You specify the position and orientation of the subassembly as inputs to the subassembly.

Since the part is positioned and oriented relative to the parent assembly, we say that the parent provides a *local coordinate system* for its children, that is, it supplies the orientation of the three major axes. This is why many top-level parts are a kind of Box, since a Box makes it easy to visualize the local coordinate system. A *global coordinate system* describes the coordinate system of the overall design in which the major subassemblies are positioned and oriented.

Modeling with Simple Geometric Primitives

For example, each Hole in the Top-clamping-plate is made of a Counterbore and a Straightbore, both of which are a kind of Cylinder. The Counterbore and the Straightbore are positioned and oriented relative to the Hole assembly, which defines a bounding box for these two parts. The Counterbore is positioned at the front of the bounding box defined by the Hole, and the Straightbore is positioned at the rear of the bounding box defined by the Hole.



Counterbore and Straightbore positioned inside a Hole assembly, whose bounding box is indicated using dashed lines.

The Hole is really a generic part; you can position and orient each Hole in the Top-clamping-plate (and anywhere else in the mold base) as an overall assembly. The positioning and orientation of the Straightbore and Counterbore inside the Top-clamping-plate occur automatically, since they are defined relative to the Hole.

Summary

- ICAD-supplied geometry primitives can be mixed into a user-defined part to give the part geometric meaning. Typically, the leaves of the product structure tree represent the geometry of the product.
- The most basic geometric primitive is a Box. ICAD provides many other geometric primitives as well.
- You position and orient a geometric part relative to its parent. The parent supplies a local coordinate system in which the part is positioned.

Additional ICAD-supplied Geometric Parts

Keywords: ICAD Surface Designer, ICAD Solids Designer, parametric curves and surfaces, B-splines, solid modeling, mass and volumetric properties

Overview

ICAD offers products that provide additional geometric primitive parts that are extremely powerful in specialized situations. This section discusses these products and includes the following topics:

- Products that supply additional primitives.
- The ICAD Surface Designer.
- The ICAD Solids Designer.

Products that Supply Additional Primitives

Some applications require that you use geometric parts that are more powerful than those supplied by the basic system. ICAD offers the following component products that add more powerful parts to the basic ICAD system:

- The ICAD Surface Designer provides high-level curve and surface parts.
- The ICAD Solids Designer provides parts for building solid models.

The ICAD Surface Designer

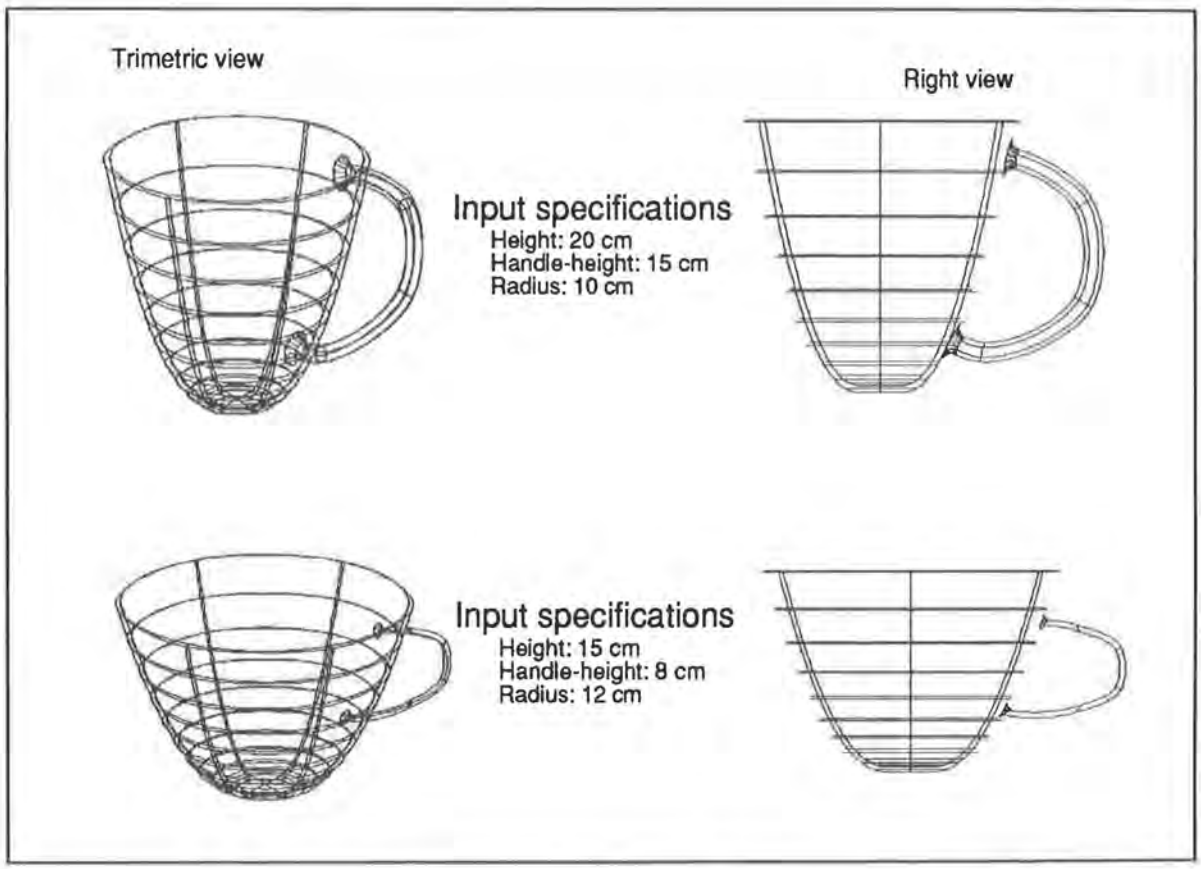
The ICAD Surface Designer integrates parametric curve- and surface-modeling techniques with a knowledge-based engineering system. It allows you to describe the "free-form" surface geometry of many products such as airplane wings, ship hulls, vehicles and complex molded objects.

An ICAD Surface Designer part behaves like any other defpart in The ICAD System; it defines inputs and engineering rules, and you can mix it into your own defparts. The surface geometry is recalculated automatically when you change the inputs to the part.

The ICAD Surface Designer includes a rich set of NURB (Non-Uniform Rational B-spline) curves and surfaces. It includes a large number of high-level primitives such as swept surfaces, offset surfaces, surfaces of revolution, extruded surfaces, arbitrarily trimmed surfaces, and various types of filleted and blended surfaces.

Additional ICAD-supplied Geometric Parts

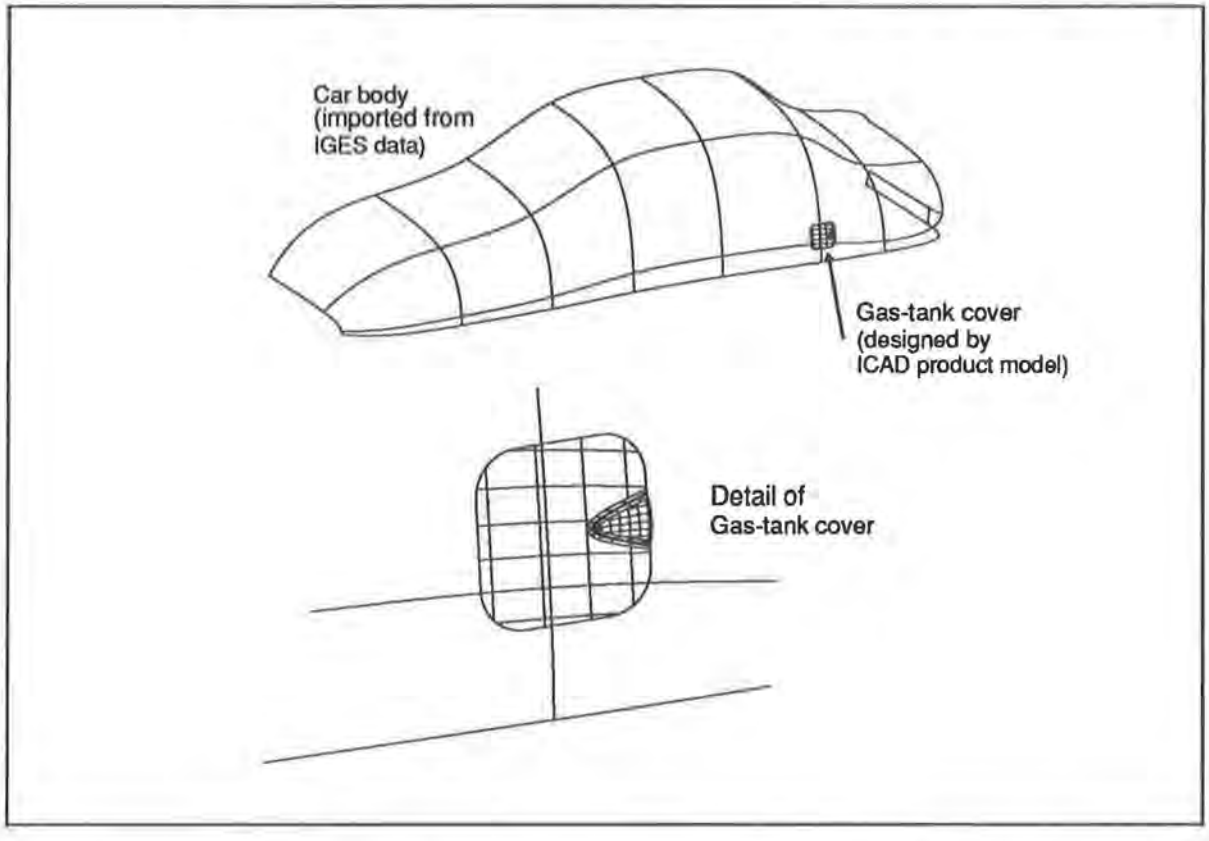
A simple product model of a cup, for example, uses the ICAD Surface Designer to construct the surfaces that make up the cup. The body of the cup is constructed from a revolved surface. The cup handle is a swept surface. Fillets generate a smooth blend between the handle and the body. Notice that simply by changing the input specifications the entire surface geometry is automatically recalculated.



Trimetric and right view of two different design instances of a cup. As the input specifications change, the surface definitions (fillets, sweeps, revolutions) are automatically updated.

Additional ICAD-supplied Geometric Parts

The ICAD Surface Designer often uses surfaces imported from other surface packages, typically a CAD system. For example, a gas-tank cover product model imports a set of surfaces generated by a CAD system that models an automobile body. The ICAD Surface Designer is used to design a gas-tank cover for this existing car body. The designer specifies where on the auto body the gas-tank cover should be placed, and the ICAD Surface Designer automatically calculates the surface geometry of the gas-tank cover based on its location.



View of a gas-tank cover that is automatically generated by an ICAD product model. The geometry for the automobile body is imported via IGES.

ICAD Solids Designer

The ICAD Solids Designer integrates the ruled-based techniques of a knowledge-based engineering system with a solid modeling system. With the ICAD Solids Designer you can create and view solid objects that are built from a library of ICAD-supplied parts, such as basic 2D and 3D primitive shapes, swept solids, and Boolean unions, intersections, and differences (that is, subtraction).

Additional ICAD-supplied Geometric Parts

A solid model contains much more information than a wireframe model. It includes topological information that represents the connections between faces, edges, and volumes of the solid. This allows The ICAD System to display a view-dependent picture of the part with all hidden lines removed (that is, the lines that are obscured by other parts are not displayed). You can also make section cuts that give detailed auxiliary views of a part and create exploded views to show entire assemblies.

A solid model also can calculate volumetric and mass properties such as moments of inertia. In addition, you can check for interference between two parts or part assemblies.

These properties of solid models are useful for manufacturing. A solid model can represent features such as holes, slots, chamfers, etc., which include manufacturing attributes such as surface finish, hardness, and tolerances. These features can be used by the design engineer to create the product design, and the information carried by them can be passed directly into manufacturing to create process plans. The manufacturing systems can use the feature-interference capabilities of the solid model to help determine the best sequence of operations in a process plan.

The Manufacturing-view of the mold base uses a solid model of the mold-base components as a tool for manufacturing. View-dependent images of the mold base with hidden lines removed can be displayed which are easier to interpret than wireframe images. In addition, many components are represented as features which contain manufacturing information. The Top-clamping-plate, for instance, contains Counterbore and Straightbore features with information that can be passed directly to manufacturing to generate a process plan.

Summary

- ICAD provides various additional geometric primitives that meet the needs of special design applications.
- The ICAD Surface Designer is a powerful surface-modeling package that integrates knowledge-based engineering with parametric curves and surfaces. Its surface modeling capabilities surpass those of most CAD systems.
- The ICAD Solids Designer is a powerful solid-modeling package, based on the Parasolid solid modeling kernel from Shape Data, Ltd., that integrates knowledge-based engineering with solid modeling.

Developing Complex ICAD Product Models

Keywords: *edit/compile/test loop, editing, compiling, testing, demand driven, ICAD Browser*

Overview

This section highlights some of the features of The ICAD System that facilitate the development of large-scale, complex ICAD product models and includes the following topics:

- The complexity of real-world applications.
- ICAD Design Language features for developing complex ICAD product models.
- How ICAD's application development environment, including the ICAD Browser, helps you develop complex ICAD product models.

Complex ICAD Product Models

Real-world applications are extremely complex and can encompass thousands of different component parts with tens of thousands of engineering rules. The ICAD System is designed to scale up to large and complex problems without becoming unwieldy. There are features both in the ICAD Design Language and the ICAD Browser that give you access to the complete ICAD product model as needed while you concentrate on a tiny subset of it. The ICAD System is designed to take advantage of the fast processors, high-resolution displays, and large-capacity disks of today's powerful engineering workstations.

Features of the ICAD Design Language

The ICAD Design Language is specifically designed for building large-scale, complex ICAD product models. Using its object-oriented language features, even the most complicated product designs can be developed in a modular, easy-to-understand, maintainable fashion. These language features include:

- *Object-oriented definitions*

  Components of the product are defined as separate objects (defparts), which allows you to think about the overall product in terms of relationships between separate parts.

Developing Complex ICAD Product Models

- *Product structure tree*

  The product structure tree is similar to how an engineer thinks about the product structure of a mechanical assembly. Each level of the product structure tree can be developed separately, which makes it easy to allocate different components of the project among several members of the design team.

- *Mixins*

  Mixins allow you to build complex definitions by "mixing" together simpler definitions. Each definition is clear and easy to understand, which makes the overall product definition easy to understand and maintain.

- *Symbolic referencing*

  Symbolic referencing allows you to define complex relationships between different components in the product structure tree. Attributes in each component can be referenced by name and location in the tree, instead of using a complex addressing system.

- *Part-whole inheritance*

  Part-whole inheritance allows the values of engineering attributes in the subparts to be inherited from attributes with the same name higher up in the product structure tree. This makes it easy to define engineering values and make them accessible throughout the entire product definition.

- *Geometric parts*

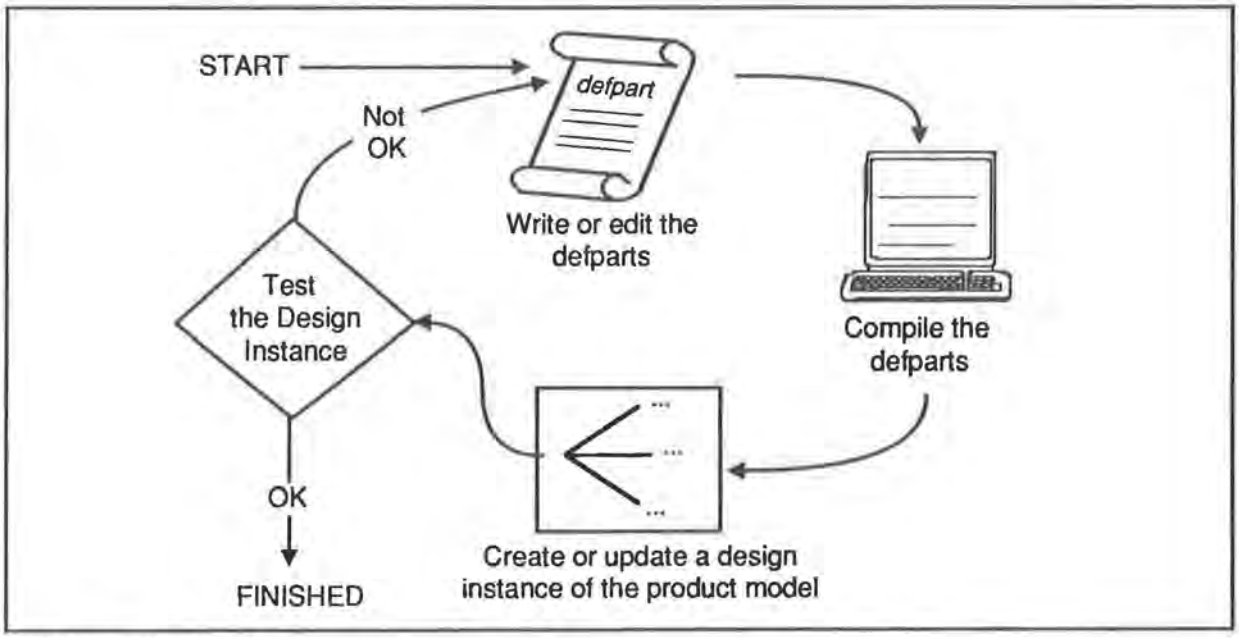
  The rich set of geometric primitives provided by The ICAD System allows you to model all kinds of mechanical parts. The ICAD Surface Designer and the ICAD Solids Designer offer even more kinds of geometric parts and operations.

Developing Complex ICAD Product Models

ICAD Development Environment

The ICAD development environment has two major purposes:

- To cut down on the time involved in the *edit/compile/test* loop, which is a cycle common to most software development efforts. That is, the designer *edits*, i.e., writes or makes a change to a defpart, *compiles*, i.e., uses the compiler to translate the defpart into a form the computer can understand, and *tests*, i.e., tests the design instances created from the defpart for errors. The cycle repeats until no errors are found.



Edit / compile / test loop

- To facilitate the inspection and testing of complex design instances generated by an ICAD product model. The ICAD product model takes an input specification, applies the engineering rules, and generates a design instance. ICAD provides special tools for "browsing" the design instance.

The ICAD rule-development environment includes the following features:

- All platforms include a powerful, full-screen, customizable editor for writing defparts.

- ICAD provides an *incremental compiler* for compiling defparts. Often a defpart needs to be recompiled after a small change. Incremental compilation means

that The ICAD System only compiles the change, not the rest of the ICAD product model. This saves a significant amount of time in the development cycle.

- ICAD provides the ICAD Browser which is a powerful graphical-debugging environment. The ICAD Browser is where you create a new design instance, supply it with the top-level inputs, demand values from the engineering rules, and generate output. The ICAD Browser includes numerous debugging tools, including:

  - An Inspector Viewport that allows you to see the value of all attributes for any part in the part design.

  - A Graphics Viewport that allows you to draw the geometry for a part, the children of a part, or the leaves of a part. The ICAD Browser provides graphical picking and graphical verification within a Graphics Viewport, e.g., when viewing surface parts you can choose to view the Gaussian curvature, the surface normals, etc.

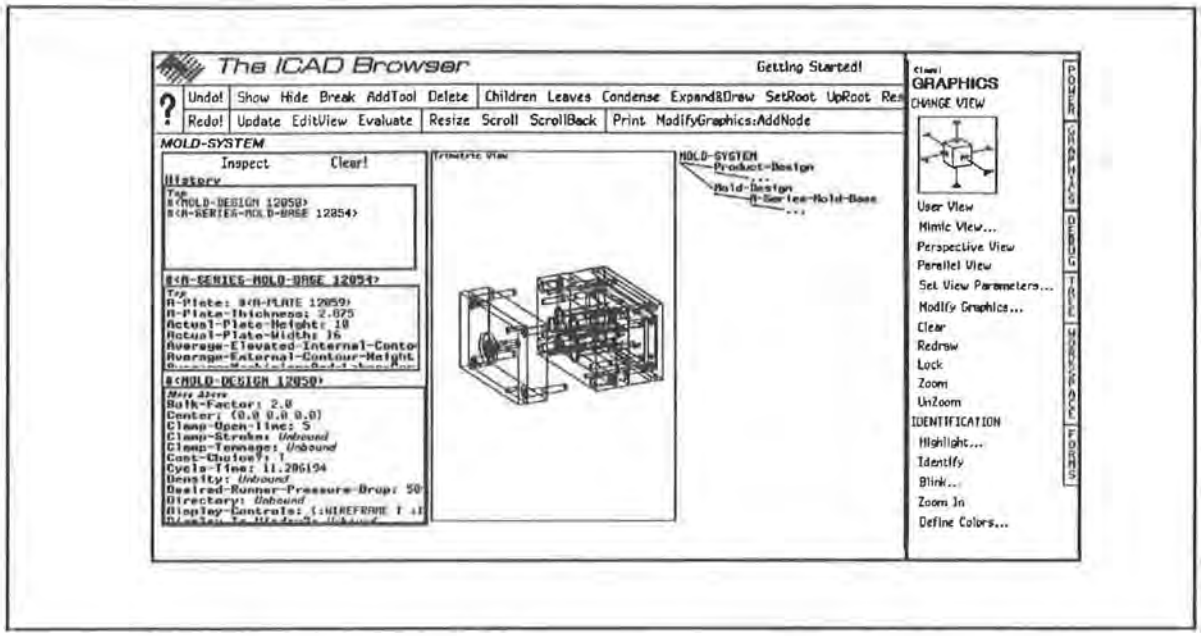
  - An Expand command that allows you to create only as much of the design instance as necessary. Parts of the design on which you are not currently working need not be created. This saves computing space and time.

  - A Set Root command that isolates the part of the product structure tree in which you are interested. This is necessary since a real product structure tree is often so large that it does not on the computer screen.

  - An Update command for updating a part design so that it incorporates any changes that you have made in the defpart. Often while you are working on a design instance in the ICAD Browser you need to make a change in one of the defparts that describes it. The Update command allows you to incorporate these changes into the existing design instance. You do not have to create a new design instance for the changed defpart.

- The ICAD System is *demand driven*. This means that only those attributes, parts, etc. that are required are calculated. This is tremendously important, since you often work on one small piece of the potentially huge ICAD product model at a time.

Developing Complex ICAD Product Models



The ICAD Browser

Summary

- The ICAD System manages extremely complex ICAD product models.
- The ICAD Design Language is designed for developing complex ICAD product models.
- ICAD provides a graphical development environment to aid in all phases of development and to view design instances generated by complex ICAD product models. This enviroment includes the ICAD Browser.

End-user Interfaces for an ICAD Product Model

Keywords: production user interfaces

Overview

This section shows how you deploy a finished ICAD product model for production purposes and includes the following topics:

- Designing an ICAD product model for production.
- The ICAD Production System
- Simple interfaces to a finished ICAD product model.

Designing an ICAD Product Model for Production Purposes

After completion, ICAD product models are often stand-alone engineering systems. End users (e.g., design engineers, project engineers, manufacturing engineers) know little or nothing about The ICAD System; they are only interested in the automated engineering solution that the ICAD product model provides. For this reason, it is important to develop an interface to the ICAD product model that includes an easy way to provide inputs and generate outputs.

For the rest of this document, we differentiate between the developer of an ICAD product model and the end user of the ICAD product model. The second person pronoun ("you") always refers to the developer of the ICAD product model. We refer to the end user of the ICAD product model in the third person as "the design engineer" or "the end user".

ICAD Production System

The ICAD Production System is a low-cost version of The ICAD System that is targeted for production use. You can use the ICAD Production System to generate output from finished ICAD product models but not to develop new ICAD product models.

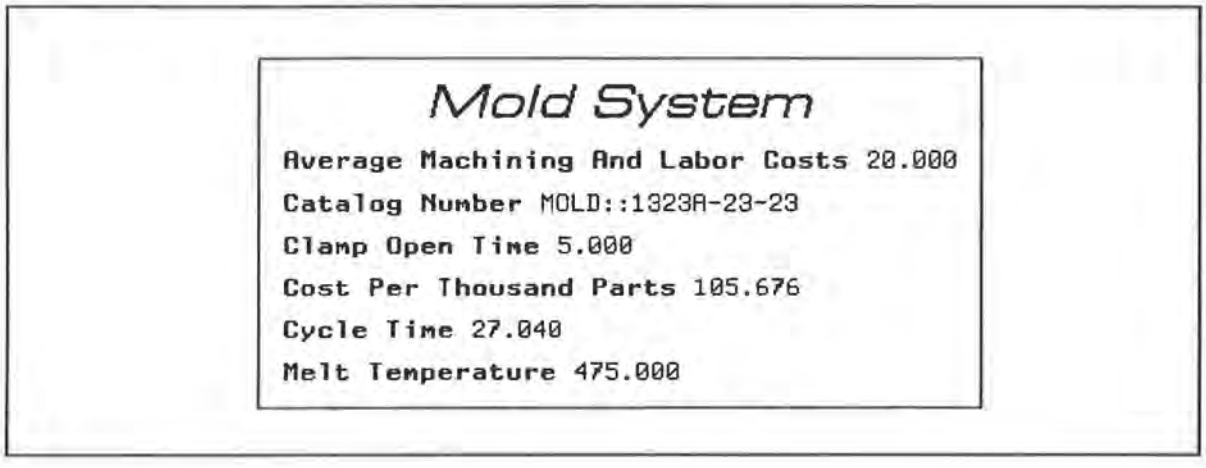
ICAD Browser Interfaces to the ICAD Product Model

ICAD provides the ICAD Browser as a graphical interface to an ICAD product model. Many simple interfaces are included in the ICAD Browser:

- The Top-level Inputs Viewport allows an end user to enter input specifications for the ICAD product model.

End-user Interfaces for an ICAD Product Model

- Choice attributes allow the end user to specify input values interactively via popup windows.
- Engineering forms that display the values of important engineering attributes can be created in the ICAD Browser.



Sample engineering form.

- Reports can be generated from the ICAD Browser using a single mouse click.

ICAD Production UI

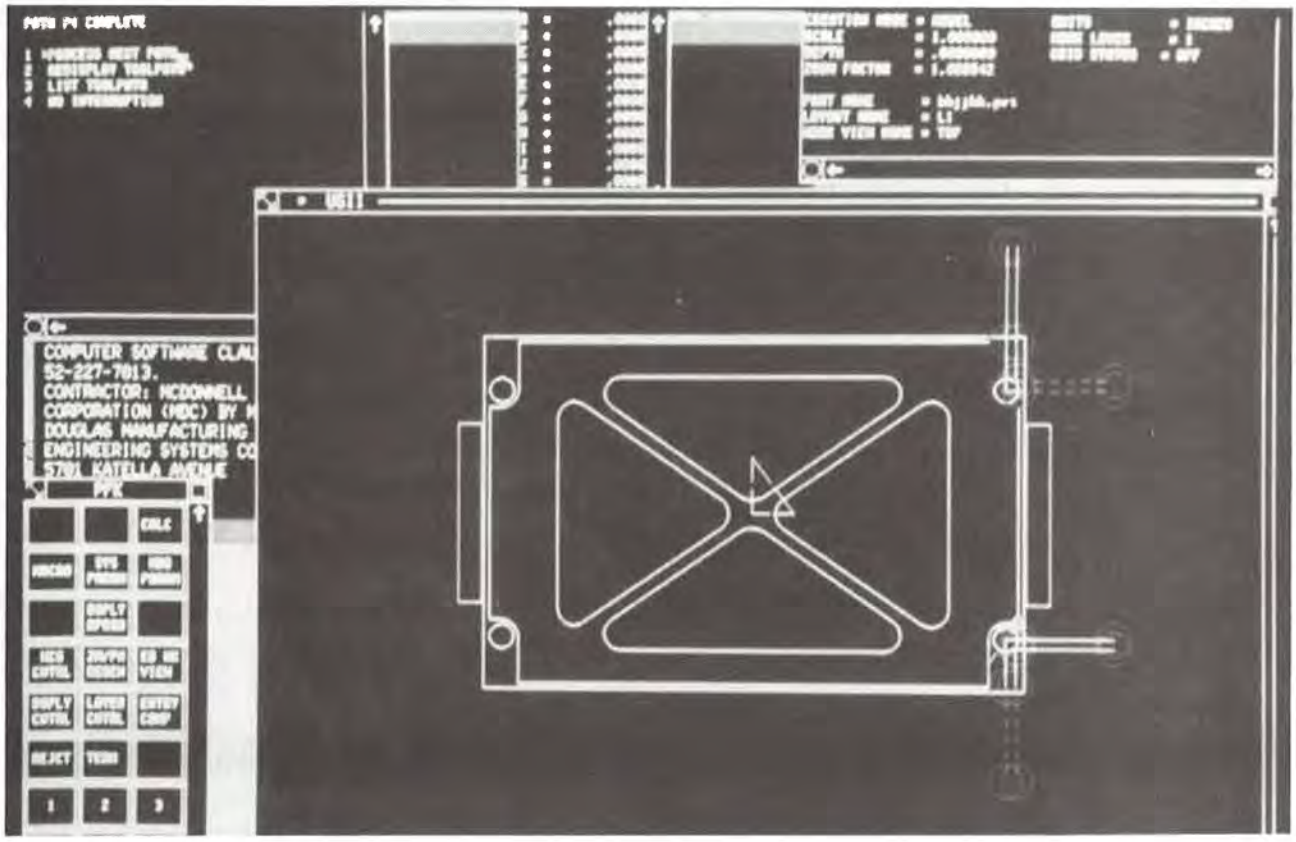
The ICAD Production UI (User Interface) allows you to develop a menu-based end-user interface in which an engineer can edit input values used to generate the design instance.[16] It includes additions to the ICAD Design Language that let you specify menus for selecting which inputs to edit and specification sheets for presenting and interacting with the input values.

[16]The ICAD Production UI is a new component product that will be released in 1990.

End-user Interfaces for an ICAD Product Model

Other Production User Interfaces

McDonnell Douglas provides the UG-ICAD Connection product, which is a simple production user interface to an ICAD product model through the Unigraphics II System.



End-user Interfaces for an ICAD Product Model

You can develop your own simple production user interface using the ICAD Kernel Interface, which lets you access the capabilities of The ICAD System without using ICAD's own user interface.[17]

Summary

- A successful ICAD product model requires an end-user interface.
- ICAD offers the ICAD Production System as a low-cost solution for use with an end-user interface.
- ICAD provides an interface to an ICAD product model through the ICAD Browser and the ICAD Production UI. Alternately, you can develop your own end-user interface using the tools provided by the ICAD Kernel Interface.

[17]The ICAD Kernel Interface is a new component product that will be released in 1990.

Summary of Chapter 4

- ICAD consultants help you plan your first application by showing you how to divide your application into smaller components, identify the inputs to and outputs from your application, and develop rules for each component.
- An ICAD product model is divided into components, each of which is described by a defpart. Each defpart is compiled before it is used to generate a design instance of the defpart. Defparts include attributes, which are its engineering rules, inputs, which are the input requirements for a new design instance, and parts, which are its components. You supply the inputs when a new design instance is generated to create the design instance.
- Design instances are organized into a product structure tree, at the root of which is a design instance of the overall product. The product structure tree decomposes a product such as a crankshaft, mold base, etc., into assemblies, subassemblies, component parts, features, and primitives. Other components in the product structure tree can represent processes used to manufacture or test the product. A defpart contains a parts section which describes how the defpart decomposes into components.
- The ICAD Design Language provides techniques for describing relationships between different parts of the product structure tree. An engineering attribute in one part of the product structure tree can be written that depends on an attribute in another part of the product structure tree. An attribute value in a part can be inherited by the descendants of the part.
- The ICAD Design Language provides techniques for modularizing an ICAD product model. These techniques include dividing the assembly into components, using generic parts, and using mixins.
- Geometric modeling is accomplished using ICAD-defined primitive parts and positioning and orientation features of the ICAD Design Language. ICAD provides a large set of primitive parts. Positioning and orientation is done relative to the subassemblies, so that entire subassemblies can be repositioned as a single entity.
- The ICAD System can manage extremely complex ICAD product models. ICAD provides tools for writing, compiling, debugging, testing, and using an ICAD product model, as well as for developing end-user interfaces.

Chapter 5

Integrating ICAD Into an Existing Engineering Environment

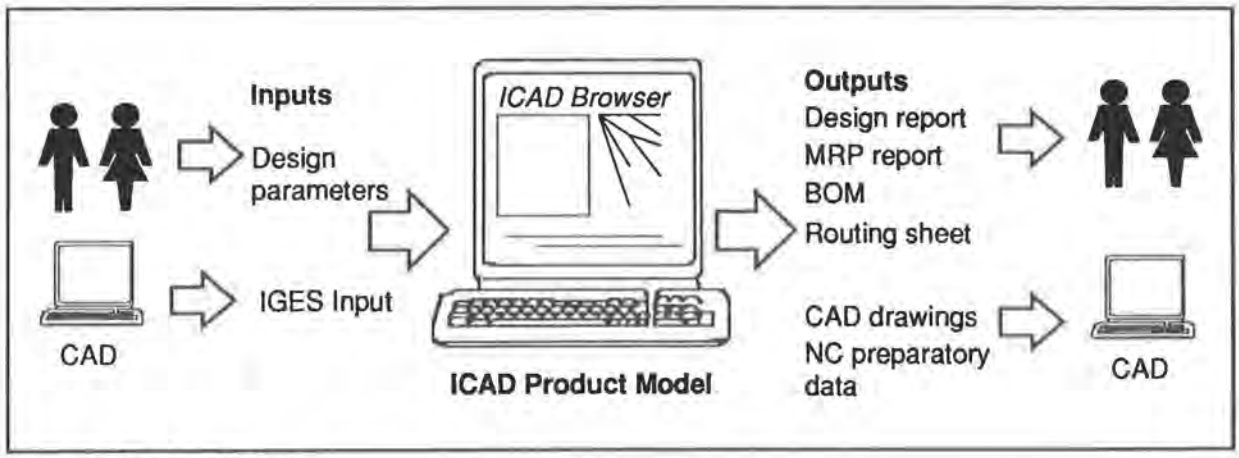
Introduction

The Design Pipeline

Most successful ICAD product models are integrated into a larger engineering environment that includes engineers in different departments, traditional CAD systems, analysis programs, corporate databases, and a corporate structure that demands accountability and justification.

An ICAD product model is typically part of a *design pipeline* that includes other engineering systems. Inputs enter the pipeline from engineers or directly from other engineering systems (e.g., CAD systems). As we saw in the previous chapter, the ICAD product model generates a design instance that meets the input specifications and generates outputs such as drawings, reports, and data for other computer systems.

The following diagram shows a simple design pipeline with inputs and outputs.

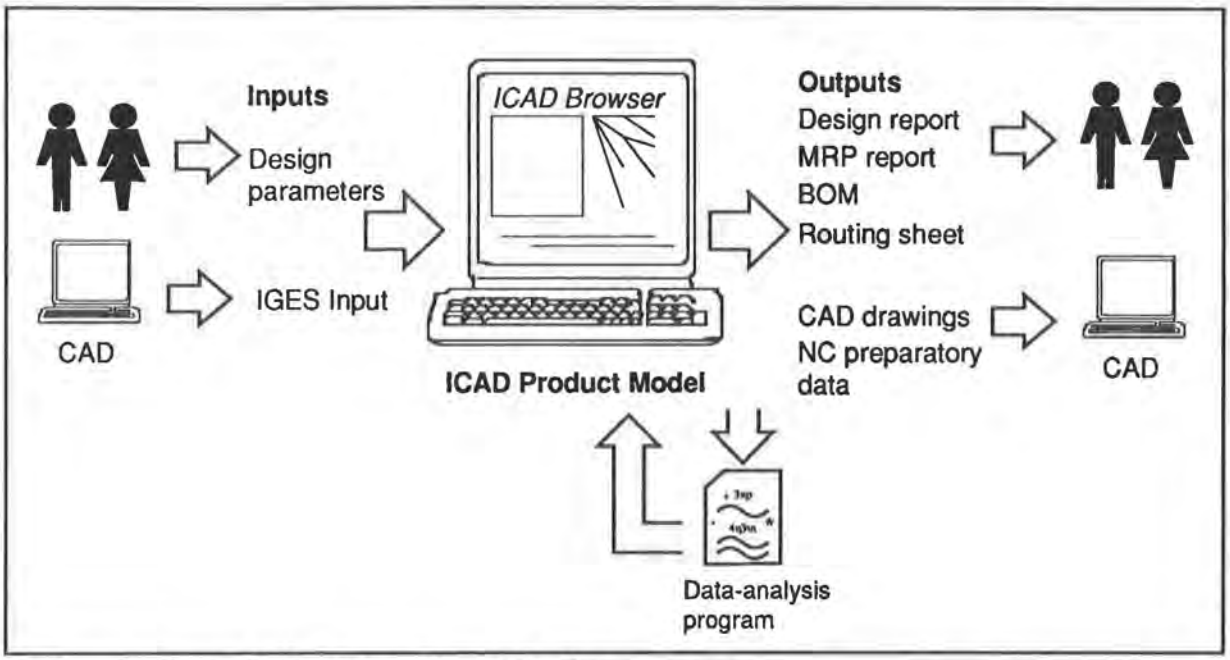


Simple design pipeline.

A more complex design pipeline might include an analysis package. The product model contains rules that generate data for the analysis package and send the data to the package for evaluation. In some applications, the product model also contains rules that use the result of the analysis package to reconfigure the design.

Introduction

The following diagram adds an analysis package to the design pipeline.

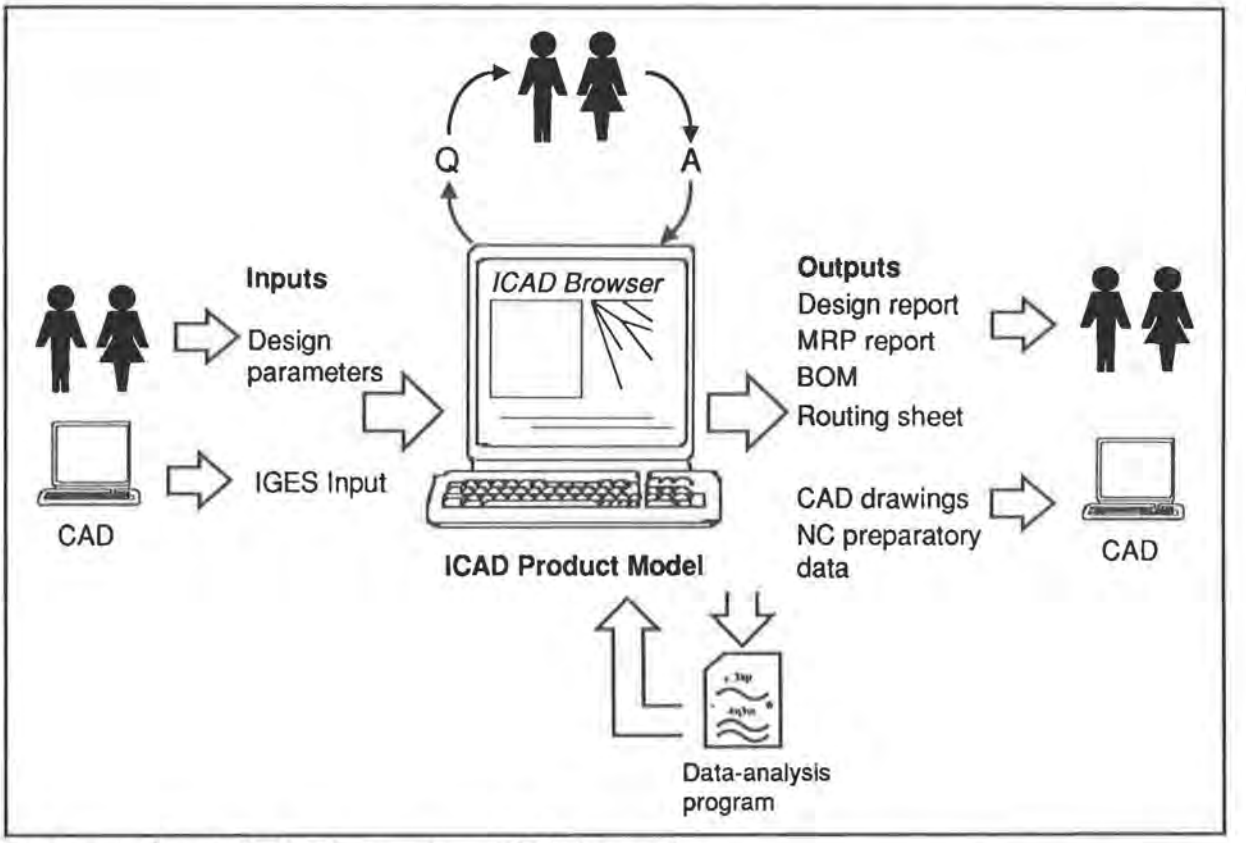


Design pipeline including interaction with other programs.

Human interaction is often part of the design pipeline. As The ICAD System evaluates rules in the design instance, it may prompt the end user for additional information.

Introduction

The following diagram adds human interaction to the design pipeline.

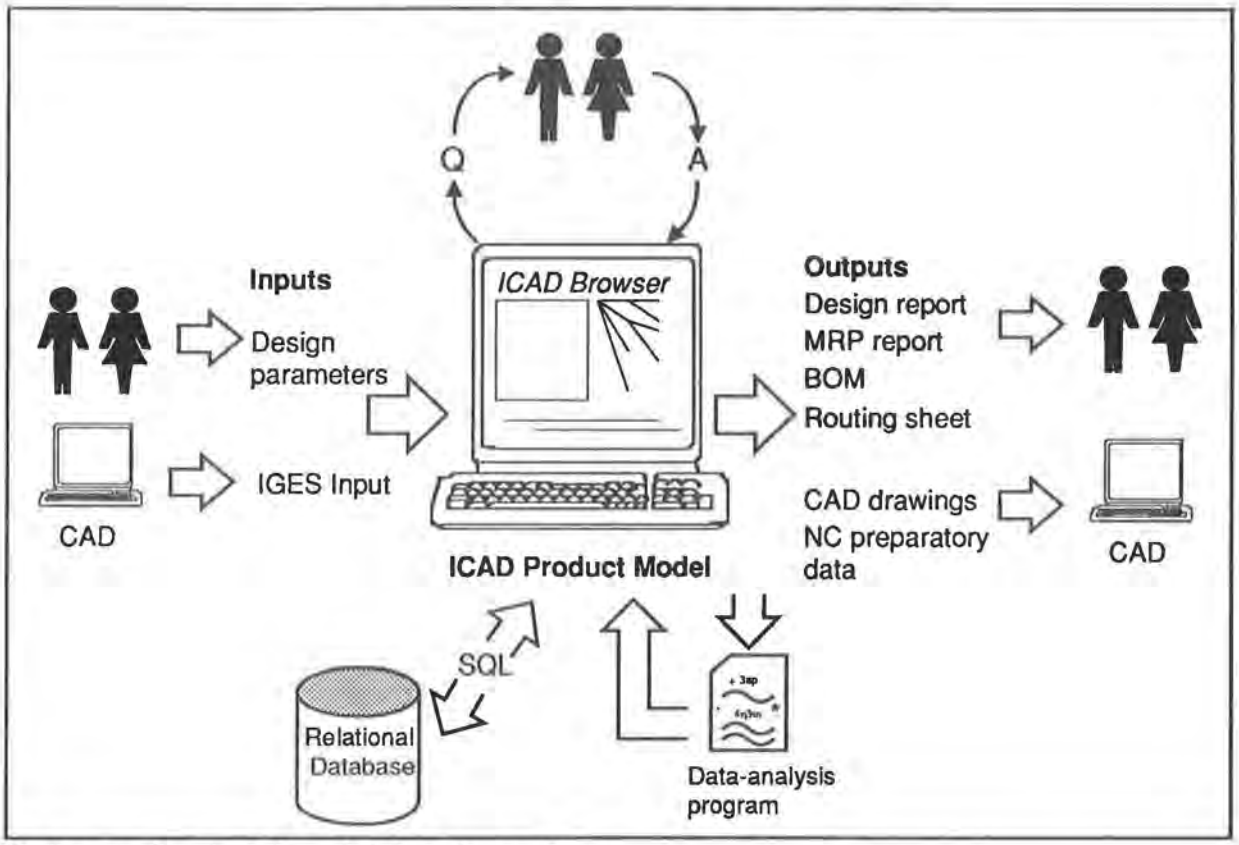


Design pipeline including interaction with design engineer.

Databases that contain part information, material data, MRP data, etc., can also be accessed by the design instance. Some applications write to databases as well.

Introduction

The following diagram adds a database to the design pipeline.

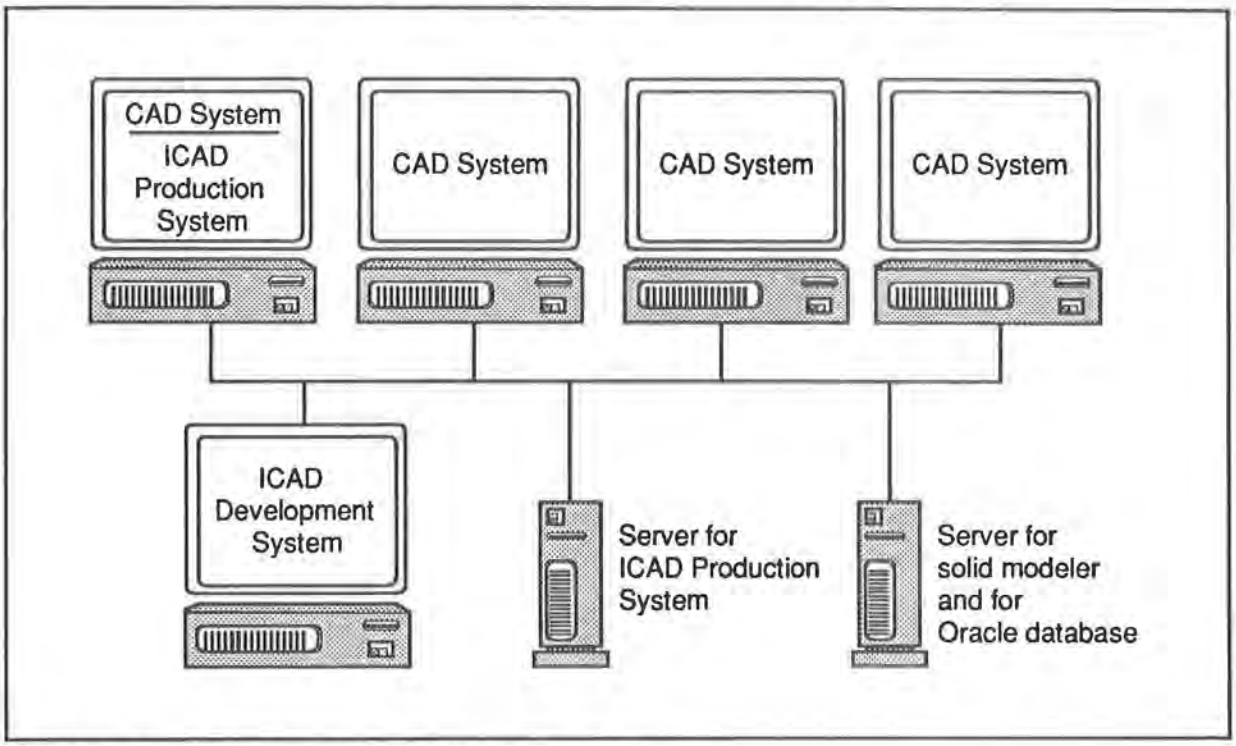


Design pipeline including database transactions.

The complete design pipeline can be implemented on a distributed network that includes different hardware platforms. Different elements of the pipeline can be on different platforms or in some cases on the same platform.

In one possible network configuration, a UNIX platform runs a CAD workstation and an ICAD development system on which a new ICAD product model is being developed. The network contains four other CAD workstations. Three of these access an ICAD production system on a UNIX workstation via a client/server configuration. The fourth has an ICAD production system resident on the same workstation. The ICAD Systems on the network are clients for a solid-modeling server and an Oracle-database server running on another UNIX workstation.

The following diagram shows this network configuration.



Possible network configuration for the design pipeline.

Design Pipeline for the Mold Base

The mold-base application is part of such a design pipeline. The ICAD System is the central part of the pipeline; not only does it receive inputs and generate outputs, but it is also a central repository of engineering data about mold-base design. The design pipeline works as follows:

- Initially, a part designer designs the part to be molded on a McDonnell Douglas UNIGRAPHICS II CAD System, which generates an IGES representation of the geometry. The ICAD product model reads the IGES representation and translates it into an ICAD representation, that is, as design instances of geometric parts.
- The ICAD product model designs an appropriately-shaped part insert.
- Once the insert is designed, the mold-base designer uses a custom-designed user-inteface (designed using the ICAD Production UI) for feature recognition in the part design.

Introduction

- The ICAD product model prompts the mold-base designer for additional information such as runner pressure drop and material temperature, and configures an optimal mold base for the part.

- The ICAD product model includes an SQL interface to an Oracle database from which it gets vendor-supplied catalog data about component parts in the mold base.

- Once a configured mold base is designed, it is sent to a flow-analysis package for analysis. The mold-base designer views the output from the flow-analysis package and uses it to make minor modifications to the mold-base design, such as repositioning the gates or changing the material temperature. The ICAD product model redesigns the mold base, and it is sent back to the flow-analysis package for re-analysis. This step is repeated until the mold-base designer is satisfied with the output from the flow-analysis package. This same process occurs with a cooling-analysis package.

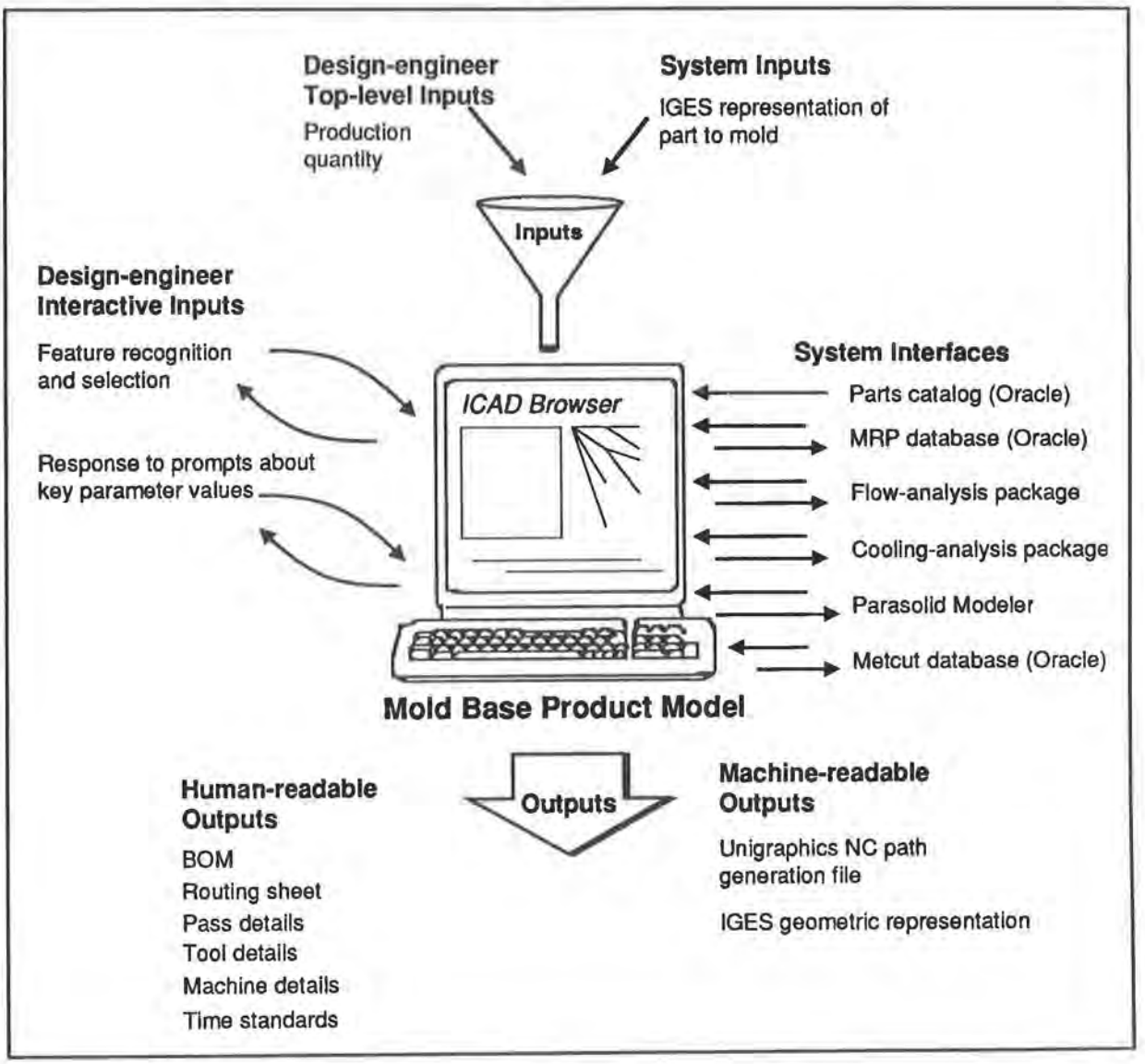
- Once a validated mold base and configuration are designed, the ICAD product model queries an Oracle database to see if an equivalent mold base has already been manufactured. If so, the existing mold base is used. If not, a description of the new mold base is registered in the Oracle data base.

- If an existing mold base is found, the existing outputs for that mold base are located. If no equivalent mold base exists, the ICAD product model creates a solid model of the mold base using the ICAD Solids Designer, and generates a process plan for milling the A- and B-plates. An interface to the Metcut database returns the appropriate machining operations data.

- The ICAD product model generates various reports, including a bill of materials, a routing sheet, and tool details. Machine-readable outputs include an IGES representation of the completed mold base, and a UNIGRAPHICS NC prep generation file.

Introduction

The following diagram shows how the complete mold-base application is integrated into the design pipeline:



Overview of this Chapter

The rest of this chapter describes aspects of The ICAD System that can integrate your ICAD product model into a similar engineering pipeline.

"Inputs to The ICAD System"
Describes various ways in which inputs can enter into the

Introduction

ICAD product model from both the design engineer and from outside systems.

"Outputs from The ICAD System"
Describes various ways that the product model generates outputs to the design engineer and to other systems.

"Integration Tools and Strategies"
Describes ICAD Design Language tools for designing an integrated ICAD product model.

Inputs to The ICAD System

Keywords: *design-engineer inputs, system inputs, catalog inputs, top-level input specifications, interactive input specifications, specification sheets, catalogs, ICAD Relational Object Manager, ICAD Oracle Interface, ICAD Production UI, ICAD Kernel Interface*

Overview

The ICAD System supports different kinds of inputs to an ICAD product model. The different kinds of inputs can be broadly divided into three basic categories. Each category is discussed later in this section.

- Inputs supplied by the design engineer.
- Inputs generated automatically by another computer system and read by the ICAD product model.
- Inputs from databases that can be queried by the product model.

Inputs Supplied by the Design Engineer

The design engineer can supply input in two ways: via top-level inputs and via interactive inputs.

Top-level inputs, or initial input specifications, are supplied in order to create a new design. Top-level inputs are implemented in the ICAD Design Language using input-attributes.

In the mold-base application, the only top-level input to the mold base product model is the production quantity, that is, the number of plastic parts to mold.

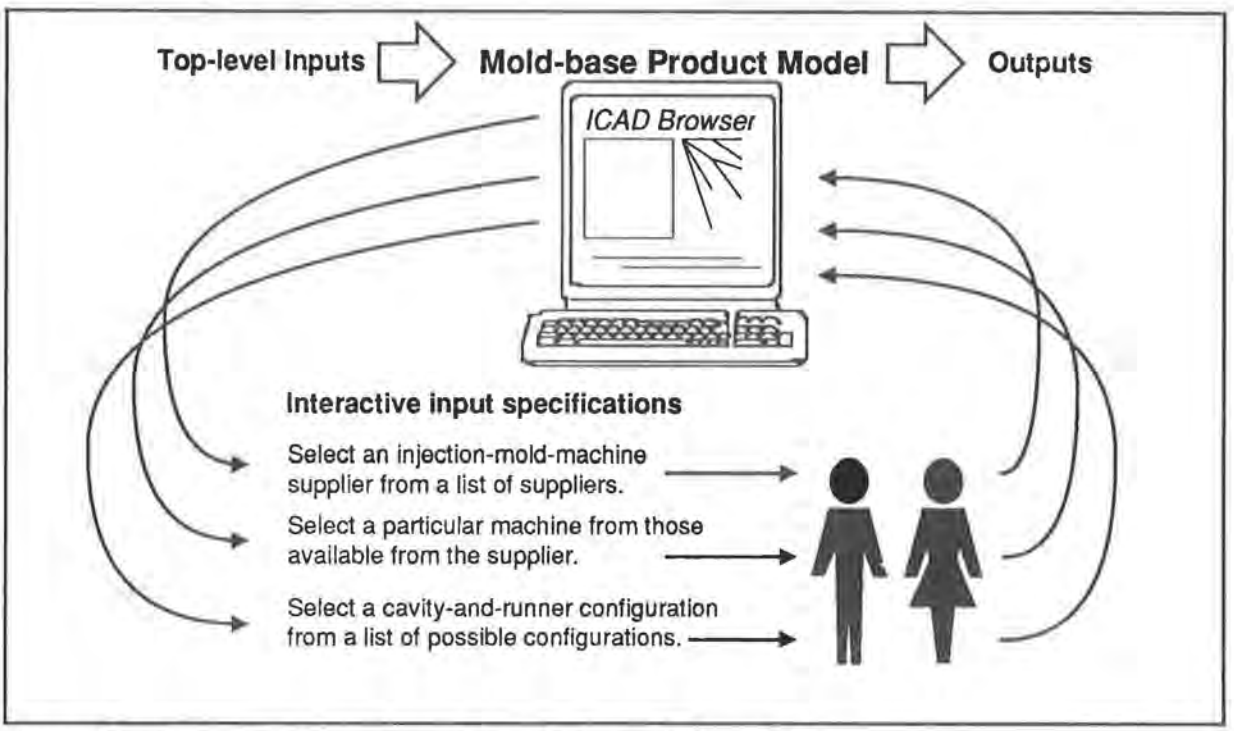
Interactive inputs are supplied by the design engineer when prompted by the ICAD product model. While The ICAD System is generating a new design instance, it may require additional information from the design engineer. Therefore, the actual interactive inputs for which the design engineer is prompted depends on the processing that has already occurred. Interactive inputs are implemented in the ICAD Design Language using choice-attributes or using the ICAD Production UI.

The mold-base design pipeline includes many different interactive inputs. In one case, it prompts the mold-base designer for the name of an injection molding machine supplier from a set of available suppliers. Depending on the selected supplier, it then prompts the designer to choose a particular injection molding machine from a set of mold bases provided by that supplier. In another case, rules in the Cavity-and-runner-configuration part determine the maximum number of cavities that can be used with that machine, configure the cavities

Inputs to The ICAD System

for each possible number of cavities, and calculate the cost per part and production time for each configuration. The designer is then asked to choose the appropriate cavity configuration based on the calculated data. Sometimes the mold designer chooses to optimize for cost, other times to minimize manufacturing-production time.

The following diagram shows some of the interactive inputs in the mold-base application.



Some of the interactive inputs for a design instance of the mold base.

ICAD provides the ICAD Production UI for customizing interactive input in special-purpose specification sheets that are organized into hierarchical menus. By accessing the appropriate specification sheet, a design engineer can select graphical entities, enter textual values, or choose from a set of items. You design the appearance of the specification sheet as part of developing the ICAD product model.

The process of extracting features from the mold-base design is well suited to an interactive spec sheet. The designer interactively indicates the parts of the mold base that are features, and then specifies what kind of features they are, e.g., a boss, a pocket, etc.

Inputs to The ICAD System

System Inputs

System inputs are generated by some other computer system and are brought into the product model automatically. They are integrated into the ICAD product model by rules that read and interpret the data from the external system. Often (although not necessarily) the data is in a computer file that can be accessed by the computer on which The ICAD System runs.

The ICAD System supports the direct import of IGES files. In the mold-base application, the geometry of the part to be molded is brought into the ICAD product model as IGES data.

Alternately, you can write special rules to interpret any other data format. For example, many aerospace companies have geometric data stored in proprietary file formats. In an ICAD product model that designs hatch covers on airplane wings, the airfoil profile of the wing is stored in a proprietary format. The data is read and interpreted by rules in the ICAD product model and used by the ICAD Surface Designer to construct parts with the same geometry.

Database Interaction

Manufacturing companies often maintain corporate databases to store engineering data. Likewise, vendor catalogs can be maintained in databases, since one of the most common, repetitive, and error-prone engineering tasks is to look up data in catalogs. Such databases can be incorporated into an ICAD product model.

For example, the actual mold-base part in the mold-base product model is a standard part chosen from a catalog of available parts from vendors such as DME or Husky. This catalog is entered as an on-line database from which the product model can select individual items. The ICAD product model chooses the best part from the catalog based on its engineering rules, and can automatically access all the data associated with that catalog entry.

Sometimes it is desirable for the ICAD product model to update an existing database. For instance, after the mold-base product model chooses a mold base and a plan for manufacturing the mold base inserts and runners, it queries a database for the existence of an equivalent mold base design. If none exists, the database is automatically updated to include the new design.

Inputs to The ICAD System

ICAD provides two products that automate database look-up and integrate it into a knowledge-based engineering system.

- The ICAD Relational Object Manager lets you enter catalog information using the same editor with which you write defparts. A powerful set of query operators can then be used to access the data in the catalogs. A typical query might find all of the mold bases that are between two meters and three meters wide, sort them in increasing order by price, and return the first and second one. Another typical query might find the price of an item with a particular catalog entry number, such as A0031.
- The ICAD Oracle Interface is an SQL (Structured Query Language) interface to an Oracle Relational Database Management System, and can easily access and update an Oracle database.

Summary

- The basic kinds of inputs are design-engineer inputs, system inputs, and database inputs.
- There are two kinds of design-engineer inputs: top-level inputs and interactive inputs. Top-level inputs are supplied when the design instance is created — they are inputs to the root of the product structure tree. Interactive inputs are supplied when prompted by the design instance.
- System inputs are inputs that are generated by another computer system and read into the ICAD product model.
- Database inputs come from interactions with a database. Database input is useful for catalog lookup. The ICAD product model can also update existing databases.

Outputs from The ICAD System

Keywords: graphical output, reports, CAD Output, shaded images, bills of materials, report-attributes, translators, CAD Output Tools, ICAD Output Interface Toolkit, ICAD Drawing Tools

Overview

The ICAD System supports three kinds of output. Each is described in this section.

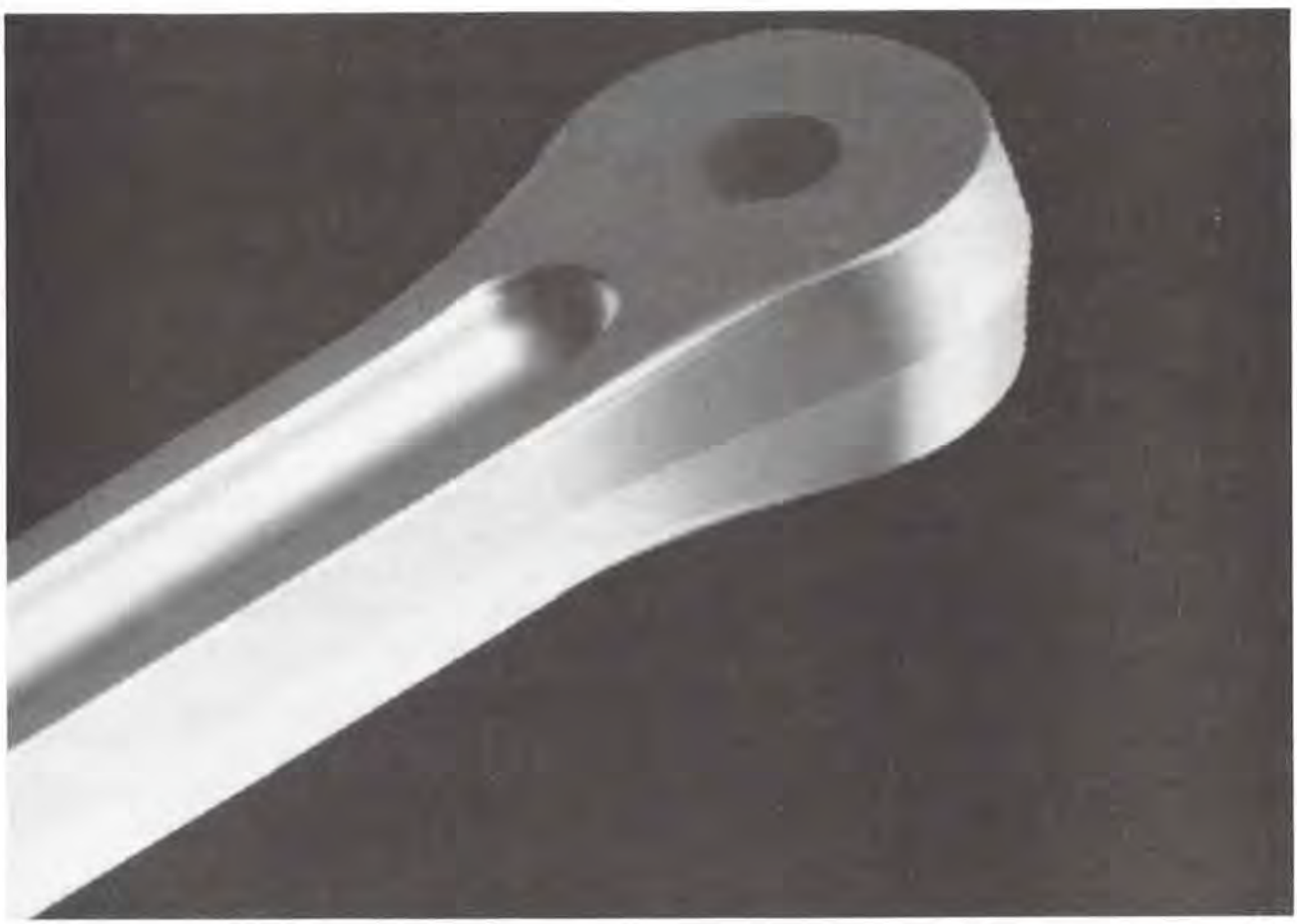
- Graphical output.
- Reports.
- Output to other computer systems, including CAD systems.

Graphical Output

You can display various views of the geometry of the design instance in the ICAD Browser. These views can then be printed on a Postscript printer.

Outputs from The ICAD System

For higher-quality graphical output, an HP 9000 Series 300 SRX or Turbo SRX or a Silicon Graphics IRIS 4D workstation can be networked to the platform running The ICAD System to display full color shaded images. Shaded pictures are especially useful for viewing complex 3D models.



Report Output

Much of the desired output from a design instance is in the form of reports. Anything calculated by the ICAD product model can be output in report form.

For example, the mold-base application includes a number of engineering reports. The main components are a detailed bill of materials and a process plan that includes a routing sheet, pass detail, tool detail, machine detail, and time standards.

Outputs from The ICAD System

The mold-base application automatically generates the following bill of materials:

Bill-of-Materials

DME Catalog Number	Description	Unit Cost	Quantity	Total Cost
1112A-27-27	A-SERIES-MOLD-BASE	$ 1739.00	1	$ 1739.00
B-6605	SPRUE-BUSHING	$ 58.70	1	$ 58.70
5714	SHOULDER-BUSHING	$ 11.40	4	$ 45.60
5108-GL	LEADER-PIN	$ 9.60	4	$ 38.40
EX-7	EJECTOR-PIN	$ 4.15	9	$ 37.35
7517	RETURN-PIN	$ 7.35	4	$ 29.40
6501	LOCATING-RING	$ 16.10	1	$ 16.10
7100	STOP-PIN	$ 0.90	6	$ 5.40
7218	SPRUE-PULLER-PIN	$ 4.60	1	$ 4.60
3/1-4	TUBULAR-DOWEL	$ 2.25	2	$ 4.50
1/2-13	CLAMP-PLATE-SCREWS	$ 0.00	1	$ 0.00
5/16-18	EJECTOR-PLATE-SCREWS	$ 0.00	1	$ 0.00
1/2-13	EJECTOR-HOUSING-SC	$ 0.00	1	$ 0.00

The total cost for the design is $ 1979.05

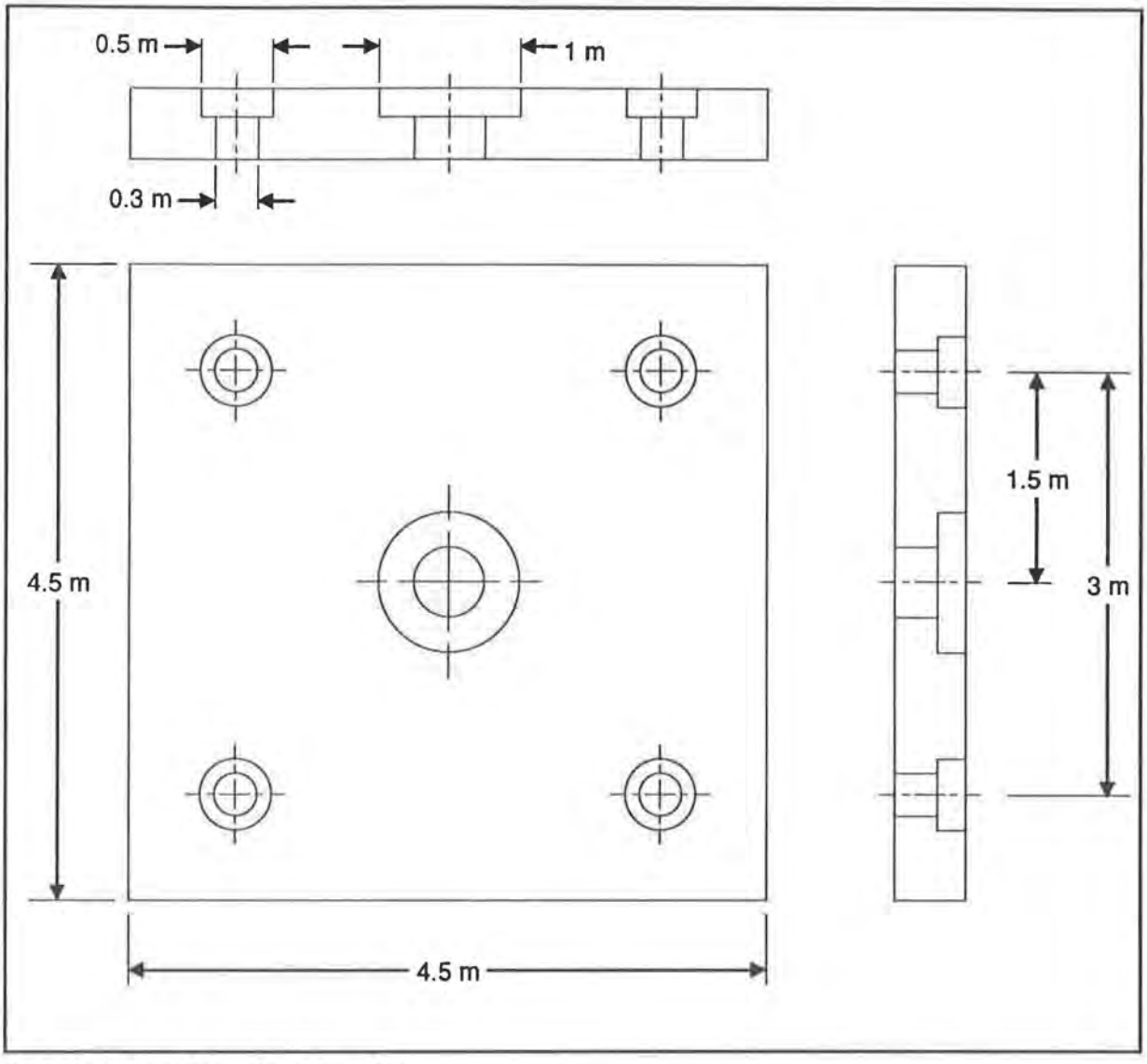
End of Bill-Of-Materials

You include a report definition in a defpart by defining a special attribute called a *report-attribute*. Report-attributes contain rules that collect the desired data for the report and format it appropriately.

CAD Output

Most ICAD applications generate CAD output in the form of 2D and 3D models or detailed, finished drawings. CAD systems usually have a facility whereby a CAD drawing can be read from a file that contains data in a special input format. IGES is often used as a generic input format, although many CAD systems have their own proprietary input formats as well.

Outputs from The ICAD System

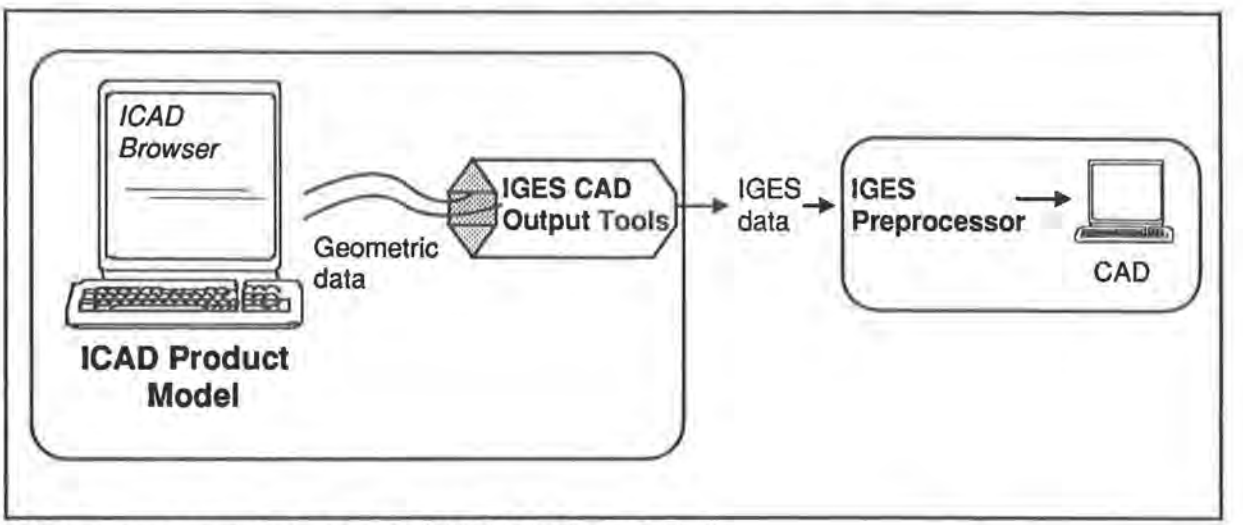


CAD drawing of top clamping plate.

The ICAD System can generate files that contain the geometric data about the ICAD product model in CAD format. ICAD supports the IGES format, as well as direct translators to Intergraph Microstation PC, AutoCAD, CADDS4X, and UNIGRAPHICS II.

Outputs from The ICAD System

The following diagram shows how ICAD's IGES CAD Output Tools are used to generate an IGES data file that can be read by a CAD system.



IGES output generated using the IGES CAD Output Tools.

To generate CAD output for the ICAD product model, you write a report-attribute that collects the geometric data and sends it out in the appropriate format. The following ICAD Design Language fragment defines an engineering report, IGES-Mold-Base, that generates a file in IGES format representing the mold base geometry. The two lines of code are all that is necessary to define the IGES "report." To generate the IGES data, the design engineer "presses" the IGES-Mold-Base report button in the ICAD Browser.

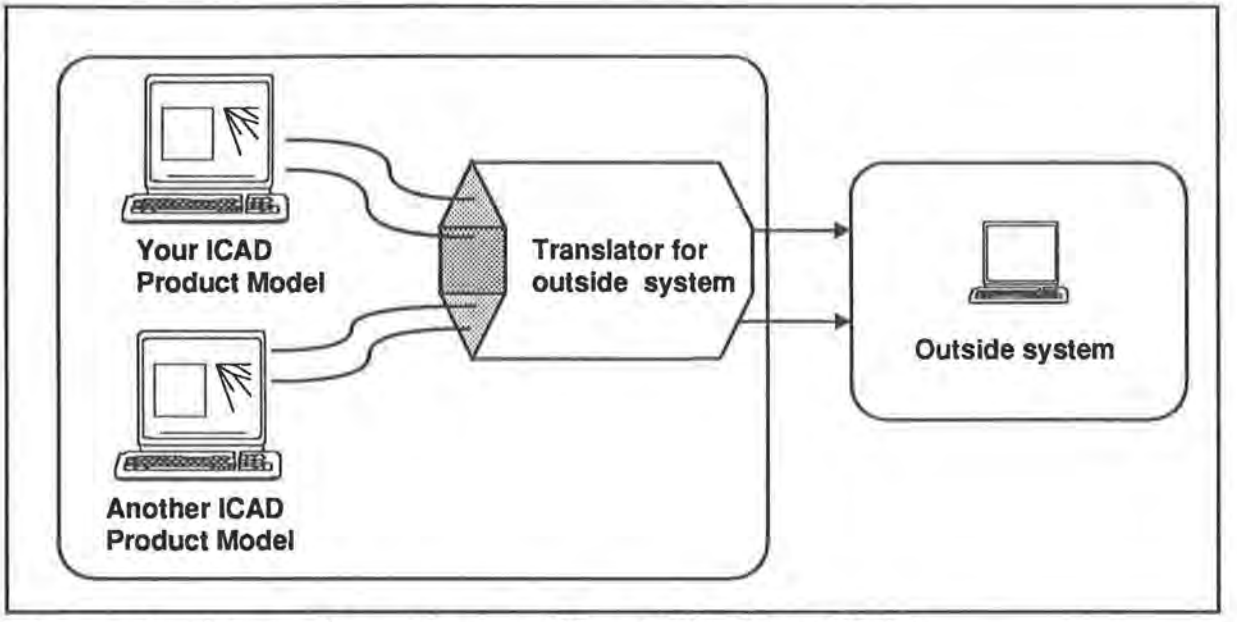
```
:IGES-Mold-base (with-writer (iges "filename")
                    (write-the :cad-output-tree))
```

Output for Other Computer Systems

Most other computer systems, such as analysis packages, NC machining systems, and other CAD systems, accept input data from a file in a custom, often proprietary format. ICAD provides the tools to write your own *translator* from the ICAD product model to the outside computer system of your choice using the ICAD Output Interface Toolkit. The translator collects the necessary data from the ICAD product model and formats it so the other system can understand it. For example, you might write a translator to a finite element analysis package, or to your company's proprietary CAD system.

Outputs from The ICAD System

Once you have written the translator, it can be used for any future ICAD product model as well. This is shown in the following diagram.

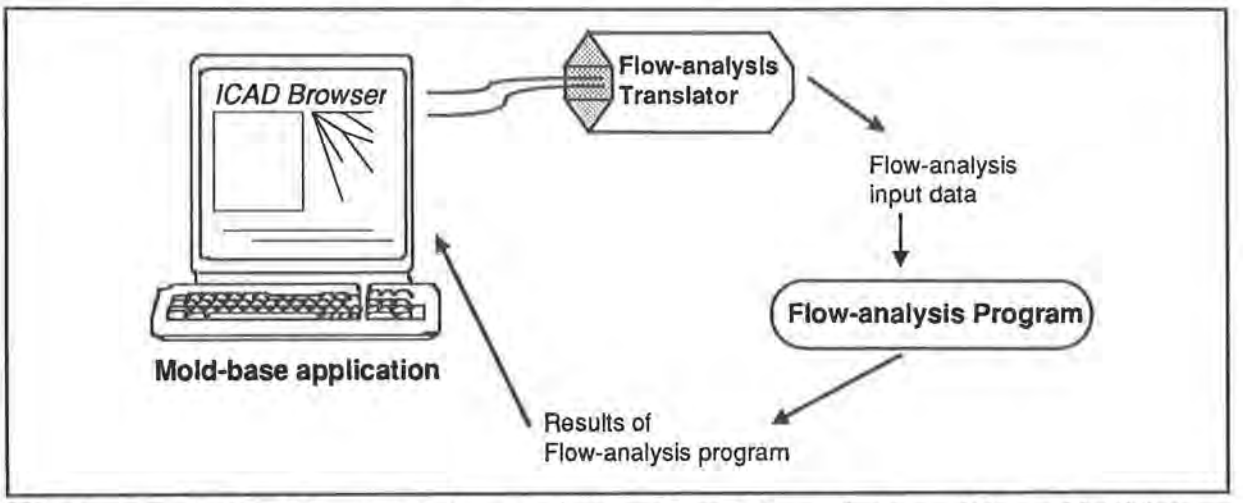


The same translator can be used for many different ICAD Product Models.

In the case of an analysis program, there is often a closed data loop between The ICAD System and the analysis program. The data to be analyzed is sent to the analysis program through the translator. Once analyzed by the analysis program, the ICAD product model can read and interpret the results as a new system input (see the section "Inputs to The ICAD System", page 5-9), using them to reconfigure the design. This loop can repeat until a satisfactory design is found.

Outputs from The ICAD System

The mold-base application, for example, uses a translator to a flow-analysis package. Data from a mold-base design instance is formatted and written to a computer file that can be read by the analysis program. The analysis program analyzes the data from the computer file. The analysis results are read as system inputs and used to reconfigure the cavity and runner configuration.



Mold-base product model output to flow-analysis package. The analysis result is read back into the ICAD product model as system inputs, forming a closed loop.

Summary

- The three kinds of output are graphical output, report output, and output to another computer system (including CAD systems).
- Wireframe graphical images of the model can be drawing in the ICAD Browser. With additional software, full color shaded images can be displayed on various workstations.
- Reports can be generated using report-attributes.
- ICAD supports CAD output for various CAD systems.
- You can write customized translators for other computer systems such as analysis programs or proprietary CAD systems.

Integration Tools and Strategies

Keywords: engineering views, custom user interfaces

Overview

This section describes integration tools and strategies and includes the following topics:

- Organization of the product structure tree into different views.
- Different tools provided by ICAD for integration.
- Custom user interfaces.

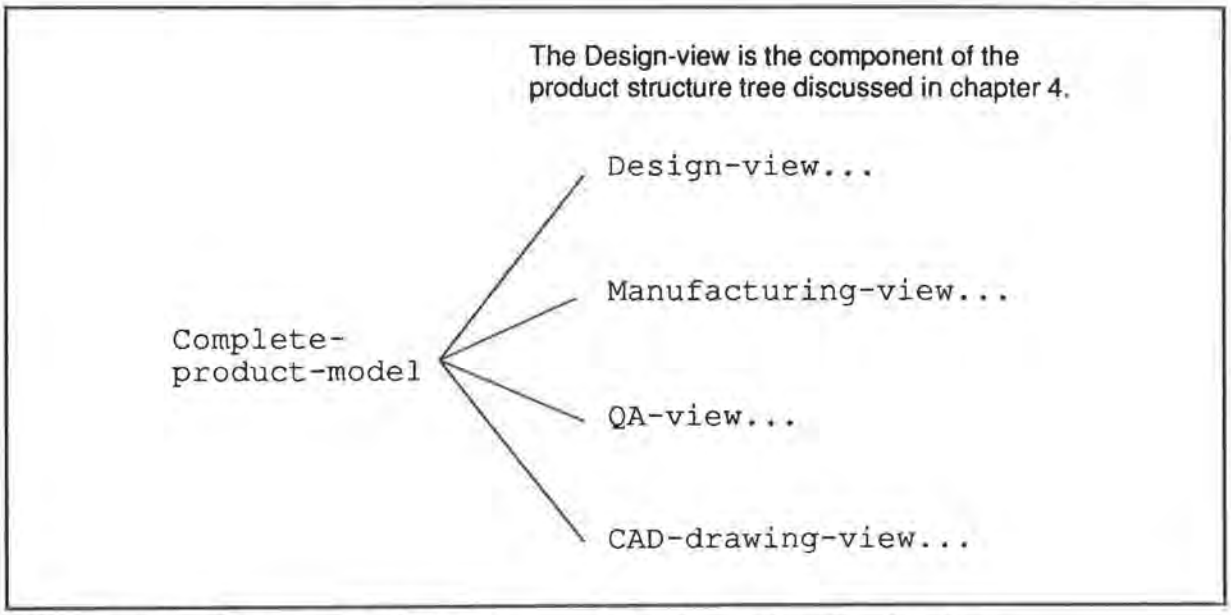
Organizing the Product Structure Tree for Integration

In the previous chapter, we saw how to organize the product structure tree for the product design (see the section "Creating the Product Structure", page 4-29). In this section, we add to the product structure tree so that it integrates the ICAD product model with other engineering systems.

Different engineering disciplines see the same product differently. There can be multiple *views* of the same product, that is, multiple ways to look at the same product. For example, a design engineer and a manufacturing engineer look at the same product differently. The design engineer is concerned with design issues such as form, fit, and function. The manufacturing engineer is concerned with manufacturing issues such as manufacturability, standard manufacturing features, and required manufacturing resources.

Integration Tools and Strategies

You can use The ICAD System to represent the different views of a product by dividing the root of the product structure tree into views of the product. Each view is a component of the root and includes its own subtree. One component represents the design view of the product; this is the component that we have been concerned with up to now. Any number of other components could be created to represent the manufacturing view, the CAD view, the analysis view, the quality assurance view, etc.

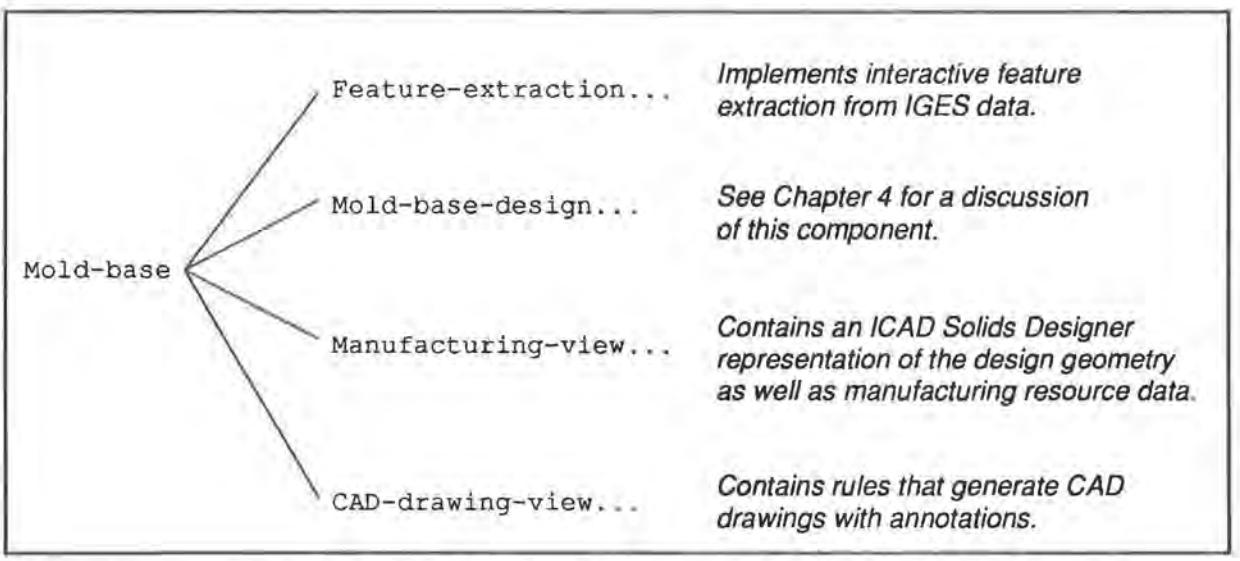


Different views of the same product are represented in the product structure tree using a different component of the root to model each view.

Usually the design view is developed first. The other views use ICAD Design Language tools such as symbolic referencing to read the appropriate data from the design view and interpret the data for their own purposes.

Integration Tools and Strategies

The root part of an integrated mold-base application, for example, is divided into four different views. A design view, implemented by the Mold-base-design part, represents the product design (discussed in the previous chapter). A feature-extraction view, implemented by the Feature-extraction part, recognizes features in the part to be molded. A CAD view, implemented by the CAD-structure part, organizes the geometry defined in the design view into different drawings for a CAD system, complete with annotations. A manufacturing view, implemented by the Manufacturing-structure part, translates the design into a solid model using the ICAD Solids Designer. The solid representation can be used to drive a process planner. The manufacturing view also contains rules and databases that represent the available manufacturing resources.



Different views of the same mold-base application represented in the product structure tree.

Tools for Integration

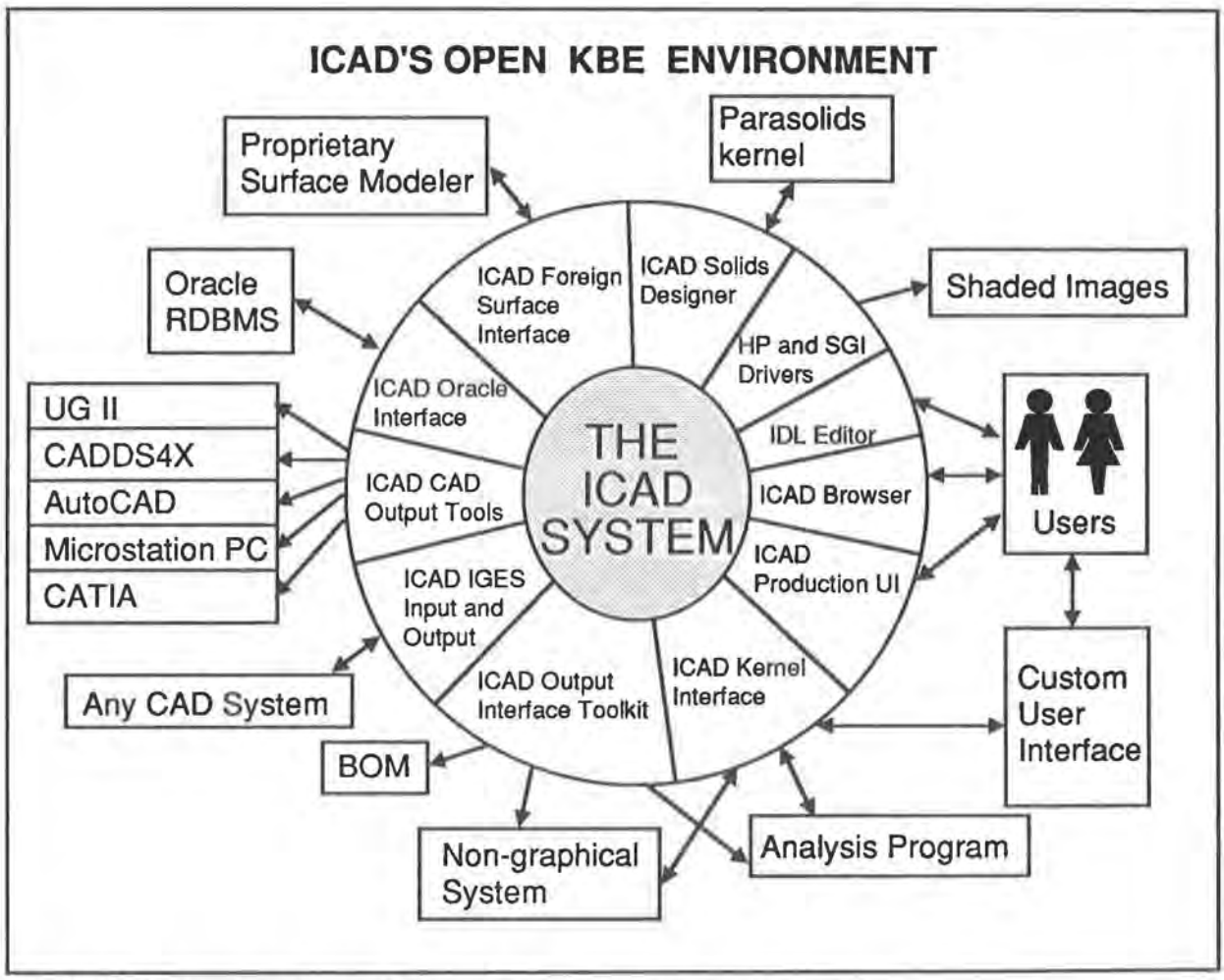
ICAD provides various tools for integrating an ICAD product model into the engineering environment. Most of these tools are available as layered products, that is, products added to the basic ICAD system.

- Specific CAD Output Tools automatically generate CAD files in IGES, Intergraph Microstation PC, AutoCAD, CADDS4X, or UNIGRAPHICS II format.
- The ICAD Output Interface Toolkit allows you to define reports, or generate output for any CAD system or computer analysis system.

Integration Tools and Strategies

- The IGES Input Tools automatically reads in data files in IGES format.
- The ICAD Oracle Interface provides an SQL interface to Oracle databases.
- The HP 300 Graphics Driver and Silicon Graphics IRIS 4D Graphics Driver can be used to generate shaded images on HP 9000 Series 300 SRX workstations or Silicon Graphics IRIS 4D workstations, respectively.

The following diagram shows The ICAD System integrated into an engineering environment.

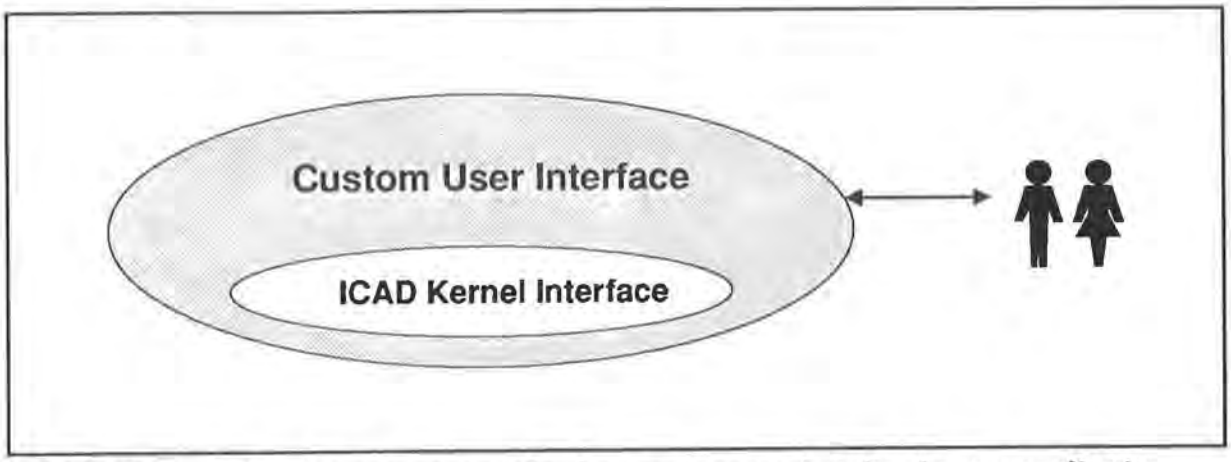


The ICAD System is integrated into an open engineering environment.

Integration Tools and Strategies

The ICAD System and Custom User Interfaces

ICAD can be used as a knowledge kernel for your own interface. The ICAD Kernel Interface allows you to design your own user interface that communicates directly with The ICAD System kernel. You can write your own interface to define defparts, create design instances from the defparts, and generate results from the design instance.



The ICAD Kernel Interface allows you to develop a custom user interface for your application.

For example, the UG-ICAD Connection product is a user interface developed by McDonnell Douglas that connects the Unigraphics II CAD system with The ICAD System. With the UG-ICAD Connection product, you create an ICAD product model in the standard way using The ICAD System. Then, an end user supplies the inputs to the ICAD product model and generates the outputs from the ICAD product model directly from the UNIGRAPHICS II CAD System.

Summary

- You integrate different engineering views in the same product structure tree by making each a component of the root.
- The ICAD System provides many different products that can be added to the basic system and used for integration purposes.
- You can use The ICAD System as a kernel for your own custom-designed user interface.

Summary of Chapter 5

- The ICAD System fits into a design pipeline that includes other kinds of engineering systems. Inputs come into the ICAD product model from the pipeline. The ICAD product model can interact with other computer systems or with the design engineer. Output from the final design is in the form of reports, graphical output, or machine readable output.

- Inputs can be from the design engineer or from other computer systems. Design-engineer inputs can be supplied when the design is first instantiated, or "interactively" while the ICAD product model is computing the design instance.

- Different "views" of the same product, e.g., manufacturing, QA, and design, can be incorporated into the product structure tree.

- The ICAD System provides many different products that provide an interface for the ICAD product model with other computer systems. Alternately, you can develop your own user interface using the ICAD Kernel Interface.

Appendix

Glossary

annotated drawings

Drawings that include dimensions, labels, etc. The ICAD System can produce annotated drawings automatically.

attributes

Engineering rules written in the ICAD Design Language. Attributes are always associated with defparts and can be calculated in many different ways. See *input-attributes*, *choice-attributes*, *descendant-attributes*, *report-attributes*, and *trial-attributes*.

bounding box

A box that encloses a 3D geometric object.

box

The basic geometric primitive in The ICAD System.

cached value

A value of an engineering attribute that has already been calculated and is stored in the computer memory. The next time the value of the attribute is required, it does not have to be recalculated.

CAD Output

2D- or 3D-geometric output from an ICAD product model in a format that can be read by a CAD system.

catalogs

Tables of engineering or other data. You can define rules in The ICAD System that automate the repetitive task of catalog lookup.

children

The components of a design instance in a product structure tree. Also referred to as *parts*.

choice-attributes

Engineering rules in which the design engineer is prompted for a value. See *attributes*.

Common LISP

A powerful general-purpose computer language upon which the ICAD Design Language is based.

compiling

The process of turning the engineering rules, that is, defparts, into a form the computer can understand.

Glossary

concurrent engineering
Engineering teams working in parallel on different aspects of the same product, for example, design and manufacturing. The ICAD System facilitates concurrent engineering.

product configuration
An application of The ICAD System that automates the organization of products with varying structure and many interrelated components.

custom user interface
An end-user interface developed for a particular ICAD application.

defpart
The basic building block of the ICAD Design Language. A defpart represents a component of the product model. It includes *inputs,* that is, input requirements, *attributes*, that is, engineering rules, and *parts*, that is, references to subcomponents.

demand driven
A feature of The ICAD System in which only those attributes that are needed to generate the desired result are calculated. For example, if you just want to generate the geometry of a product, the rules that relate to the cost analysis do not need to be computed.

descendant-attributes
An attribute whose value is made available to all the parts in a branch of the product structure tree.

design-engineer inputs
Inputs to an ICAD product model that are supplied by the design engineer, as opposed to being supplied by another computer system. See *system inputs.*

design instance
A particular design created by an ICAD product model. The ICAD product model takes the input specifications, applies the engineering rules, and creates a new design instance.

edit/compile/test loop
The process of developing an ICAD product model. First you edit the ICAD Design Language defparts, then you compile them into a form the computer can understand, and then you test them by generating a few design instances. If they are correct, you are finished. Otherwise, you repeat the process.

engineering rules
Representation of the strategy for designing or manufacturing the product. Rules describe how to divide the product into components, how to model the

Glossary

product's geometry, how to calculate various engineering attributes, how to describe the process of manufacturing or testing the product, etc. Rules are written in the ICAD Design Language using *attributes* and *parts*.

engineering views
Multiple ways to look at the same product. For example, a design view contains design information, a manufacturing view contains information about the processes used to manufacture the product, and a quality-control view contains information about testing the design. The same ICAD product model can incorporate multiple engineering views.

expert systems
Computer system that captures some amount of design knowledge in an automated decision-making format, generally in the form of "if-then" rules.

generative process planning
An application of The ICAD System in which the product model automatically generates process-planning information.

generic part
A defpart that describes a design component that can be used in multiple places in the product structure tree.

geometric primitives
ICAD-provided defparts that have geometric meaning, such as boxes, spheres, etc.

global coordinate system
The coordinate system in which the geometry of the entire product is represented. See *local coordinate system*.

graphical output
Display of design-instance geometry. The ICAD System generates 3D wireframe displays automatically.

ICAD Browser
ICAD-provided user interface to The ICAD System. In the ICAD Browser you create new design instances and generate output from them.

ICAD Design Language
An engineering specification language used to describe defparts that is provided by ICAD.

ICAD product model
A description of the design strategy required to produce a particular product from a product specification — in essence it is the set of engineering rules used to design the product. An ICAD product model is written using the ICAD Design Language as a set of defparts.

Glossary

incremental compiler
A feature of The ICAD System that allows you to change a small piece of a defpart without having to recompile the entire defpart. See *edit/compile/test loop*.

input specifications
Engineering attributes whose values are supplied by the design engineer when a new design instance is generated. Changing the input specifications can drastically change the design instance. Input specifications are written in the ICAD Design Language as *input-attributes* or *choice-attributes*.

inspector
A tool in the ICAD Browser that allows you to see the value of attributes for a particular design instance.

instantiation
The process of creating a new design instance from an ICAD product model.

interactive input specifications
Inputs to an ICAD product model for which the engineer is prompted by the ICAD product model. Interactive input specifications are written in the ICAD Design Language as *choice-attributes* or as *specification sheets*.

KBE
Acronym for knowledge-based engineering.

kind-of inheritance
A feature of The ICAD System in which a new defpart can be written that includes all of the attributes and parts of an existing defpart. See *mixin*.

Knowledge-based engineering
An engineering method in which engineering knowledge about the product, e.g., the techniques used to design, analyze, and manufacture a product, is stored in a product model which represents the engineering intent of the design.

leaves
A part in a product structure tree that has no children.

local coordinate system
The coordinate system in which the geometry of the components of an assembly in the product structure tree is represented. By defining the components of an assembly in the assembly's local coordinate system, the assembly can be positioned and oriented as a whole with respect to the overall product in the global coordinate system. See *global coordinate system*.

Glossary

mixin
A defpart whose attributes and parts are included in the description of another defpart. We say that the new defpart mixes in the existing defpart. See *kind-of inheritance.*

modularization
Breaking up an application into discrete components such that developing each component separately and then integrating the components is significantly easier than developing the application all at once. ICAD provides many tools for modularization, including dividing the product into components of a product structure tree and dividing defparts into mixins.

object-oriented
A way to develop computer systems such that the basic form of the program is an "object" that includes both data and instructions for manipulating data. An ICAD product model is an object-oriented computer system; each object is described by a defpart, which includes both data for the object and rules for relating it to other objects. Many of the features of The ICAD System, including mixins, attributes, parts, etc., are a direct result of its being an object-oriented system.

parametric curves and surfaces
Mathematically-described free-form curves and surfaces. The ICAD Surface Designer lets you create products whose geometry is based on parametric curves and surfaces.

parametric CAD system
A type of modern CAD system that lets you relate the geometry of different elements of a product. When you change one element, the geometry of the rest of the product changes as well.

parts
The children of a design instance in a product structure tree. Parts indicate how a design is divided into components. You specify parts when you write a defpart.

part-whole defaulting
A feature of The ICAD System that allows attribute values to be inherited from other parts that are higher up in the product structure tree.

product model
See *ICAD product model.*

product structure tree
The components of a design instance of an ICAD product model are organized into a product structure tree. A component might represent a product assembly, a part, or geometric feature. Alternately, a component might represent a manufacturing process.

Glossary

referencing
A feature of The ICAD System that allows you to use the value of an attribute in one component of the product structure tree to compute the value of an attribute in another component of the product structure tree.

remote evaluation
Using a computer system other than The ICAD System to compute the value of an attribute.

report-attributes
Attributes that let you generate output from a design instance.

reports
Textual output from a design instance. Typical reports include bills of material, routing sheets, etc.

root
The design instance that is the top of the product structure tree. The root of the product structure tree represents the design instance of the ICAD product model.

rule base
The rules that form an ICAD product model.

rule-based system
Another name used to describe a knowledge-based engineering system.

specification sheets
A technique for interactively specifying the inputs to an ICAD product model by selecting and editing them in menus or by identifying them graphically.

semi-custom design
An application of The ICAD System in which a standard design technique is used, with minor modifications, to produce slightly-differing designs.

solid modeling
A geometric modeling technique that can be used to build and manipulate solid objects and combine them into assemblies. Complex solid objects are built from primitive shapes using Boolean operations such as unite, subtract, and intersect. Primitive shapes usually include analytic solids such as a block, cylinder, cone, sphere, and torus, linear and rotational sweeps, and parametric (free-form surface) sheets.

surface modeling
A geometric modeling technique used to build and manipulate free-form surface objects. A surface is represented using parametric equations that allow the use of many geometric operations such as sweeping, extrusion, etc.

Glossary

system inputs
Inputs to an ICAD product model that are supplied by another computer system, as opposed to a design engineer. See *design-engineer inputs*.

translators
A tool in The ICAD System that translates information in a design instance into a form that can be understood by another computer system. For example, a translator is used to generate output to a CAD system. ICAD provides translators for several CAD systems, as well as tools for writing your own translators.

trial-attributes
Attributes whose values are computed by trying out new values until a successful value is found.

unbound values
An indication that the value of a particular attribute has not yet been computed.

variational geometry systems
A type of modern CAD system that allows you to define omni-directional geometric constraints on 2D geometry.

ICAD, Inc.
201 Broadway
Cambridge, MA 02139

Price $25.00