AFHRL-TP-90-81



(2)

AIR FORCE

HUMAN RESOURCES

LABORATORY



# DESIGN KNOWLEDGE MANAGEMENT SYSTEM (DKMS)

Richard J. Mayer, et al.

Knowledge Based Systems, Inc.
2746 Longmire
College Station, Texas 77845-5424

LOGISTICS AND HUMAN FACTORS DIVISION
Wright-Patterson Air Force Base, Ohio 45433-6503



December 1990

Final Report for Period September 1989 - August 1990



AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5601

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this paper, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

BERTRAM W. CREAM, Technical Director
Logistics and Human Factors Division

JAMES C. CLARK, Colonel, USAF
Chief, Logistics and Human Factors Division

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | December 1990 | Final Paper - September 1989 to August 1990 |

**4. TITLE AND SUBTITLE**

Design Knowledge Management System (DKMS)

**5. FUNDING NUMBERS**

C - F41622-89-C-0018
PE - 65502F
PR - 3005
TA - L2
WU - 03

**6. AUTHOR(S)**

Richard J. Mayer, et al.
2746 Longmire
College Station, Texas 77845-5424

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Knowledge Based Systems, Inc.
2746 Longmire
College Station, Texas 77845-5424

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)**

Logistics and Human Factors Division
Air Force Human Resources Laboratory
Wright-Patterson Air Force Base, Ohio 45433-6503

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

AFHRL-TP-90-81

**11. SUPPLEMENTARY NOTES**

Prepared as final report for Phase I Small Business Innovation Research (SBIR) contract effort. 

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Approved for public release; distribution is unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** (Maximum 200 words)

This document describes the Design Knowledge Management System (DKMS), which provides a software environment for both the development and the delivery of intelligent assistants. These intelligent assistants can be used for computer-aided design (CAD), computer-aided engineering (CAE), and computer-aided manufacturing (CAM) applications in product design, engineering, manufacturing, and logistics planning.

The DKMS can be thought of as an integrated concurrent engineering system whose environment includes facilities for (a) design knowledge representation including shape-based associative retrieval and container objects, (b) intelligent user interface including form feature interfaces and a generalized constructive solid geometry (GCSG) engineer, (c) engineering performance modeling functionality including constraint management and bond graph techniques, (d) data integration support, and (e) configuration management. This facility will enable the delivery of design advice on-line to the designer as new definitions are being created. Backing up this geometry engine, a container object management system will be produced that extends the proven composite object capabilities by integrating geometry concepting and manipulation primitives into the basic object definition and inheritance lattice operators. As an initial manufacturability, reliability, and maintainability capability, the baseline system will include a manufacturability checker, an associated generative process planning system, and an engineering performance model development environment.

**14. SUBJECT TERMS**

computer-aided design (CAD)
computer-aided engineering (CAE)
computer-aided manufacturing (CAM)
concurrent engineering
decision support
unified life cycle engineering (ULCE)

**15. NUMBER OF PAGES**

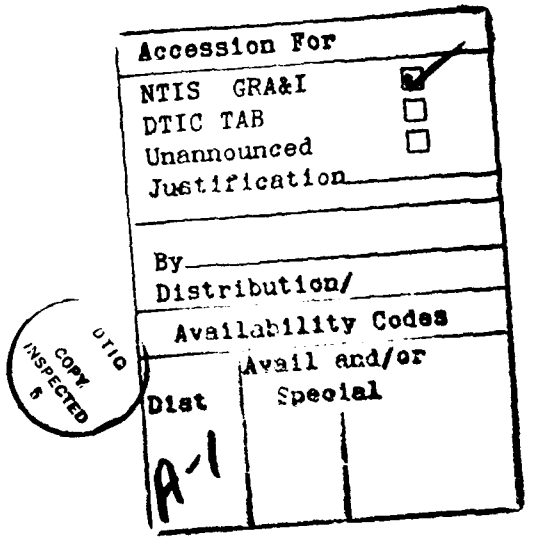
144

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |

Item 13 (Concluded):

This environment is intended to be used to support the rapid experimentation, prototyping, and development of a new generation of integrated engineering and manufacturing decision support applications to meet the challenges of concurrent engineering. Most important, this architecture will enable the capture of and delivery to the engineer of product life cycle experience relative to manufacturability, reliability, and maintainability.



AFHRL Technical Paper 90-81 December 1990

# DESIGN KNOWLEDGE MANAGEMENT SYSTEM (DKMS)

Richard J. Mayer, et al.

Knowledge Based Systems, Inc.
2746 Longmire
College Station, Texas 77845-5424

LOGISTICS AND HUMAN FACTORS DIVISION
Wright-Patterson Air Force Base, Ohio 45433-6503

Reviewed by

Bertram W. Cream, Technical Director
Logistics and Human Factors Division

Submitted for publication by

James C. Clark, Colonel, USAF
Chief, Logistics and Human Factors Division

## SUMMARY

The Design Knowledge Management System (DKMS) provides a software environment for both the development and the delivery of intelligent assistants. These intelligent assistants can be used for computer-aided design (CAD), computer-aided engineering (CAE), or computer-aided manufacturing (CAM) applications in product design, engineering, manufacturing, and logistics support. The DKMS focuses on the overall system problems rather that just individual solutions and differs from general efforts in knowledge-based management systems by its:

1) focus on engineering knowledge,

2) addressing of heterogeneous forms and the distributed nature of such an engineering knowledge base,

3) focus on the shape nature of the indexing and organization of such engineering knowledge, and

4) consideration of knowledge acquisition, application support, and evolution management as well as storage and retrieval.

The DKMS can be thought of as an integrated concurrent engineering system providing facilities for:

1) design knowledge representation including shape-based associative retrieval and container objects,

2) intelligent user interfaces including form feature interfaces and a generalized constructive solid geometry (GCSG) engine,

3) engineering performance model development support functionality including constraint management and bond graph techniques,

4) tools and utilities for construction of knowledge-based engineering assistants for the entire scale of life cycle engineering knowledge support needs,

5) data integration support, and

6) knowledge configuration and version management support. This architecture enables the capture and delivery, to the engineer, of product life cycle experience relative to manufacturability, reliability, and maintainability (MR&M).

This Phase I SBIR effort demonstrated the viability of an environment that includes generic utilities for:

1) shape- and feature-based knowledge representation,

2) a GCSG modeling capability,

3) geometry-based container object modeling systems,

4) a multi-schema, object-based Common Data Manager (CDM),

5) engineering data configuration manager, and

6) smart Knowledge Base Editors/Browsers (KBEB).

# PREFACE

The purpose of this technical paper is to document work performed under a Phase I, Small Business Innovative Research (SBIR) effort. The effort was supportive of AFHRL efforts in Unified Life Cycle Engineering, a precursor initiative to the larger, DOD-initiated, effort in Concurrent Engineering under the umbrella initiative of Total Quality Management.

This report describes the concepts, algorithms and designs developed during the Phase I effort. The investigations, designs, and prototypes reported on are intended to support a Phase II SBIR development effort.

## Table of Contents

## List of Figures

# Design Knowledge Management System - DKMS

The Design Knowledge Management System (DKMS) provides a software environment for both the development and the delivery of intelligent assistants. These intelligent assistants can be used for computer-aided design (CAD), computer-aided engineering (CAE), and computer-aided manufacturing (CAM) applications in product design, engineering, manufacturing, and logistics support.

In the seventies and early eighties, a crisis was recognized in US industries relating to a slip in the rate of growth of industrial productivity and quality of US made products. This recognition gave rise to an increase in investment in automation and modernization of manufacturing equipment, facilities, systems, organizations, and labor management. In the late eighties, a shift of the focus to problems in worldwide competitive position forced the recognition that many of the previous problems were actually symptoms of deeper-rooted issues addressing the entire corporation from marketing and finance through research, engineering, manufacture, and field support. Among the key issues affecting the erosion of worldwide competitive position are:

1) Comparatively long lead times to take a concept to rate production,

2) Inability to assimilate research results into product or production technology,

3) Lack of a total systematic approach to quality management,

4) A retiring or otherwise lost knowledge base,

5) High rates of change in the product definition (and the production process) during buildup to rate production and upon any major product or process change.

In short, the DKMS is proposed as a part of the answer to: Why does it take so long to design/engineer/produce a product? Why are so many of the mistakes of the past repeated? Why are we always the last to use the new ideas and technology being produced by our own research laboratories? The part of the problem that DKMS addresses is focused on the acquisition, management, and effective delivery of engineering knowledge, experience,

and rationale. Lest the reader be misled by common biases, we use the term engineering in the broad sense to include product design, product engineering, manufacturing engineering, process engineering, maintenance engineering, and sustaining engineering. In the remainder of this document we will refer to this holistic enterprise engineering knowledge base as the life cycle engineering knowledge base. It is this life cycle engineering knowledge that must be marshalled to achieve the goals of concurrent engineering and total quality management.

Distinguishing the DKMS effort from general efforts in knowledge base management systems are (1) the focus on engineering knowledge, (2) the addressing of heterogeneous forms and the distributed nature of such an engineering knowledge base, (3) the focus on the shape nature of the indexing and organization of such engineering knowledge, and (4) the consideration of knowledge acquisition, application support, and evolution management as well as storage and retrieval. What distinguishes the DKMS effort from previous design expert systems is the focus on the overall system problems rather than just the individual solutions. These problems and knowledge-sharing issues that form the requirements for the DKMS are discussed in more detail in Section 2 of this report.

The DKMS can be thought of as an integrated concurrent engineering system which provides an environment with facilities for (1) design knowledge representation including shape based associative retrieval and container objects, (2) intelligent user interfaces including form feature interfaces and a generalized constructive solid geometry (GCSG) engine, (3) engineering performance model development support functionality including constraint management and bond graph techniques, (4) tools and utilities for construction of knowledge-based engineering assistants for the entire scale of life cycle engineering knowledge support needs described previously, (5) data integration support, and (6) knowledge configuration and version management support. This environment is intended to be used to support the rapid experimentation, prototyping, and development of a new generation of integrated design, engineering, manufacturing and maintenance decision support applications to meet the challenges of concurrent engineering. Most important, this architecture will enable the capture and delivery to the engineer of product life cycle experience relative to manufacturability, reliability, and maintainability (MR&M).

Sections 2 through 12 of this report describe the concepts, algorithms, and designs developed during our Phase I SBIR effort. We have demonstrated the viability of an environment that includes generic utilities for:

1) Shape & Feature Based Knowledge Representation of engineering, manufacturing, and maintenance knowledge associated with the geometric properties of the part. This includes knowledge, relevant to design, planning, costing, and scheduling decisions, that is associated with shape. Shape-based means that the knowledge representation scheme can handle shape references that have no symbolic names. This provides the capability to represent "conditions on" or "contents of" design knowledge that contains arbitrary geometry structures. Feature-based knowledge representation handles geometry references which have named prototypes (hole, slot, pockets, etc.).

2) A Generalized Constructive Solid Geometry (GCSG) modeling capability including form feature-based user interfaces for high-productivity CAD man/machine interfaces. Form feature interfaces support geometry concepts using named features as well as the ability to define prototypical forms as new feature sets. Such high productivity CAD interfaces are required to enable the acquisition of shape based knowledge and to allow its delivery to the personnel actually making the design/engineering decisions.

3) Geometry Based Container Object modeling systems needed for efficient delivery of the compiled design, manufacturing, and maintenance knowledge across the product life cycle.

4) Multi-Schema Object-based Common Data Manager (CDM) for information and knowledge base integration support and integration with existing CAD and CAM application systems.

5) Engineering Data Configuration Manager (EDCM) for control of technical data objects throughout their life cycle including version management to support the controlled evolution of those technical data objects as well as the associated knowledge units.

6) Smart Knowledge Base Editors / Browsers (KBEB) for support of human interaction with the evolving engineering knowledge bases. The primary use of the KBEB is for ad hoc application of the design and product knowledge bases. They will also support tutor modes of use of the static portions of that knowledge base.

Layered onto this platform is a development support environment including a powerful complement of system, software, and knowledge engineering tools and utilities as shown in Figure 1.

The investigations, designs, and prototypes reported in this technical report are intended to support a Phase II SBIR development effort. The objectives of the Phase II project are to implement an initial DKMS based concurrent engineering support environment. This support environment includes (1) life-cycle engineering knowledge representation, (2) knowledge acquisition support, (3) knowledge storage, and (4) retrieval utilities, as well as (5) knowledge and information integration utilities. It also includes the development of the Generalized CSG engine and form feature geometry input interfaces to allow rapid capture of shape-based design knowledge. This facility will enable the delivery of life-cycle advice on-line to the designer as new product definitions are being created.

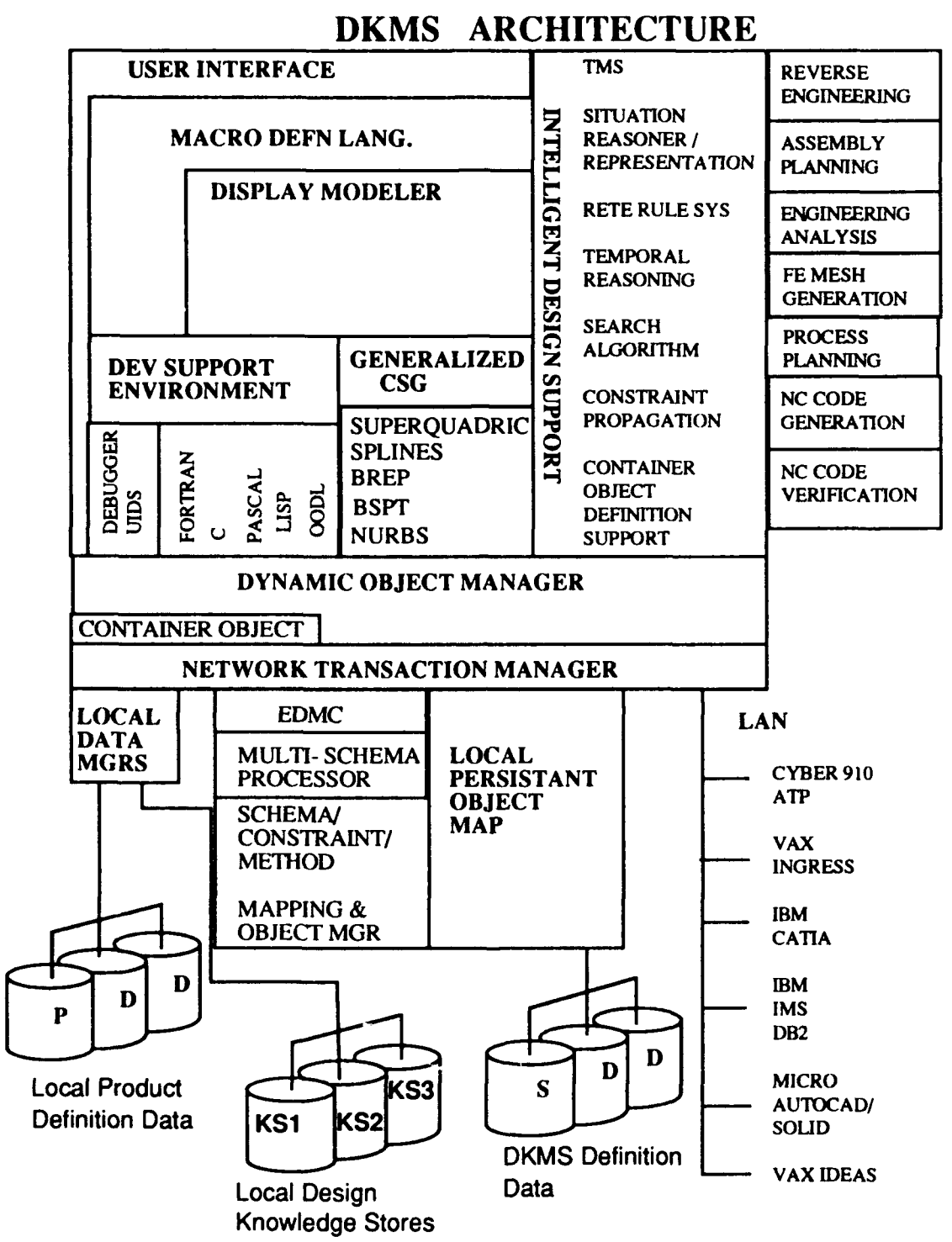



**Figure 1: DKMS Architecture Overview**

Backing up this geometry engine, a container object management system will be produced that extends the proven composite object capabilities by

integrating geometry concepting and manipulation primitives into the basic object definition and inheritance lattice operators. As an initial MR&M (manufacturability, reliability, and maintainability) capability, the baseline system will include a manufacturability checker, an associated generative process planning system, and an engineering performance model development environment. To ensure the usability of the resulting system, industrial advisors and beta test sites will be established and operated throughout the life of the project.

The completed Phase II will represent a major milestone in architectures, knowledge representation techniques, and geometry acquisition/reasoning methods for concurrent engineering. It will establish a demonstration (and possibly a de-facto standard) for engineering support environments for the next 15 years. These results will have widespread potential for application in private industry, government operations, the defense community, and universities. The Phase II results will enable a Phase III effort focused primarily on commercialization and rehosting to a wide variety of hardware platforms. This level of product maturity (due to the beta testing with industrial partners during Phase II) will significantly enhance the ability of KBSI to secure Phase III support. In fact, we believe that a significant level of support will be available from the direct marketing of the Phase II results. This will be possible because of the availability of the Symbolics development hardware as imbedded processors in traditional hardware hosts (currently Sun and Macintosh).

## 2. The Flow of Knowledge Required to Support Concurrent Engineering

The DKMS approach to the solution of the serious problems and challenges facing U.S. industries is to support the assimilation, application, access, and maintenance of the life-cycle engineering knowledge base. This section provides some observations on the complexities of the flow of knowledge that the DKMS was designed to facilitate.

The complexity of the design, engineering, manufacture, and maintenance of maintainable, reliable, safe, high-performance systems that employ the state of the art in technology within tight cost and schedule constraints requires the assembly and effective delivery of life-cycle engineering knowledge assistance to the design/engineering activities (whether these engineering activities occur in product, manufacturing, or maintenance arenas). As a minimum, such assistance includes:

1) Application of previously acquired knowledge in the engineering area to the task at hand. The product designer must have efficient access to the knowledge of product design accumulated by his predecessors. The manufacturing or process engineer must have access to the knowledge of previous experts.

2) Application of knowledge from outside the engineering area to the task at hand. The product designer must have assistance in the application of the experiences and lessons learned by the manufacturing and maintenance engineering discipline.

3) Access to a trace of the decision rationale that led to that most current state of the product or process definition. The product engineer must be able to quickly uncover the rationale of the product designer. The manufacturing engineer must be able to lay open the decision rationale and assumptions of both the product designers and engineers.

4) Access to the most current state of the product or process configuration description.

5) Man machine interfaces to support the capture of product definition data as the decisions relating to that data are made and not after the fact. If these don't exist, delivery of concurrent engineering support is almost impossible. Similarly, the accumulation of the rationale, experience, and knowledge bases that the concurrent engineering support would be based on cannot be affordably achieved without such interfaces.

Management of the assimilation and flow of knowledge is complicated by both long project life spans and short life spans. For example, in the NASA environment, project managers used to be able to see from start to finish several major initiatives within a single assignment. However, the space station and other advanced proposed projects have development and operation life spans that exceed an entire career. Excessively long development and operation cycles mean that the use of the "human" as the knowledge base manager has no chance of success. In addition, there is a problem with accumulation of knowledge based on multiple experiences with similar situations. In contrast to this, a 60-month (or longer) development process has for decades been the norm in the automotive industry, whereas now the goal is to reduce this concept to rate production to less than 30 months. Dramatically reduced development cycles impair the formation of knowledge, denying the opportunity for reflection and

experimentation. Throughout the life-cycle, the result of compression of the engineering results in critical point engineering rather than the development of any complete understanding of the features and characteristics of either the product or problem environment.

To give the engineer visibility into the Life Cycle implications of a design decision requires giving access to historical information. That is, there are many unquantifiable factors in the design process where the designer must make judgments that result in implications manifesting themselves much later in the product life cycle. In large, long-life, complex products such as modern weapons systems, the original design team is long since gone when the product exhibits design problems. The only way to provide the new design team with insight into the correlation between the design decisions and their implications is to provide a means of capturing the design rationale and correlating this rationale and the product definition information to the effects (lessons learned or experience of the manufacturing or maintenance engineering activities). Both the original designers and the sustaining engineering staff need access to manufacturing, maintenance, and logistics experience.

There is a need for flow forward and flow in of knowledge relative to the product development cycle. Design rationale, alternatives considered, and lessons learned in the process of life-cycle design/engineering must flow forward to the next individuals in the development chain as shown in the upper portion of Figure 2. Similarly, experiences, lessons learned, and limitations discovered must flow from downstream engineering activities of previous projects back to upstream engineering activities of the current project as illustrated in the lower half of Figure 2.

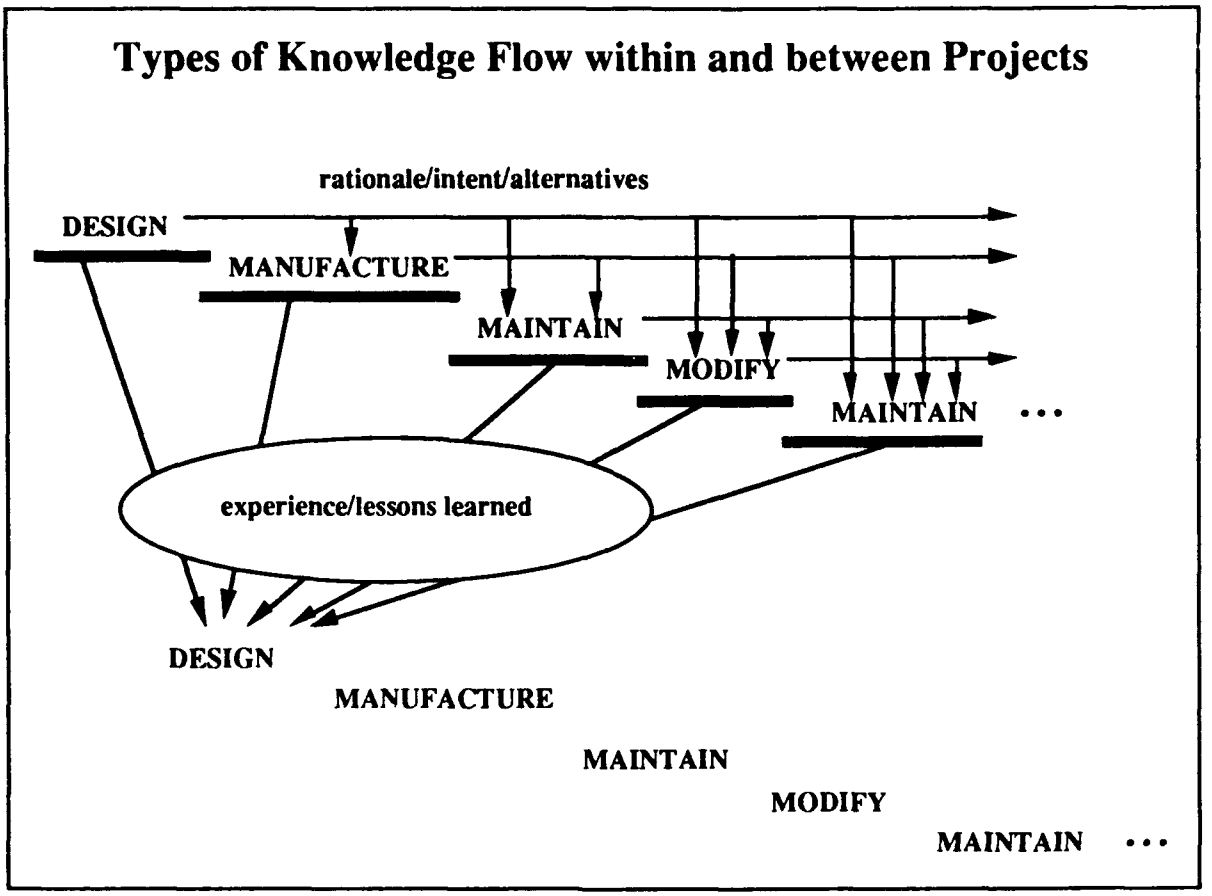



**Figure 2: Typical Types of Knowledge Flow Within and Between Projects**

The flow of knowledge that needs to be supported by the DKMS appears to be somewhat (though not radically) different from the flow of information studied for many years in support of information integration and information resource management efforts. Relative to a particular product development effort, the knowledge flow tends to be unidirectional whereas the information flow is bidirectional, as shown in Figure 3. For example, one of the primary flows of information into the product design activities is feedback information from product engineering and manufacturing engineering activities related to the current product definition. On the other hand, the flow into product design of concurrent engineering knowledge assistance is most often the accumulation of experience from many previous product histories. The loss of the current product definition as a frame of reference for the knowledge flow makes the management of, access to, and application of the knowledge resource much more difficult (as though management of the flow of information were not difficult enough).

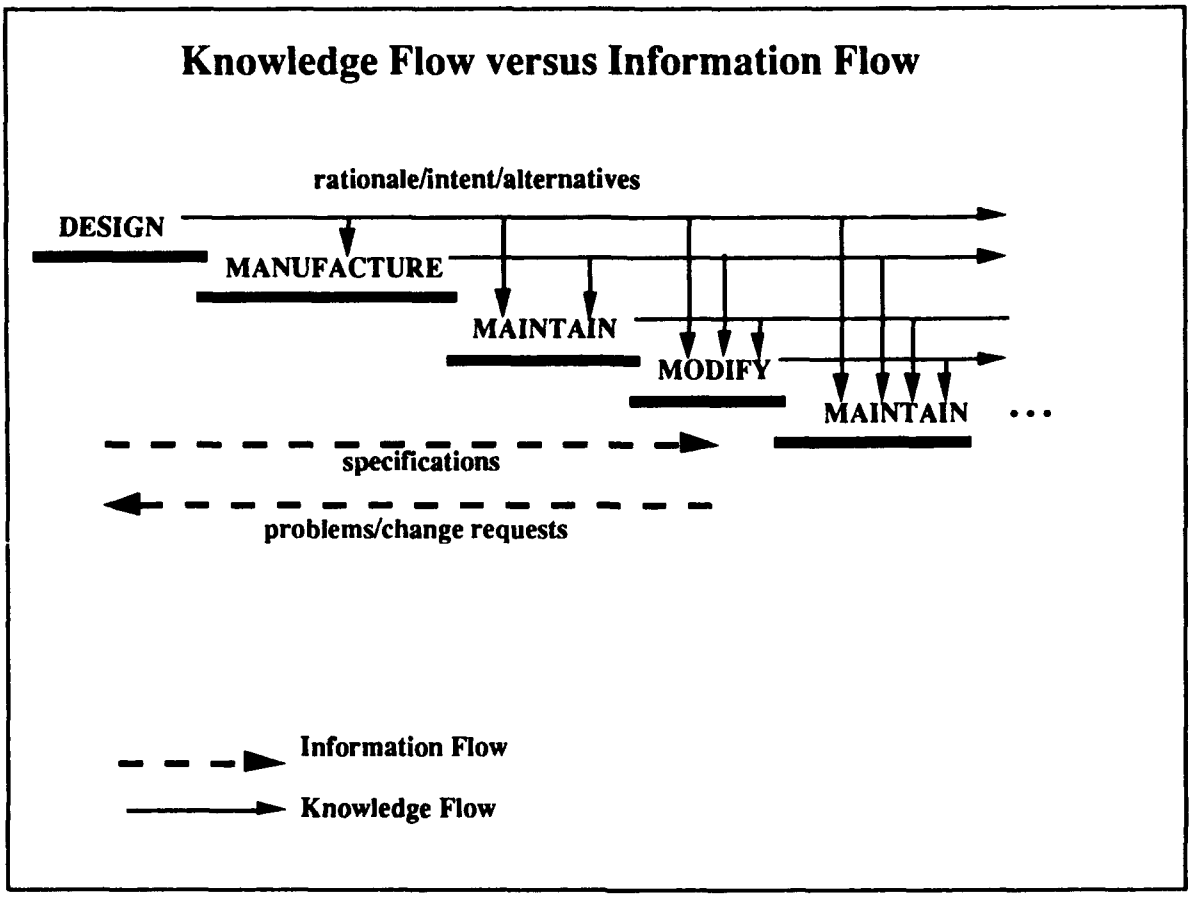



**Figure 3: Predominant Flows for Knowledge and Information**

There is a need for capture of design knowledge

To support passing on of that knowledge to future engineers,
To support downstream decision making by other product designers,
To support downstream decision making by manufacturing engineers,
To support downstream decision making by field support & maintenance engineers,
To support downstream decision making by sustaining engineering efforts.

How can these knowledge flows be assisted and effected? First, we must efficiently be able to capture such knowledge. Second, we must be able (hopefully efficiently) to represent such knowledge. Third, we must be able to deliver the necessary advice or recommendations implicit in this knowledge to whatever task requires those recommendations. Fourth, we

must be able to manage the evolution of such knowledge because (1) we don't want to deliver out-of-date recommendations, (2) we don't want to accumulate lessons learned on out-of-date product definitions, and (3) we can't afford to deliver recommendations on out-of-date product definitions. We also need more efficient means for generation of accurate and complete specifications as a part of the engineering concepting / analysis / decision-making process and not as an afterthought. Finally, we must improve the use and availability of engineering performance models. The product design / engineer needs faster access to product design evaluation mechanisms. The manufacturing planner/engineer needs faster access to process / plant design evaluation mechanisms. The maintenance engineer needs faster access to product design / process / plant design modification evaluation mechanism. These types of performance prediction models speed the acquisition of knowledge by these various engineering groups.

The DKMS, as described in the remainder of this report, attempts to serve these needs. The next section of this report provides a possible scenario of use of the DKMS to illustrate how these needs might be met operationally.

## 3. Envisioned DKMS Scenario of Use and Benefits

When the Design Knowledge Management System (DKMS) based CAD/CAE/CAM reaches maturity, the following user scenarios will be possible:

- A product definer/designer accesses requirements, chooses previous design data and, if the design is routine, invokes a knowledge-based system designer based on design knowledge embedded in a container object system to develop concepts for a new product.

- A designer sketches geometry concepts into the system which provides dimensioning support as well as material, tolerance, and finish specification support. In cases where the design is routine and both compiled design knowledge and engineering analysis models exist, analysis of the shape and forms of the sketches by the system allows for automated specification generation. Using the DKMS facilities, the evolving shape is mapped against a knowledge base of fabrication, assembly, and maintenance/support experience, and potential problems are indicated to the designer. Using this same shape associative retrieval mechanism, the designer can request suggestions for alternative shapes with similar functional

characteristics but with better manufacturability, reliability, and maintainability (MR&M) life cycle characteristics.

- When engineering performance prediction models do not exist, the designer uses the Modeler support environment to rapidly adapt existing models or to generate new models. Using the geometric reasoning capabilities of the DKMS, the necessary geometry based input to these models can be easily prepared.

- At any time, the entire existing set of product definition data, including geometry and design attributes, can be passed to the manufacturability, assemblability, or reliability assessors for analysis and life cycle cost estimation.

- As the design matures, it can be released to the manufacturing engineers who use the generative planning capabilities to generate, evaluate, and simulate process plans, assembly plans, and quality assurance plans for the proposed product.

- When the designer moves into system level design situations which involve heterogeneous components, or when a new designer is added to the team, they can browse the interned design knowledge sources. Using these resources in a tutorial fashion, he can update himself on both basic principles as well as specialized knowledge about specific components and devices.

To provide the above illustrated support for design engineers to better integrate the tradeoff of various design attributes such as performance, costs, schedule, manufacturability, and supportability requires the ability to:

1) Capture, store, and reason about an efficient representation of concurrent engineering knowledge including:

   a) Design knowledge such as: design rationale, design intent, design limits, and constraints,

   b) Manufacturing fabrication, procurement, assembly, and test experience,

   c) Maintenance and field support experience.

2) Capture the design attributes determined at various stages in a design life cycle.

3) Relate the design attributes and designer's intent to the product definition.

4) Distribute the above relations to any level in the product definition including the form features of an individual part of the product.

5) Support the rapid entry of a design concept or intent, particularly when that concept requires the definition of geometry-related information.

6) Effectively control the design data evolution process (including the product definition data, engineering model data, test data, as well as manufacturing, operations, and logistics experience data) over the entire product life cycle.

7) Provide integrated systems combining:

   a) Multiple knowledge-based assistants,

   b) Engineering modeling and analysis programs, and

   c) Traditional engineering, product, manufacturing, and maintenance data base systems

KBSI's method of providing these capabilities is centered around the development of a Design Knowledge Management System (DKMS) with an underlying Generalized Constructive Solid Geometry (CSG) capability to allow for the:

1) Integration of a number of "design decision support tools."

2) Development of "designer" systems which automatically make the desired design tradeoff decisions.

3) Reduction in time for concept entry, particularly the entry of form/shape geometry data and the generation of "design-tradeoff-model" input data derived from that product geometry data.

4) Delivery of qualitative assessments of design options by providing the ability to associatively index historical knowledge structures

around form feature interpretations of the evolving product geometry.

The proposed DKMS would be usable as a base concurrent engineering platform for almost any engineering automation effort in the Government, Defense Contractor, and Commercial industrial sectors. It would provide a quantum improvement over any available design automation concept available today. It could also serve as a means to promote better university/industry/government ties in that it would provide a direct vehicle for moving design, manufacturing, and field experience into the academic classroom. It is anticipated that initial installations would be in (1) very aggressive small business communities which must leverage scarce resources, (2) DoD installations where management of knowledge bases over long life-cycle weapon systems is a critical issue, (3) NASA Space Station and deep space missions where knowledge bases will span multi-careers, (4) commercial industries where reduction in product development time is critical to the maintenance of a competitive position. The results of this project will be focused primarily on mechanical device design support. We have been careful to track and maintain (at least conceptual) compatibility with the Air Force initiative in electronic system design integration [EIS 1989]. We also are monitoring the Air Force and NASA initiatives in integrated software design and development environments. The DKMS will be able to be integrated with these systems as they emerge. This will allow complete concurrent engineering support across all the major technology types involved in complex products (weapons systems to automobiles). We also believe that this technology will be applicable to the design and sustaining engineering for the manufacturing systems (plants to machine tools and robotic workstations) required to produce these 21st century products.

In the following section, we describe the results of our studies into the nature of the knowledge and life-cycle engineering processes that the DKMS must accommodate to support the above described scenario.

## 4. The Design Phenomena

What is design? Why can some people do it well and others not? Why does the design process of systems require so much time and yet still be so error prone? What mental processes are involved in understanding another person's design? What information constitutes the essence of a design description? Why in some situations does a short passage and a sketch communicate far more than volumes of structured documentation? These

are just a few of the questions which we investigated as a part of the Phase I effort to understand the knowledge environment that the DKMS must work within. While a complete answer is not necessary, a consistent characterization of the set of "reasonable" answers is needed to effectively define the DKMS requirements.

The Phase I plan called for the study of the IDEF models of the design process developed for the Air Force under the ICAM (Integrated Computer Aided Manufacturing) [Softech 81] and IDS (Integrated Design System) [Bsharah 86] programs. The results of this effort were disappointing as little insight was gained from those models into the nature or cognitive skills of design. Rather, these models tend to describe the organizational view of the design process. While such a view is valuable from the point of view of DKMS requirements for interfacing with the design culture of an organization, it provides little insight into the knowledge types or reasoning processes that the individual designer must possess and which the DKMS must support. This problem caused the Phase I team to synthesize its own models of design. The result of this effort is an understanding that design must be characterized from the following points of view:

1) Characteristics of the design situation--understanding what are circumstances where design is normally performed and understanding what constitutes an instance of a design process.

2) Characteristics of "modes" of design--delineation of major classifications of design activities during the product life cycle (the major focus of the ICAM and IDS models).

3) Characteristics of the cognitive skills or "generic activities" of the human designer--including a characterization of the distinguishing reasoning methods.

4) Characterization of predominant design strategies under various design situations--arrangements or orderings of the design activities to deal with different design situations.

5) Types of knowledge possessed by designers--characteristics of the content and structure of knowledge used to recognize the design situation, perform the design activities, and interact with the related product development processes within a particular organization.

6) Role of shape in design cognition--characterization of how shape concepting, shape triggered recall, and shape based constraints influence the generic activities of design.

7) Characterization of design rationale--the beliefs and facts as well as their organization that the human uses to make design commitments and propagate those commitments.

The following subsections provide an overview of several of the major findings in each area. These results set the stage for the architecture and major components of the DKMS presented in the remainder of this document. However, before getting into the details of design, there are four terms that we must characterize: system, subsystem, component, and system element. A basic characteristic of design is that it is an activity that produces specifications of artifacts. These terms allow us to talk in general about both the artifacts and their specifications.

A "system" is a collection of objects standing in purposefully defined relations to address some set of problems [Ramey 83]. It is the designer's task to forge such an association of related objects with a set of problems to make a system. It is this association (of the related objects with the problems that they together address) that forms the essence of a system. In the following sections, we will point out that all proposed models of design account for this association building process.

Both natural and man-made systems may consist of large numbers of objects, each having innumerable complex relationships and interdependencies. In fact, systems often appear to have arbitrary or even random structure [Ramey 83]. Several different organizational mechanisms are used by designers to assist in the management of this complexity, including the formation of hierarchies of both abstractions and idealizations as well as imposing hierarchical layers into the solution structure.

The term "subsystem" refers to an assemblage of objects within a system. A subsystem can be created by the designer for many different reasons, such as (1) partition of the design or development effort, (2) separation of a part of the problem for independent consideration, or (3) creation of an internal system to deal with a problem inherent to the solution design in mind. Subsystem creation can be recursive in that each subsystem defined within a system may be further partitioned into yet smaller subsystems. In fact, this decomposition of subsystems can be viewed as a part of the design process in that lower and lower level subsystems are created down to the

point that specifiable components (objects that can be procured or fabricated) are recognized. In general, a subsystem is any separable part of a system that has specifications that include reference to other specified objects. This part-whole structure will form the basis for an important design knowledge representation component of the DKMS called the container object system.

The lowest-level objects that are formally specified within a system are known as components. A "component" is an element of a system that does not need to be further specified in terms of the objects that it is made up of for its realization to successfully contribute to the satisfaction of the system requirements. Components may be systems or subsystems from the point of view of manufacturing or a supplier. However, from the point of view of design, they represent an important completion criteria. That is, they represent the isolation of an element of the system for which a specification can be developed that, if achieved, will result in an artifact that will perform correctly within the overall system design. In the system/ subsystem partitioning, components appear at all "levels" in that they are used to form the interfaces between the various levels. Finally, we use the term "system element" to refer to any of the previously characterized levels within a system. It is a generic term that may refer to a component or to any portion of the system.

## 4.1. Characteristics of the design situation

One of the means of characterizing design is by the situation types where it occurs. Characteristics of the design situation [Goel 89, Friel 88, Ramey 83] that we have chosen to describe the environment of use for the DKMS include:

1) Many degrees of freedom--the problem constraints do not determine the solution.

2) A large solution space--there may be an infinite number of very good solutions.

3) Delayed / limited feedback--the time between commitment or specification to realization of the artifact is long.

4) High cost of action--benefits are realized for correct decisions; penalties are paid for incorrect decisions.

5) Lack of clear right /wrong decision (evaluation) criteria.

6) Input consists of goals and intentions that are generally unclear and subjective.

7) Complexity of the problem addressed.

8) Many poorly understood parameters or subproblem interactions.

9) A large search space--parameters may take on a large number of possible values that could be reasonable in some context.

10) Existence of referents (reusable system elements)--many of the problems faced by the designer have been solved at least to some extent .

11) Necessity of producing a specification for an artifact.

12) Need for languages of thought to leverage the capabilities of the designer or to communicate the specifications.

13) Inherently iterative process.

14) Involves learning a mapping of the problem space onto the design space--the iterations are not random trial and error. Rather, later iterations employ knowledge gained from previous iterations on the same problem.

## 4.2. Characteristics of "modes" of design

The design process is a part of a product life cycle. It is a predominant activity early in the life cycle as well as in the sustaining engineering activities after the product has been produced. The notion of a design life cycle is a convenient device often used to help produce an understanding of the basic design processes, particularly for administrative purposes. The design process from such a view is assumed to begin at some point, to continue through maturity, and eventually to stop. This view of design as a series of incremental and sequentially interdependent steps is an attempt to order the steps of the process in such a fashion that each step can be looked on as an independent state except for its occurrence relative to the other states that surround it. This is in fact the view taken by the IDEFØ modelers in the ICAM and IDS models of design reviewed as a part of this

Phase I effort. Maintenance of an existing design is often overlooked in this way of viewing design [Bsharah 86]. Use of a system often turns up problems that either are not addressed by the system or are products of changes in the system's environment. These new problems must be addressed by the sustaining engineering activities as illustrated in Figure 4.

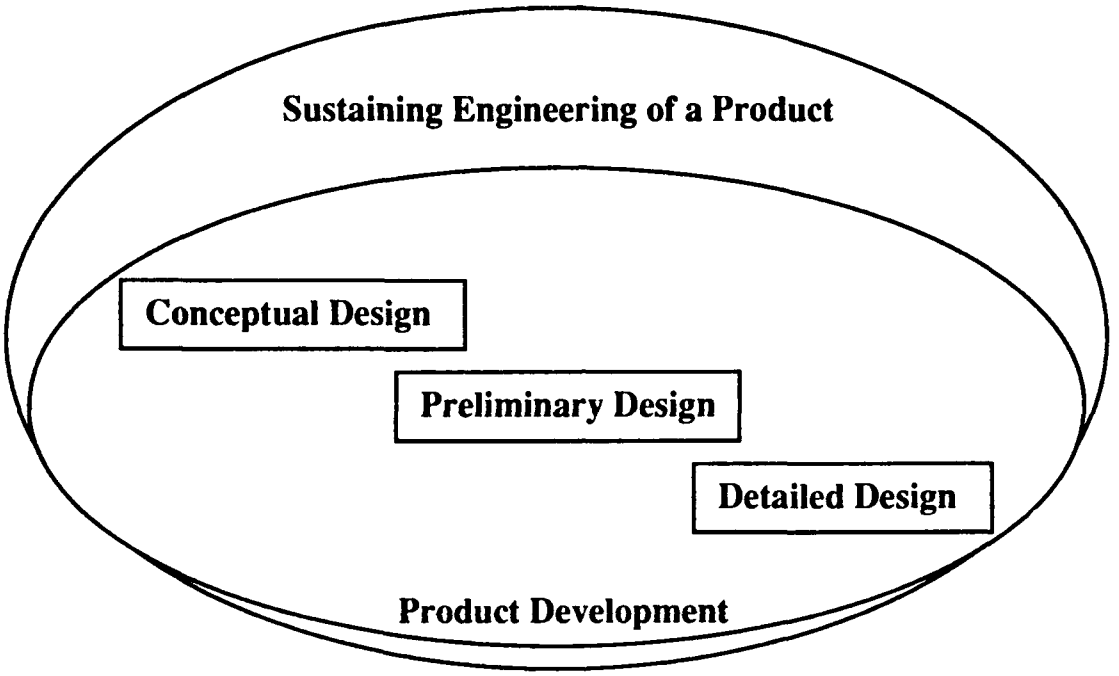



**Figure 4. Modes of Design**

As appealing as the life-cycle model of design may be, it is only one view needed for understanding the requirements for a DKMS. This is because the DKMS is targeted at the support of the individual designer's thinking process which, from studies of actual design instances, tends to take on an ad-hoc application of a number of skills described in a later section. While this ad-hoc approach is not chaotic, since it tends to follow particular strategies, it is certainly not linear as life-cycle models would imply. What the life-cycle partitions do provide is a characterization of the types of specifications that emerge from the design process and a framework for describing the more basic design activities (as certain of those activities predominate certain portions of the life-cycle).

With its chronologically ordered events, the life-cycle model of design, particularly one imbedded within a product life-cycle, is an effective tool for understanding the administratively oriented aspects of design evolution.

With it, we can understand the support DKMS must provide for the project administrator to establish his strategic plans and schedules based on prior experience with similar design situations.

Depending upon the level of application within the solution artifact, the purpose of conceptual design is primarily the structuring of the problem, either the discovery or analysis of requirements and the identification of the boundaries of the solution space [Friel 88]. The purpose of preliminary design is to separate the promising from the unlikely solutions. Ultimately, the objective of the designer is to select solution approaches to the problems posed that address those problems in an optimal fashion. In the detailed design mode, designers undertake three major tasks [Ramey 83]:

a. Identify and define the interfaces between system elements,

b. Separate out subsystems (those elements requiring yet more structural definition) from components (those elements requiring no further structural definition to be realized), and

c. Detail the characteristics that will govern the realization of components.

This mode is called detailed design and it is the detailing of component definitions that is at its heart. The mode has nothing to do with its placement in time--that is, preliminary design is not a "sketchy detailed design" and vice versa. The detailed design mode is focused sharply on identification of the detailed characteristics of components to the end that those components may be reliably realized by a supplier.

This life-cycle view does provide insight into support required by the DKMS for large-scale system design that involves groups of individuals. The way a system design effort arrives at its specification of a system is as an iterative process of developing more and more detail. Higher-level system requirements can be viewed as giving rise to subsystem requirements, which in turn, give rise to still lower-level subsystem requirements, and so forth. The mechanism for this involves several of the technical modes of thought that a system designer goes through. To begin with, the designer defines requirements to which a system element is to respond. In the preliminary-design mode, the designer considers various alternatives by which these requirements may be met and identifies logical partitionings of the element that will satisfy different functional aspects of the requirements. These allocated to particular technology disciplines where they may be realized. Finally, the designer enters the detailed-

design mode, and may determine that a particular element cannot be implemented directly but requires further definition.

At this point, a new subsystem is defined, allowing the definition of an independent design effort. Thinking as a requirements definer, the system designer determines which system requirements are appropriate to be imposed upon the new subsystem. The designer then defines additional requirements that reflect the decisions leading to the identification of this new subsystem. Both the functionality expected of this subsystem and the degree to which it is expected to contribute to the resolution of other system requirements must be shown. These requirements are then imposed on the new design effort, where they are combined with requirements or limitations that are specific to the discipline in which development will occur.

Since requirements imposed on the new subsystem originate from several sources, there are bound to be some conflicts. Consequently, the subsystem design effort must carefully analyze and compare these requirements. Specifically, the requirements must be checked for consistency and completeness. Any conflicts or voids identified during this requirements assessment call for negotiation between personnel from the next-higher level effort that imposed the requirements and the new design effort that will be held responsible for meeting them. All of this implies different types of problem structuring efforts for the subsystem level design than for the system level design.

This kind of negotiation can also be triggered by a number of other issues. For instance, a requirement imposed on a subsystem might lead to a design that would have a system element use resources over which that element had no control. Similarly, a design might call for a failure to be generated (or detected) in one system element that had to be resolved in another system element. Other problems could be technology costs that prove to be too high or byproducts unexpectedly produced in a subsystem that it is not equipped to handle.

In all of these cases, the requirements causing the problem must be fed back to the next-higher level in the system hierarchy. There, the problem is resolved or passed back yet another level. Generally, the system or subsystem effort responsible for resolving the problem will do so by imposing additional requirements on one or more of its subordinate subsystem development efforts. If these additional requirements create new problems for the development effort they are imposed upon, negotiations to iron out the problems resume. This requirements

assessment process tends to be an ongoing activity and continues until all subsystem efforts have a sufficiently workable set of requirements upon which they can base their design activities. Of course, this iteration is never in real life as simple as it appears on paper. The definition of subsystems and subsystem boundaries is an experimentation process with each designer discovering the boundaries to be finally imposed.

## 4.3. Generic activities of design

In the Phase I effort, we have attempted to abstract away the structure influences of the life-cycle model to identify the cognitive primitives (generic activities) of design. This results in what Ramey refers to as "patterns" of behavior. All such patterns would be present to some extent in each life-cycle mode. However, certain patterns would predominate in each mode. For example, the following are predominant generic activities in preliminary design [Serrano 88]:

1) Generation of plausible design alternatives (primarily based on historical precedence [Ramey 83, Friel 88]).

2) Identification and exploration of the boundaries of the design space (principally the identification of constraints).

3) Evaluation of the global performance of the alternatives to select the most appropriate ones for detailed analysis and refinement. If a candidate is found to violate a set of the constraints, the following actions can be taken:

   a) Alter the candidate solution,

   b) Try a different candidate solution,

   c) Change the original design specifications.

The set of generic design functions identified in Phase I to be relevant to the DKMS design are [Goel 89, Friel 88, Ramey 83]:

1) Extensive problem structuring--partitioning, decomposition, establishment of minimal conditions and constraints.

2) Fitting of prestructured approaches--use of knowledge of previous problem structures or solution approaches to force structure into

the problem and to provide a means of understanding how the current situation maps into previous situations.

3) Performance modeling--use of mathematical idealizations designed to reliably predict the performance (steady state or dynamic) of a proposed solution or of the problem situation and its environment to assist in the understanding of each.

4) Prototyping--building artifacts which mimic the problem situation or a proposed solution in certain ways in order to evoke information / decisions from the domain experts or to demonstrate feasibility of a concept.

5) Personalized stopping rules--completion criteria for level to level decision making, component identification, and deciding on the completeness of a design effort.

6) Limited commitment strategy--the use of multiple contexts for decisions allowing decisions to be reversed by elimination of the context.

7) Minimal decision making strategy--an optimal decision-making strategy is one that leaves open the maximum number of alternatives as the result of each decision.

8) Initiation and propagation of commitments and constraints discovered throughout the design process.

9) Solution decomposition into "leaky" modules--toleration of the delay in specification of the components that form the interfaces between proposed modules of a solution.

10) Hierarchies of idealizations--in general, sets of descriptive models that fit together into a hierarchy such that models lower in the hierarchy can represent all of the relations and objects of their parents as well as some additional relations. This allows the quick determination of how more detailed decisions might compromise the commitments made on more global issues.

11) Artificial symbol systems--special purpose (often graphical) languages for representing critical information that must be identified or managed during the design process.

12) Process driven reasoning--envisionment of courses of events, causality consequences, and enablement relations to determine if the design will "do the right thing."

Figure 5 below illustrates how these generic design activities often fit into a pattern of design prevalent in the conceptual and preliminary design modes.

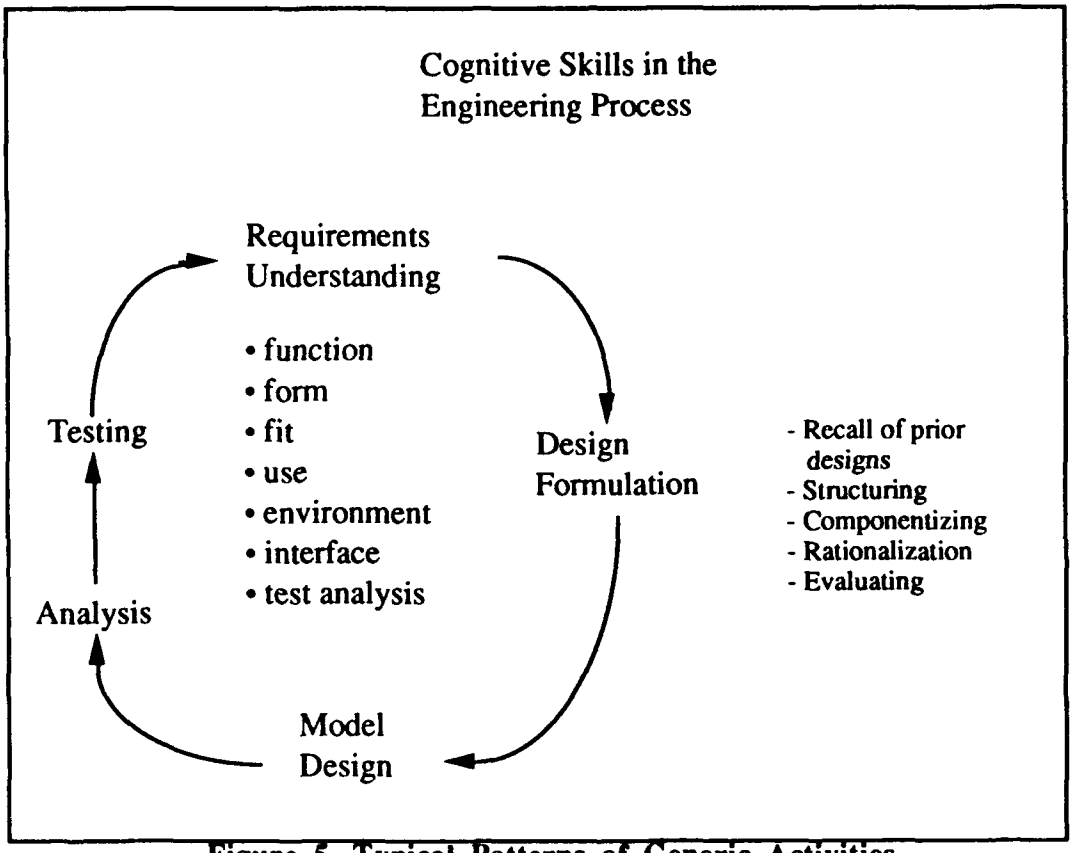



**Figure 5. Typical Patterns of Generic Activities**

Another way to understand how this collection of generic activities is structured by the human designer is to recognize that the process of designing systems is fundamentally a learning activity. Every stage in the process requires that the designer undergo a substantial amount of hypothesis generation, testing, and belief revision to attain the desired objectives. At first, the designer is involved in learning about the problem or problems to be solved and about the environment into which the solution must fit. In the development of the solution, the designer is using models, prototypes, and lab results to learn about how the problem fits into the design space. As a solution begins to emerge, the designer is involved in learning about the technology available for that solution and about the ways various technologies can be juxtaposed to solve the problem.

The supplier (manufacturer) of an artifact has to learn not only about the product that has been designed, but also about the characteristics of materials and processes that may be used in realizing that design. Finally, the learning process comes full cycle, as the designer becomes involved in evaluating problems that may arise because of, or in connection with, the new system when it becomes operational. From the DKMS point of view, a primary role of the knowledge management facilities is to accelerate the learning process as well as capture its results throughout the product life cycle and to be able to distribute those results throughout the life cycle.

### 4.3.1. Minimal decision making and least commitment strategies in design

The notion of minimal decision making is fundamental to the capability of being a competent designer. Because the designer is constantly placed in a position of uncertainty with regard to possible learning, the best course of action in decision making is to delay decisions as long as is reasonable. An optimal decision-making strategy in this environment is one that leaves open the maximum number of alternatives as the result of each decision. These are termed minimal decisions, consistent with the notion of least commitment introduced by Goel [89]. Using the practice of minimal decision making, the decision maker evaluates each decision, separating those that must be made from those that can be delayed. For those that must be made at this time, the alternatives are determined and grouped into those having harmful side effects if they are not selected and those with essentially benign side effects. Finally, those alternatives that cannot be avoided are evaluated in terms of their costs, benefits, and the degree to which they leave other options open. All things considered, this latter criterion is usually taken as the driver of the decision-making effort [Ramey 83].

Often, this principle of minimal decision making runs directly counter to most management practices. Traditionally, managers have learned that if a decision can be made, it should be made. In a business environment, this not only creates an appearance of decisiveness, but also leads to quick and orderly commitments, often to considerable competitive advantage. However, in the design environment, such convenient decisions can be extremely risky. A decision made now, whether it is needed or not, will almost always preclude the making of other decisions in the future. Often this is precisely why a decision is made, to reduce the size of the design space. However, experienced designers are aware that, in the design environment, very few decisions can be made with certainty in the

outcome. In general, decisions that can only be made with uncertainty early in the project, can be made with much greater certainty later. If the decision doesn't have to be made, it should not be made. In this way, the decision maker with the most information, the person with the best perspective on the problem, is allowed to make the decision, and this point in time almost always occurs where the decision has to be made [Ramey 83].

### 4.3.2. Prototyping in design

Figure 6 illustrates the relation of prototyping to the other generic activities in the design process. Prototyping is one of the core generic activities in the design of systems because of either the weakness, unavailability, or computational complexity of analytic performance models.

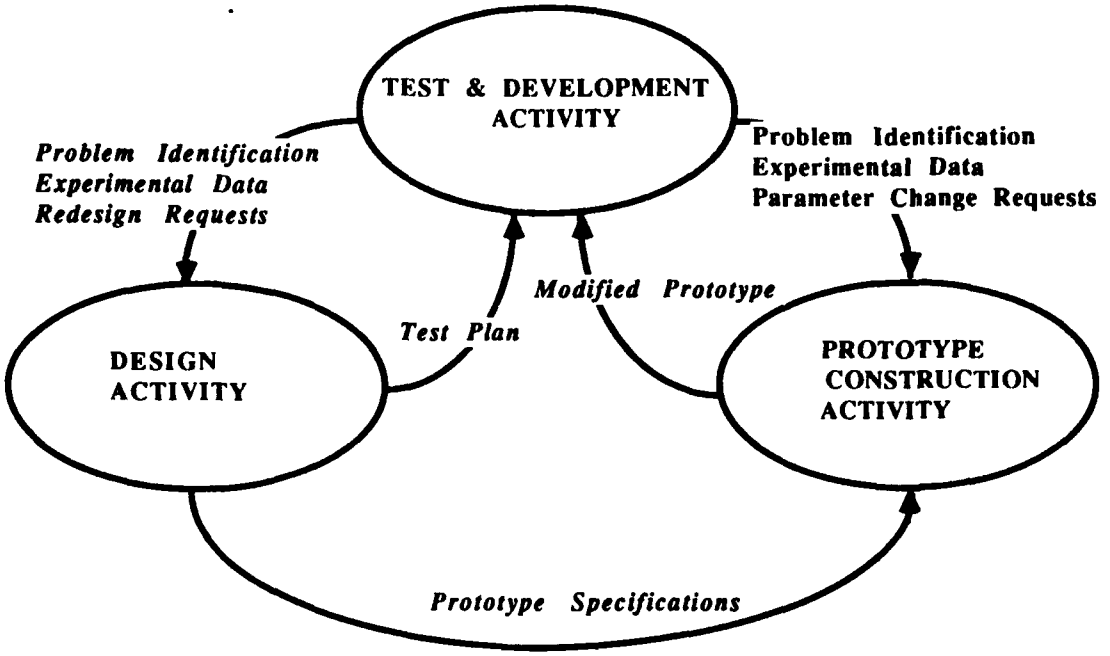



**Figure 6. Role of Prototyping in Engineering Design**

Prototypes serve to approximate aspects of the desired system. They are used to evaluate some aspect about which the designer is unsure by ignoring temporarily other aspects. There are basically three kinds of prototyping efforts relevant to design:

1) Requirements prototypes--both demonstration and feasibility prototypes designed to assist in the structuring or understanding of the problem and its environment.

2) Technology prototypes--including proof of engineering principles, proof of design maturity, and proof of production prototypes used to assist in the solution generation, evaluation, and testing of the artifact specification itself.

3) Research prototypes--prototypes built for the purpose of acquiring performance data for engineering models and design planning support.

Because design prototypes are developed in the same fashion as any other system element, the DKMS can support this phase of the design process. Almost all of the concurrent engineering phenomena related to the product are evident in the prototype life cycle. Requirements and specifications are determined for them; they are constructed, integrated, and tested, and finally they are installed and operated. Many prototypes are not maintained; however, some prototypes, particularly research prototypes, are operated over many years [Friel 88].

Requirements Prototypes

Requirements prototypes are developed to establish a firm understanding of the requirements to be imposed on a system. These prototypes are rapidly developed mini-systems that implement certain important aspects of the system as understood and ignore many other aspects.

Technology Prototypes

Technology prototypes are prototypes developed for the purpose of evaluating the suitability of technology to solve some specific problem. Like requirements prototypes, these are systems implementing certain aspects of the ultimate system and ignoring many others. Unlike requirements prototypes, technology prototypes may be major development undertakings in and of themselves. The traditional engineering-design model (EDM) is a type of technology prototype as is the mock-up. These prototypes demonstrate some aspects of the system, such as functionality in the former case and physical structure in the latter, at the expense of other aspects.

Research Prototypes

Research prototypes can vary from experimental apparatus designed to enable the measurement of some basic parameters (e.g. the viscosity of a new freon substitute) or complete end item systems (preproduction units) specially instrumented for test stand operation. Unlike technology prototypes, research prototypes generally are tied to the detailed design

phase of the product development and have a very focused role. They are used to generate the engineering data required for the final artifact specifications.

In general, the contribution that the DKMS can make in the prototyping area is reducing the iteration time between the activities shown in Figure 6. By reducing the time required to develop the engineering models, the requirements for various prototypes can be developed earlier. By application of the feature extraction and process planning capabilities of the DKMS, the production data for the construction of the prototypes (including NC code generation) can be automatically developed, shortening the lead time for prototype development.

### 4.3.3. Engineering reasoning mechanisms

Underlying the above generic functions of design are a collection of mechanisms for extracting additional information from the information that the designer has assembled as well as from his personal knowledge-base. These mechanisms form the basis for decision making and rationale generation throughout the design life-cycle. The following are the critical methods necessary to support the DKMS mission, in relative order of their *importance:*

1) Abduction

2) Production application

3) Belief maintenance

4) Constraint management

5) Constraint propagation

6) Qualitative reasoning based on envisionment or simulation

7) Induction

8) Deduction.

An important p' int first made by Ramey [83] and Friel [88] and substantiated by all of our experience to date, is the importance of abduction and induction over deduction in the design process. The designer does not deduce the proper system structure from the requirements, but

rather uses induction (or abduction) to arrive at a structure that can be demonstrated to be appropriate. In other words, the designer makes an educated guess based on experience and then uses data (or rationalized explanation) to support or discredit these beliefs. The solution is not a necessary one in the sense that many other solutions exist, but it is an adequate one. Within the limitations of schedule and budget constraints, the designer can demonstrate its optimality with regard to design criteria. In reaching this point of knowledge, the designer is guided by an understanding of the options available. To a large extent, this is driven by an ability to understand other systems of which he has some prior experience or knowledge as approximations of the system being developed.

Given the focus of the Phase I effort, another important relation to note is that the abductive (or inductive or production rule) reasoning processes that the designer goes through with respect to geometric data clearly are not at the points and lines level, but at the feature level. This holds for reasoning related to the model development as well as reasoning relative to the product design. In traditional CAD systems, it is impossible to directly associate features with points and lines. This observation is a primary motivation for the feature based CAD interfaces and the GCSG components of the high productivity CAD element of the DKMS.

## 4.4. Predominant design strategies

Design strategies can be considered as "meta-plans" for dealing with the complexities of frequently occurring design situations. They can be viewed as methodizations or organizations of the primitive design activities identified above. The three types of design strategies considered within the Phase I effort included:

1) External-constraint driven design--design carried out under situations where the goals, intentions, and requirements are not even characterized well, much less defined. These situations often result when the designer is brought into the product development process too early.
2) Characteristic driven design--design in a closely controlled situation where strict accountability and proof of adequacy are rigidly enforced. These design situations often involve potentially life threatening situations.

3) Carry over driven design (sometimes referred to as "routine" design).

From a DKMS point of view, the external constraint driven and carry over driven strategies are the most important because it is our experience that they characterize over 95% of design today. There are definite differences in the types of support and knowledge needed for these two strategies. For example, external-constraint-driven strategies rely heavily on working, observable prototypes and mock-ups because these allow the customer to crystalize an understanding of actual goals and objectives. Engineers using carry over driven strategies would prefer to, and in better design organizations, do rely on the use of predictive analytical models as a means of design evaluation.

## 4.5. Types of design knowledge

From the analysis of the life-cycle and generic activities views presented above and our experience with the construction of knowledge based systems for mechanical system design, the following classes of knowledge are evident in the practice of system design:

1) Knowledge of basic principles.

2) Knowledge of the general design process in a domain.

3) Knowledge of available components.

4) Knowledge of previous solution approaches.

5) Knowledge of available engineering performance models and workable modeling approaches.

6) Knowledge of test capabilities and results (e.g. what sorts of experimental data can be affordably, reliably, or physically acquired).

7) Knowledge of the human network (i.e. where is the knowledge and information in the organization or in professional associations).

8) Knowledge of the requirements, design goals, design decision / evaluation process, and design environment of the current problem,

9) Knowledge of political or governmental constraints.

An effective and comprehensive knowledge based design support system must extend current capabilities on several fronts. First, an effective design optimization aid must incorporate at least three distinct types of design knowledge. The first type involves knowledge of the process of engineering design that is, the iterative process of conceptualization, analysis, design specification, prototype construction and testing, and design refinement as displayed previously in Figure 6. The second involves knowledge of the life cycle of the product itself as it evolves from an operational need through an engineering concept to a manufactured reality into field service and repair / refurbishing. The need for both types of knowledge is evident when one considers the activities and the knowledge sources that a designer uses. A designer going through the iterative process of design is constantly drawing on a wealth of product life cycle information gleaned from experience. Test results on the design currently being generated are evaluated in the light of "similar" historical cases. A designer may, for instance, remember on observing certain characteristics of a design that a similar design had certain problems in maintenance. Thus a tool to support the generation of more optimized designs in terms of manufacturability, reliability, survivability, and maintainability must incorporate support for the design cycle activities and a store of knowledge of product life cycles. The third type involves knowledge of how to design effective performance prediction models in the product domain. Our experience in development of modeling support environments has been that usable models are rarely developed from first principles (basic laws of nature). Expert modelers bring experience in test data acquisition, computational limitations, required accuracy, and previous models developed by them or theirpeers to bear along with the knowledge of first principles to develop new or modify old models.

In the DKMS, given the current state of the art of knowledge representation, we must distinguish between the capture of knowledge (design, manufacturing, or maintenance experience or principles) that is to be passed on as applicable knowledge (e.g., a representation such as a container object representation or a shape prototype based production rule is actually engineered so that it can be applied at a later time) and design rationale which must be captured on the fly and is suitable for browsing but not necessarily application. Both types of knowledge can contribute to the delivery of concurrent engineering support. With the first type, we can actually automate portions of the design decision process. With the second, we hope to leverage the lessons learned throughout the product life cycle to allow designers to operate under the influence of this accumulated experience.

## 4.6. Role of shape in design cognition

The physical world around us is a rich source of constraints for designers, a kind of constraint they are often attempting to overcome and yet the kind they first rely on to partition and control the design evolution. Different design disciplines treat shape and form differently. To the architect, shape is a means for expression of themes. To the microelectronic device designer, characteristics of the physical shape such as size of die or size of intercomponent spacing are naturally provided constraints that must be managed. To the mechanical system designer, shape is often synonymous with function. Our contention is that much of a designer's rationale is based on shape and form-related considerations. This is not to say that materials, processes, and environmental constraints are not also important considerations. A primary goal of the design activity is the production of specifications and a major aspect of such specifications is the specification of shape and form.

In our Phase I effort, we analyzed the role of shape based reasoning and discovered critical relationships between shape concepting and both problem partitioning and solution structuring. When a person describes a natural shape, one characterization of the involved cognitive processes consists of the following four generic activities applied recursively:

1) Partitioning--breaking up the object into separable elements.

2) Classification--categorization of an element with a prototype.

3) Deformation --local / global topology changes to a prototype.

4) Arrangement--description of relationships between the elements.

The term "recursive application" implies that the same process of shape description is normally applied to each element of the partitioning. This recursion appears to continue until the prototype classifications of the resulting elements can be clearly established. When a person forms concepts of shapes, many of the same cognitive activities are involved, only slightly rearranged. In the design situation, the shape concepting process appears to proceed recursively with the following steps:

1) Use "function to prototype" knowledge to establish a general shape.

2) Use partitioning to separate out elements for further specification.

3) Use functional relations to suggest arrangements.

4) Use deformations to accommodate constraints of physical form.

5) Use arrangements to build up the whole from the parts.

6) Use shape description to generate the resulting specifications.

The discovery of this process organization led to some important insights relative to the capabilities required of the form feature CAD interface component of the DKMS. For one, traditional CSG systems operate on the assumption that the elemental pieces will be defined first and then composed into the whole. In fact, it appears that this strategy for user interaction only works if you have an image of the object in front of you and can develop a partitioning strategy that can be accommodated by the CSG system. Since current CAD systems were designed to replace or augment the draftsman function (which is merely creating the artifact specification), this style of interface is marginally acceptable. However, to deliver concurrent engineering knowledge support to the designer, one must develop a geometry concepting environment that does not assume that the decisions have been made before it is engaged. We believe we have captured the essence of an interface and a geometry engine that will provide such design concepting support.

We believe that the close parallelism between the cognitive activities of design and those of shape concepting and shape description argue for a central focus on a powerful shape concepting and manipulation component of the DKMS. Rather than treating the geometry generation element merely as a necessary element for artifact specification support, we believe it is a critical component in the capture of design knowledge, delivery of design advice, and support for the human design cognition process.

## 4.7. Characterization of design rationale / intent

In the design of a DKMS, it is important to define and contrast design rationale, design specifications, and design process history. Design specifications describe **what** should be realized in the physical artifact. Design rationale describes **why** the design is the way it is and **how** the specified artifact is intended to work. Included in this design rationale are the principles and philosophy of operation as well as models of correct behavior including models of how the artifact is intended to behave as it

fails. The design process history records the steps taken, the plans and expectations that led to these steps, and the results of each step.

One of the problems with the capture of design rationale is that it requires the statement of characteristics beyond the minimum specifications required to produce the product. Since the major goal of design has traditionally been the construction of specifications for artifacts so complete that any realization of them will satisfy the requirements (and thereby solve the problems), specification of the underlying logic of the decisions that contributed to, led to, or resulted in such a design description are not naturally recorded. After all, their inclusion into the traditional document structures used to record the design artifacts may cause confusion or at best complicate the acquisition / interpretation of the critical information communicated by these artifacts. In addition, as noted in Friel [88], Goel [89], a designer may make hundreds of focused component decisions or through interpretation of test results, implicitly make thousands of configuration decisions in a very short time. Lack of efficient methods for the capture and representation of these decision considerations is a primary impediment to the capture of design rationale.

In our investigation into solutions to this problem, we have characterized both types of design rationale and mechanisms for representation of these types. The core types of design rationale identified are:

1) Philosophy of a design including:

   a) Process descriptions of intended system operation, and

   b) Design themes in terms of object or relation types.

2) Design limitations expressed as range restrictions on system parameters or environmental factors.

3) Factors considered in tradeoff decisions.

4) Design goals expressed in terms of:

   a) Use or lack of use of particular components,

   b) Priorities on problems requirements,

   c) Product life-cycle characteristics (e.g. disposable versus maintainable),

d) Design rules followed in problem or solution space partitioning, test/model data interpretation, or system structuring.

5) Precedence or historical proof of viability.

6) Legislative, social, professional society, business, or personal evaluation factors or constraints.

Possibly due to the common carry-over strategy or the complexity of design rationale expression, the most frequent rationale given for a design is that it was the design that worked last year. Without making judgment on this situation, a minimum requirement for a design knowledge management capability is that it must be able to record historical precedence, as well as statements of beliefs and rationalizations for why a current design situation is identical to the one the previous design serviced.

Moving beyond text capture and association for the support of capture of design rationale becomes quite complex. In Phase I we approached this complexity by examining the typical questions a user might make of a design knowledge base. The information required to answer these questions then become requirements for the DKMS to capture and manage. We partition these questions into several types: (1) Questions about the specified artifacts' composition and structure, (2) Questions about object's purpose or function relative to the intended behavior of the artifacts, (3) Questions about causality / enablement characteristics of the established relations between individual objects or subsystems, (4) Questions about supportability of particular beliefs used as rationale for design decisions, (5) Questions about the design process, how it was planned, and how it was carried out, and (6) Questions about device behavior or failure (the proverbial "What if ..?" questions).

In general, our approach uses several strategies based on a situation theory structured description of the evolving design and the design situation (objects standing in relations in situations that stand in planned involvement relations with other situations). The questions concerning the composition and structure are answered through information access of the design bill of materials. The key to questions about purpose and function is the recognition of the difference between purpose and function. At an initial level function can be explained by knowing the dependence or independence of an object in the device relative to the environment of operation of the device [Forbus 84, De Kleer 83]. At a more detailed level, knowledge of inter- and intra-state phenomena is required. Questions about

purpose require knowledge of the events in the planned operation of the device and then an association of an object's existence with its role in those events. Supportability questions can be answered by collecting and displaying the set of support from problem and first principle premises (or assumptions) to the fact in question [Hobbs 86]. In this area a major role is played by engineering discipline specific "thematic abstraction units" (commonly agreed upon assumptions) to establish grounding conditions for many assumptions [Dyer 83]. Questions about the design process itself can be addressed with the use of qualitative simulation and planning techniques as in Wilensky [83]. Finally, the answer of "What if" questions appears to be relegated to simulation (either qualitative or quantitative) [Kuipers 84, Laughton 85].

## 5. Design Knowledge Organization and Retrieval

The DKMS must accommodate multiple forms of knowledge representation. In this section we describe some of the general characteristics of how such knowledge would be treated by the overall knowledge manager and then describe two specific forms of shape indexed knowledge and container object knowledge representation methods developed as a part of the Phase I effort.

Relative to the knowledge management functions of the DKMS, the issue of knowledge capture, storage, and retrieval is primarily one of organization and retrieval as distinct from representation and matching. Organization differs from representation in that it is focused on structuring of knowledge and information together to support use by multiple processes. Previous efforts in this area have traditionally focused on attempting to add flexibility to a base representational scheme and interfaces to traditional data base systems. Generally, the result is a representational scheme that is inefficient or ineffective for all but the class of problems addressed by the original base representation. Retrieval is distinguished from matching since matching is the dominant process required for support of the reasoning methods described above. Retrieval, on the other hand, has as its domain the management of memory use and object indexing. Retrieval is concerned with locating and retrieving collections of knowledge units that possibly contain the structures needed by the reasoning process. To the retrieval process, prototypes, matchers, and candidate objects are all treated equally. In other words, the DKMS retrieval resources can be used by a reasoning process to support its matching of prototypes against candidate objects, in which case the role of the retrieval mechanism is to produce as small as possible set of candidate objects that potentially fit the

matching specification of the reasoning mechanism. Similarly, the retrieval mechanism in support of a design decision support process could return a reasoning mechanism (i.e., a collection of patterns and a matcher) to support a particular phase of the decision support activity.

The general form of knowledge organization in the DKMS is independent of the underlying knowledge representation. It is based on a treatment of each element of knowledge as an independent object with associated descriptions and procedures. We have employed a situation theoretic model for our description structure and demonstrated the effectiveness of this model to organize first order predicate language and production-language-based knowledge (two of the major forms of declarative knowledge found in expert systems for design automation). We are experimenting with the Air Force IISyCL constraint language [IISyCL 89] for procedure representation as the form of these constraints allows for the representation of both the triggers and conditions for execution as well as alternately the execution specifications or a reference to a host language (C, Ada, Fortran, LISP etc.) module.

Thus the proposed DKMS knowledge management facilities will operate on a declaration and definition of the knowledge that exists in the system much in the same manner that the ISO multi-schema architectures provide integration services through a definition of the common information. An overview of the architecture and processing operation of this approach is provided in the DKMS Platform Architecture section of this report. This approach allows us to use both simple key indexing of knowledge units that can easily be derived from the logical structure of the knowledge as well as associative links based on patterns or contexts of knowledge use.

The remainder of this section will focus on three new knowledge representation and reasoning methods developed to fill critical voids in the capture and delivery of design knowledge and product experiences within a concurrent engineering environment.

## 5.1. Shape based knowledge representation and reasoning

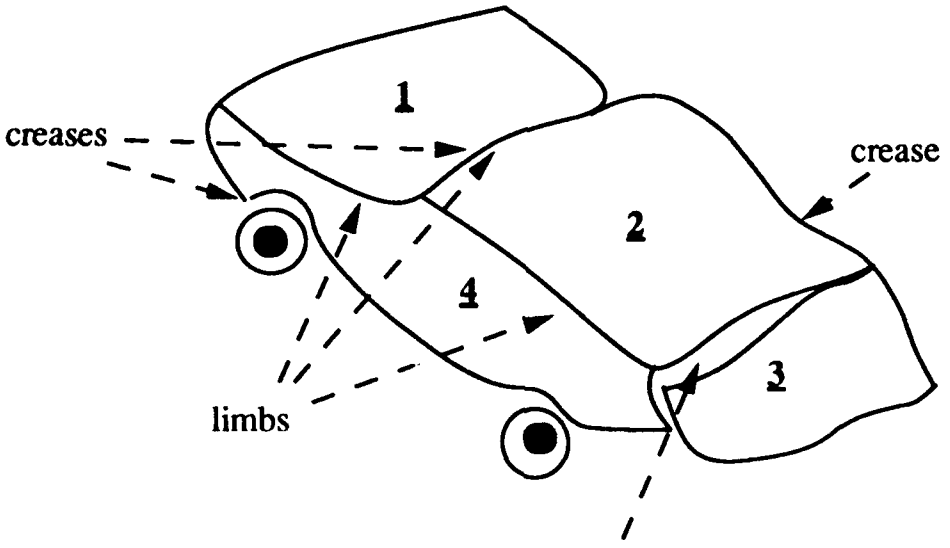
Shape Matching is an important issue in Unified Life Cycle Engineering (ULCE). The research results addressed in this section relate to two important aspects of this problem: (1) finding of potential problem regions and (2) matching the geometrical model of a potential problem region with problem shapes described by manufacturing experts and stored in a product life cycle experience knowledge base.

We propose a method using object descriptions in terms of their surfaces. The surface of an object is described by segmentation into surface patches and the complete description consists of each patch's normal and their interrelationships. The partition and description of the surface is based on measured curvature properties of the surface. We believed that our chosen representation has many advantages for potential problem shape finding and shape-based associative matching for retrieval of life cycle experience from the knowledge base. It also enables us to construct alternative shapes on the part design as advice (or work-arounds) on potential problems [Ting-jung 89].

As previously mentioned, we have chosen partitioned surface descriptions for representing objects in the potential problem matching work. The automatic identification of potential problem shapes has not been addressed in the research community to date. The difficulty of performing this task automatically is similar to the difficulty of performing it manually. That is, how can you tell designers to avoid a shape they don't know about. Since these shapes can result from the interaction of more than one part, they tend to evolve as the design evolves. We are proposing an approach that is a variation on the theme "common things occur commonly." More specifically, we are proposing to break up complex shapes into component shapes by examination of "unexpected" changes in the shape definition. The discontinuities (breaking surfaces), creases (radical change in curvature), and limbs (connection curve between surfaces) are chosen to be the factors for partitioning surfaces as they are explicit shape determiners of the surfaces of 3-D objects. These factors are also used for localizing the region for potential problem shape checking.

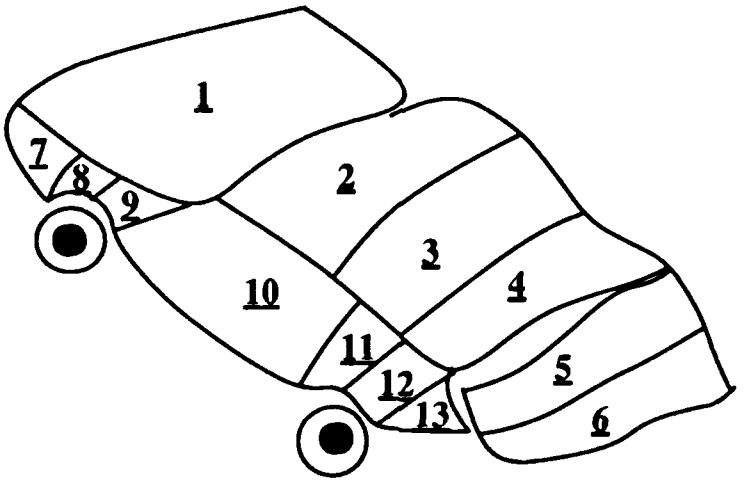
In the first stage (Figure 7), the object represented with surfaces is partitioned at discontinuities which can be detected by examining (1) the zero-crossings, (2) limbs which are formed by the boundary of two connected surfaces, and (3) creases which can be found by evaluating the extremal values of surface curvature. These detected discontinuities, limbs, and creases are then used to partition a complex surface into patches (Figure 8). These patches can be subsequently approximated by planar facets. The region which holds the property of discontinuity, limb, or crease would be localized for problem shape checking (Figure 9).

**Example :**



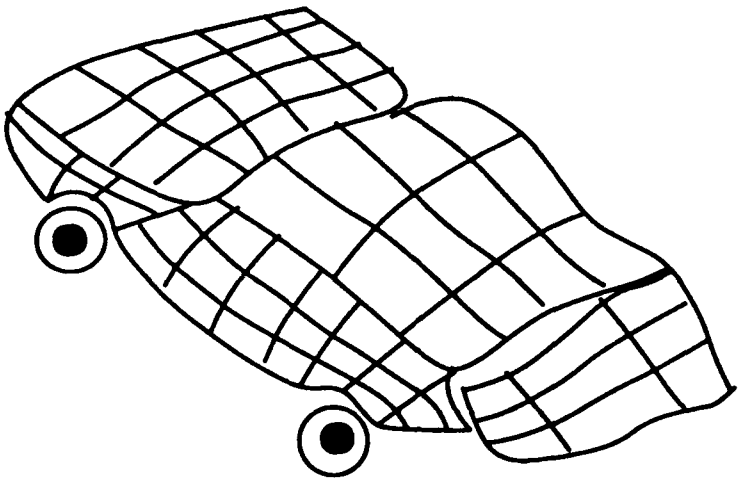

**Surface Model**

**Figure 7. Characteristics of Problem Shapes**




**Surface Partition**

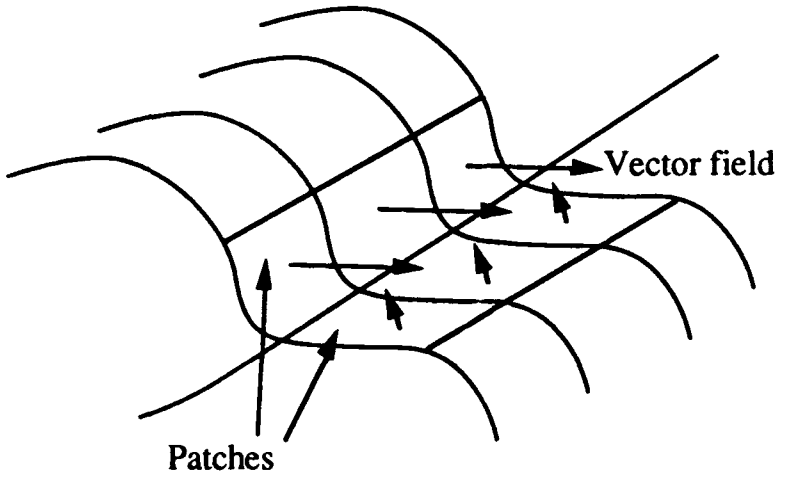
**Figure 8. Problem Shape Analysis**



Patch Generation

**Figure 9. Problem Isolation**

After the stage of patch generation, the system would query the designer about the discontinuity situation (if it exists). If it is a design error, the designer can request the system to reconnect the broken surface to its neighboring surfaces using a surface smoothing algorithm. Along the boundary curve between two surfaces, the vector field formed by the normal vectors of patches would be checked against the vector field of the problem shapes in the database (see Figure 10). If, using the isolated shape to index into the DKMS knowledge base, the system determines a potential problem shape has been detected, the system will then report the problem to the designer and provide a suggested modification.




**The vector field formation along the boundary curve between two surfaces.**

**Figure 10. Boundary Curve Traversal for Shape Recovery**

This potential problem shape detection method can be described algorithmically as follows :

### 5.1.1. Algorithm for shape extraction

Given a set of surfaces (planar or sculptured) which represent an object.

1. Compute curvature for each sculptured surface and find the extremal of the curvature.

Note: for a general space curve **r** = **r(t)**, the curvature **k** is obtained by differentiating r(t) twice. Thus

$$\dot{\mathbf{r}} = \dot{s}\mathbf{T} \text{ and } \ddot{\mathbf{r}} = \ddot{s}\mathbf{T} + \dot{s}^2\mathbf{k}\,\mathbf{N}$$

where **s** is the arc length and **T** is the unit tangent vector.

For a curve **u** = **u**(t) on the surface **r** = **r**(u,v) the curvature can be obtained from the following equation :

$$\ddot{\mathbf{r}} = \ddot{s}\mathbf{T} + \dot{s}^2\mathbf{k}\,\mathbf{N} = \frac{\partial^2 r}{\partial u^2}\dot{u}^2 + 2\frac{\partial^2 r}{\partial u \partial v}\dot{u}\dot{v} + \frac{\partial^2 r}{\partial v^2}\dot{v}^2 + \frac{\partial r}{\partial u}\ddot{u} + \frac{\partial r}{\partial v}\ddot{v}$$

If the extremal of the curvature is greater than a predefined tolerance, the surface is then partitioned along the extremal curve.

For each subsurface, recursively subdivide the surface until the extremal of curvatures of all subsurfaces is smaller than the predefined value.

2. Check the discontinuities among surfaces using crossing zero method. If discontinuity has been detected, query the designer for reconnection.

3. Generate patches for each subsurface, finer (more) patches are generated along the surface boundary.

4. Collect normal vectors of the patches to form vector fields along the boundary among surfaces.

5. Check the vector field against the vector field of potential problem feature stored in the knowledge base. If similar vector field is detected, report the potential problem to the designer and provide a suggestion for modification.

## 5.2. Feature based associative retrieval

The understanding of named shapes (features) is an important link in the unification of the design, manufacturing, and maintenance components of any Unified Life Cycle Engineering (ULCE) support environment. Global understanding of shape and form comes from the ability to characterize a part in terms of its constituent features. The constituent features may be specific to design or manufacturing processes. Because of the necessity of being able to associatively retrieve manufacturing and design information based on form features, it has become imperative that the major thrust in research in this area should be in feature recognition and in feature-oriented CAD systems.

Research in the feature recognition area to date has been pushed by research in process planning. The most promising techniques tried to date include:

a) Volume Decomposition [Woo 1982]

b) Syntactic Pattern Recognition [Kypriano 1980, Jared 1985]

c) Logic Programming [Henderson 1985]

d) Topological models [De Floriani 1989]

e) Projection Classification [Keen 1989]

In this discussion, it is important to distinguish between Geometric and Topological modeling:

i) Topological modeling describes the relationships between primitive geometric elements.

ii) Geometric modeling defines the shape, orientation, and location of each topological element.

For a feature representation/extraction system to be useful in providing keys for database retrieval of parts, it should have the following properties:

a) The representation must be unique.

b) The representation should not be brittle, i.e., subtle variations in parts should not produce drastically different feature representations.

c) The representation must allow the definition of its own limits. The limits of the representation should be precisely defined, i.e., the system should be able to warn the user about topologies that it cannot classify.

Henderson uses a depth first search strategy on the boundary representation of the stock minus the part to pattern match features defined in terms of rules. On matching a feature, it is subtracted from the boundary representation of the material to be removed and asserted as a fact. The problem with this approach is that subtracting features from the material to be removed may render other features unrecognizable. Furthermore, this approach requires the definition of raw stock.

Syntactic pattern recognition was used for group technology coding by Kypriano and Jared at Cambridge University. It relies heavily on local information within the boundary representation, such as loops of convex or concave edges. It does not handle interfering features very well.

Volume decomposition involves searching for patterns in an object's Constructive Solid Geometry (CSG) representation by first subtracting the part from the convex hull of the part and then recursively subtracting the difference from the convex hull of the difference until only convex differences remain. It should be pointed out that this restricted CSG representations is unique, but in general CSG representations are not unique. While this approach can provide a unique representation of the features of an instance of a part, it is brittle. Because slight variations of the same part could produce radically different CSG trees, it is not useful as a key for database retrieval.

De Floriani uses a generalized edge-face graph to represent the topological information in the part. The feature recognition algorithm partitions the graph into biconnected and triconnected components that it uses for recognizing form features such as protrusions, depressions, through holes, handles, and bridges. The decomposition of the edge-face graph into biconnected and triconnected components is a directed acyclic graph, called an object decomposition graph. The object decomposition graph can be used for associative database retrieval of parts based on shape and form because it provides genus, shape, and feature keys in the patterns of biconnected and triconnected components. Parts with similar arrangements of features can be retrieved by resolving subgraph isomorphisms between the key decomposition subgraph and the decomposition graphs of parts in the database. It is possible, however, to define patterns for which no recognizer exists. For example, De Floriani's system cannot classify features that cross more than three faces.

As an example of this procedure, Figure 11b shows the decomposition of the part shown in Figure 11a into three triconnected components T1', T2', and T3', all having Face 1 and Face 2 {f1, f2} as a separation pair. Figure 11c shows the object decomposition of the edge face graph shown in Figure 11b, where it can clearly be seen that T1, T2, and T3 are adjacent components. T2 is a closed component as l1 and l2 are closed loop nodes, as shown in Figure 11a. T3 is a trivial component, and T1 is an open nontrivial component since the edge described by T3 does not belong to T1.

From our research in Phase I, it would appear that a topologically based form feature based representation is better suited for feature indexed knowledge base retrieval than CSG subgraph isomorphism.

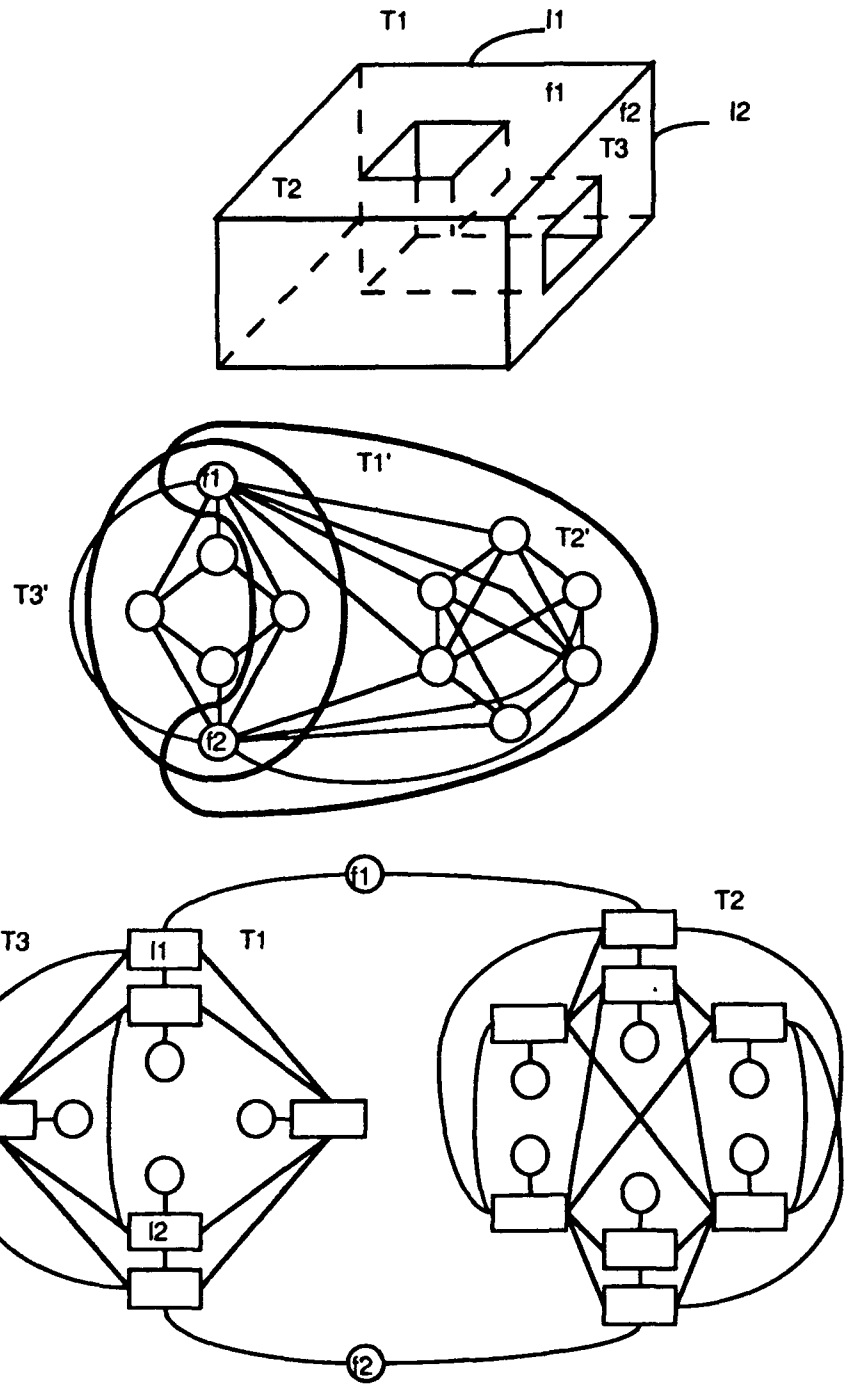



**Figure 11. Example of graph networks for topological form feature representation**

# 6. Container Objects

The purpose of this section is to demonstrate the applicability of composite objects as a knowledge representation technique for concepting and describing shape, form, and function within a framework of experimentally derived description primitives.

## 6.1. Basic concepts and theory of operation

A composite object oriented system extends the notion of objects to make them recursive under composition. This enables the instantiation of a group of objects as an entity. This is useful when relative relationships between members of the group must be isomorphic for distinct instances in the group. A composite is defined by a template that describes the sub-objects and their connections. These objects are created by an instantiation process and are describable in a class inheritance network. One benefit of creating composite object classes is the ability to make modified versions of a template by making a new subclass which inherits the properties of the super class. Facilities for making composite objects are not common in object-oriented languages, but are fairly common in application languages such as those for describing circuits and layout of computer hardware.

## 6.2. Principles for composite objects

Composite Objects have the following features:

- Composite Objects are specified by a class containing a description that indicates the classes of the parts and the interconnections among the parts. The use of a class makes instantiation uniform.

- Instantiation creates instances corresponding to all the parts in the description. The instantiation process must keep track of correspondence between the parts in the instantiated object. It fills in all the connections between objects. It must permit multiple distinct uses of identical parts.

- The instantiation process must be recursive to allow the use of composite objects as parts. For programming convenience, the system must either flag as an error the situation where a description specifies using a new instance of itself as a part or support "lazy instantiation." A description of a part which

includes itself can result in the instantiation of objects with an unbound size. Alternatively, instantiating subparts on demand (lazy instantiation) would allow the use of potentially unbounded objects as the subparts are generated only on use.

- It must be possible to specialize a description by adding new parts or substituting for existing parts. The description language must allow specialization of composite objects with a granularity of changes at the level of parts.

The following sections describe the basic requirements for the DKMS container-object system based on the above general theory of composite object operation.

## 6.3. Basic requirements for container objects

A container-object-oriented system uses structural templates to describe container objects having a fixed set of parts. Container objects are regular objects, created by an instantiation process, and describable in a class inheritance network. The benefit of creating container object classes is the ability to make modified versions of a template by making a new subclass which inherits the properties of the super class.

### 6.3.1. Container object templates

A container object is described by creating a class whose metaclass is a template or itself has a superior that is a template. Any class whose metaclass is a template or a subclass of a template is called a template. The default values of the instance variables of a template can point to other templates which will be treated as parameters and be recursively instantiated when the parent is instantiated. Instance variables that point to non-template classes or to default values are treated as constants that are inherited by the instances.

### 6.3.2. Instantiation of container object templates

The instantiation process will recurse on all the parameters of the template. Every parameter is instantiated when it is first encountered. Multiple references to the same parameter are replaced by pointers to the same instantiated instance. The instantiation of a template is isomorphic to the original template structure with constants inherited and distinct instances substituted for distinct templates (i.e., parameters). If a container needs

multiple distinct instances, then multiple distinct templates are needed in the description.

### 6.3.3. Specializing container objects

Specialization of container object templates must be supported both at the definitional level and at the instantiation of a template. That is, the template class must also be a template allowing the construction by example of new template structures.

### 6.3.4. Constrained, conditional, and iterative templates

The instantiation of container objects having repetitive or conditional parts with the logic specified with IISyCL constraints will be supported by the DKMS.

### 6.3.5. Defaulting of attributes values in container objects

When instantiating a container object, it would be advantageous for it to determine default attribute values that satisfy the instantiation constraints. For example, in ICAD to make a table, the user only has to design one leg and the table top and then specify that the table should be on top of four of the legs. ICAD automatically defaults the positions of the four legs to the corners of the table top.

### 6.3.6. Lazy instantiation and demand driven control of container objects

A container object could be very large. For example, an instance of an F-16 is composed of thousands of parts. Each part has geometric models, as well as design, test, manufacturing, assembly, maintenance, and vendor data attached to it. On referencing or creating the F-16 instance, it would be advantageous if the entire F-16 were not instantiated, rather only those components of the container that we care to reference and the set of support objects needed to instantiate those subcomponents so as to satisfy the instantiation constraints. Demand driven control means the system only performs those calculations that are necessary to respond to a query. This facilitates partial design and saves time during design revisions.

### 6.3.7. Graphic browser for examining objects

A hierarchical organizational structure manages the complexity in design by breaking down the assembly into its component parts. The designer

must be able to create, edit, instantiate, and modify container object templates using a graphic browser that displays user-selected structure forming relations. The default relations displayed would be the part-whole and inheritance relations.

### 6.3.8. Part-whole defaulting

When an object requires an attribute that does not appear in its definition, the object looks up the tree to its ancestors for the value. In ICAD, this mechanism is known as part-whole defaulting, as attributes are automatically made available as default values to parts within the subtree of the part owning the attribute. Thus, only the attributes of the descendents that are different from the ancestors need be defined. For example if a table's material-type attribute is defined to be wood, it would only be necessary to define the material type of those components of the table that are not made of wood such as nuts, screws, and bolts. For example, if we were to define a house with a red terracotta tile roof, it would be unacceptable if we would have to set a color attribute on each one of its tiles, while at the same time, we might want to be able to point to a tile and obtain its properties.

### 6.3.9. No class versus instance distinction

In many object oriented systems, a distinction is made between class and instance. Classes are generic objects or definition of a prototype, whereas instances are specific examples and instantiations of the generic class. Generally, classes are not used directly in a system, but are used to construct instances. In ICAD, for example, the design language is used to describe a class, which is the set of possible instances that may be created from a single definition, but class descriptions cannot be created by the instantiation process. In the DKMS container object system design, we are modeling our class structures after the schema structures in frame systems. All operation types applicable to instances will be executable on classes.

## 6.4. Geometry driven container object system

### 6.4.1. Container objects based on partition similarities

In fleshing out a description using prototypes or building up a description using generic primitives and operations, the process will be recursive and will involve partitionings, features, arrangements, deformations, and functions for specifying the components and their constraints, and

relationships. There is a high degree of interdependence between these terms. For example, a partition may be defined in terms of features, which are defined in terms of partitions, arrangements, functions, and deformations. A deformation may be described in terms of features.

### 6.4.2. Container object based partition similarities

The system will support the subdivision of the object in terms of features, arrangements, function, and deformation.

### 6.4.3. Container object based feature similarities

The system will support the use of predefined feature descriptors to describe the contents of a container object.

### 6.4.4. Container object based arrangement similarities

The system will support the description of the spatial / functional patterns that exist between the elements in a container object. ICAD has a set of predefined arrangement keywords, whereas LOOPS requires the writing of LISP code to control the instantiation of patterns of objects within a template. What is needed is the ability to define arrangement classes or abstractions within the framework of the container system.

### 6.4.5. Container object based deformation similarities

The system will support the specialization of prototypical descriptions by describing incremental departures from the prototype in terms of form (bending, tapering, pinching) or behavior. All object oriented languages allow the altering of behavior through the definition of class hierarchies and the definition of methods. Because LOOPS uses metaclasses to define composite templates, this allows the user to specialize the behavior of composites through the definition of methods. As CLOS supports the metaclass concept, we will be using a similar implementation strategy as LOOPS.

### 6.4.6. Container object based function similarities

The system will support the description of the contents of a container in terms of the specifications, environment, and purpose for which the components are intended. ICAD allows the definition of function through the addition LISP coded constraints defined either in the defpart macro, or externally using methods that operate directly on the instantiated objects.

## 7. High Productivity CAD Systems

To provide support for design engineers in the rapid development of product specifications and engineering prototypes, a design environment must:

1) Use a solid modeling representation and organization that is isomorphic to the representations used to perceive and structure the design situation. [Pentland 86a]

2) Support the rapid entry of a design concept or intent into the CAD system.

3) Intelligently set default dimensions, spatial orientation, and surface blends.

4) Allow inheritance and / or merging of design attributes between parts or components (e.g., blending, corner radii, etc.).

5) Manage and propagate constraints specified in the product needs analysis or in design goal specifications and enforce those constraints on the evolving solid models.

6) Relate the design attributes and designer rationale to the components of the evolving product definition.

7) Capture, store, and reason about an efficient representation of the designer's intent.

8) Transparently control the configuration of the design artifacts.

9) Support rapid browsing and modification of the above information about an evolving design.

The rationale for a GCSG to support rapid entry of part geometry and definition data goes beyond the traditional design automation needs. It is also critical to the development of knowledge-based concurrent engineering support systems. Our experience with the development of the manufacturing support portion (advanced process planning and NC code generation and verification) of AI CAD/CAM as well as with knowledge based systems that automate mechanical system designs is that no single geometry representation is sufficient for all the functions required by an integrated knowledge-based design support environment. It is also clear

that much of the design and manufacturing knowledge that must be delivered in such an environment is shape and form related. Without a means of rapidly acquiring the geometry component of this knowledge, the requisite knowledge bases cannot be constructed. Of particular interest is the rapid generation of parts which exhibit both irregular sculptured surfaces and prismatic or rotational components (as are common in die and mold components).

The design of efficient algorithms that will allow the transparent use of multiple geometry representations by both human users and geometric reasoners will lay the mathematical foundation for a unified implementation to support an intuitive and interactive design environment. To achieve the goal of ease of construction and manipulation of irregular sculptured surfaces, a collection of everywhere differentiable parametric objects, known as superquadrics, will be used as one of the base representations. Another base representation to be supported is the traditional BRep (Boundary Representation) and BSPT (Binary Space Partition Tree) for representing prismatic form and feature shapes. Efficient geometric algorithms for boolean operations on combinations of primitives in the various representations, as well as local and global deformations of the superquadrics, and surface blends must be investigated and initial development completed to pursue this goal. Furthermore, constraint propagation paradigms for interacting with the GCSG based modeler will be designed. This will allow the creation of an integrated, object-oriented front-end solid modeling system using superquadric primitives [Barr 81, 84] supported by a rich and interactive environment of orientation, deformation, boolean, and blending operations with intelligent defaulting handled via constraint management. Such an environment could support the user in the concurrent evolution of needs analysis, concept design, and feasibility analysis that will facilitate rapid prototyping and design of parts.

The primary thrusts and progress to date of the CAD interface are:

1) The design of efficient algorithms for superquadric boolean operations. This problem is not well reported in the literature. We have found that by taking advantage of the spherical product that generates a superquadric, the problem of solving the highly nonlinear 3-D surface intersection can be simplified to intersections in two 2-D spatial domains, from which the 3-D structure of the resulting solid can be reconstructed from the spherical product.

2) The design of efficient algorithms for the boolean operations between superquadric and BRep. We have found that the BSPT is well suited to the role of neutral representation between the superquadric and BRep. Our intended approach is as follows: The hyperplanes coincident to the BRep faces are used to encode both the BRep and the superquadrics into a BSPT representation. The BSPT provides a spatial ordering that will allow the sub-hyperplanes of the BRep to be efficiently classified against superquadric using the analytical form of its inside-out function. The resulting solid is a BSPT consisting of mixed superquadric and BRep.

3) The design of efficient algorithms for surface blending between superquadric surfaces and between superquadrics and BRep. With the intersection curve between the two solids and using the rolling ball paradigm, we can compute the path of the center of the rolling ball about the intersection curve and tangential to the surfaces divided by intersection curve. The spherical product can then be used to generate a solid representation of the blend surface that may be unioned to the target solid.

4) The design of efficient algorithms for performing local deformations on superquadrics. In our investigation of local deformations, we have found that there are many ways to achieve local deformations, but the choice of mathematical formalism used is strongly influenced by the user interface paradigms that are best suited for describing the local deformations.

5) The design of efficient display schemes for combined BRep, superquadric, and generalized cylinder part descriptions.

6) The development of representation schemes and inferencing mechanisms for handling design knowledge using the GCSG capabilities. We have experimented with the use of (1) Situation Based Reasoning, (2) Truth Maintenance, (3) Constraint Propagation and Management, as well as conventional paradigms in the pursuit of the representation schemes and inferencing mechanisms that would be needed in this system.

## 7.1. Shape/form feature based CAD interfaces

Most of the conventional solid modeling or CAD systems are inadequate for mechanical design due to the complexity of design modifications, difficulty of manipulation, and lack of product information (e.g., material properties, surface finish, tolerance, surface condition, etc.).

To provide an environment which is suitable (1) for the user to design parts in the level of abstraction (designer does not need to know the design beforehand) and (2) for supporting automated applications such as (a) computer aided process planning and tolerancing, (b) numerically control code generation, (c) assembly plan generation, or (d) engineering analysis, an integrated form feature based CAD system has been studied. The technique of object oriented programming is used in the system to support user definition of form features which may be positioned and manipulated in a logical manner.

In the proposed system, the user is allowed to form and define the desired features by using the combination (boolean operations and deformations such as bending, twisting, tapering, etc.) of polyhedral primitives and flexible superquadrics family (superellipsoid, supertoroid, superhyperbloid) under generalized CSG (GCSG) paradigm.

The DKMS form feature interface system design consists of four modules:

1. **Form Feature Definition Module.** This module facilitates the necessary functions for designers to create new features or delete, manipulate, and modify the existing features. This module can be customized by adding feature knowledge (the designer's expertise in needed features) into the system to get a complete system to define features.

2. **Form Feature Modeling Module.** This module is basically a feature based solid modeler with a set of predefined form features which allow designers to visualize and modify part design. Within this module, the regularized boolean operations (union, intersection, difference, gluing) of polyhedral-polyhedral, polyhedral-superquadrics are embedded. Designers can extract the existing features from database and position them in a logical manner (above, below, toward, along, etc.) using the current geometry as a reference (see example). The system would perform necessary regularized boolean operations automatically to generate desired features.

Once any new feature is created, the feature would be placed into the feature hierarchy which describes the relationship between features. If a feature has been modified, the proper modification will be made by default on all its subfeatures. The modeling functions, such as database queries (e.g., feature hierarchy, directories, names and generic types) and graphic facilities (e.g., zooming, scaling, translating, rotating, hidden surface removal, rendering, etc.), are also provided in this module.

3. **Tolerance and Material Modeling.** In this module, designers are expected to provide information about dimensional tolerance, surface finish, and material type which can be used to drive the automation applications such as process planning, robot assembly planning, and numerically controlled code generation. Designers specify acceptable deviations of dimensions, datum planes, surface finish requirements, and material type to ensure the parts can be properly assembled and function as desired.

4. **Output and Feature Transformation Module.** Three types of output are generated by the system: (a) BRep (Boundary Representation) which is probably the most commonly used representation in CAD systems, (b) CSG tree, which shows how the part is formed in terms of boolean operations, and (c) relational graph which structuralizes the interrelationship among features in the part design; this interrelationship provides valuable information for robot assembly planning and process planning. The interpretation of features is commonly varied from applications. In other words, it is application-specific. An interface program may be written to transform the design database into an application-specific database to support various applications.

Example:
Suppose a bearing is needed to mount on a wall. The designer may start with a rectangular block (primitive) with proper dimensions as shown in Figure 12. The second step may be to put a feature called 'cut out' in the block. The designer can use the mouse to select points on edges and angle ø (default 90 degree) between the cutting planes to create a 'cut out' feature (see Figures 12, 13).

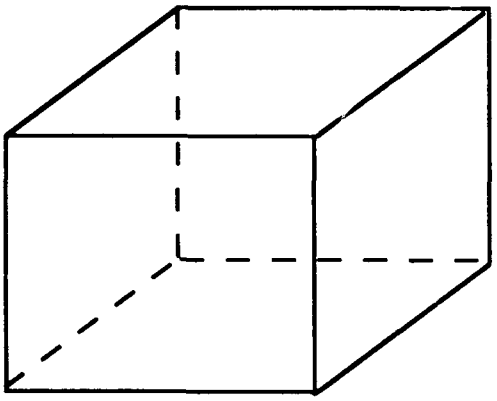


**Figure 12. Stock**

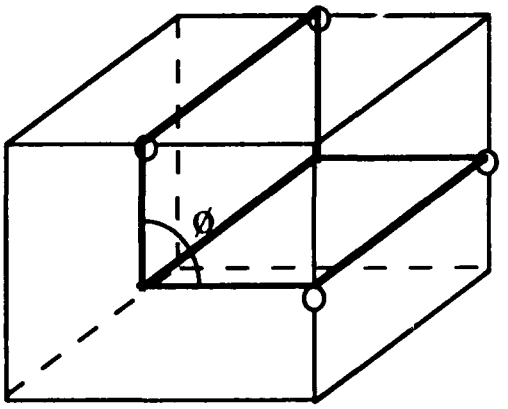



**Figure 13. Cut Feature**

The cut out can be modified by changing angle ø, or by moving faces (1 or 2) ALONG edges or normal vectors of faces as shown in Figure 14.

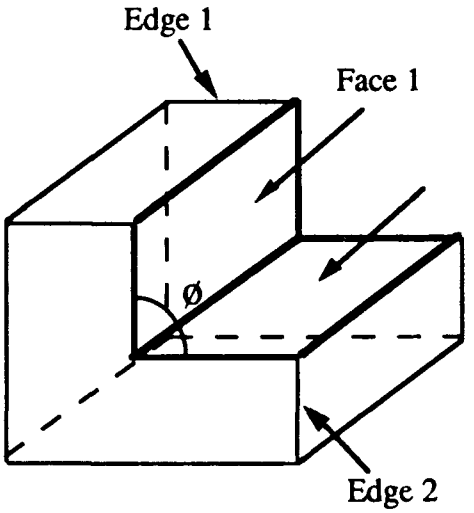



**Figure 14. Intermediate Part**

Suppose the user now issues the command:
"CREATE THROUGH HOLE ON FACE 2 RADIUS 2"

With this command, the system would generate a through hole on face 2 and 3 (bottom face) and check whether radius 2 is valid (too big or not). The hole can be moved along any edge, face normal vectors, or toward or away from any vertex. A part resulting from this type of command is illustrated in Figure 15.

After creation, the hole becomes a subfeature of the cut out and the cut out is a subfeature of the block. If any modification is made on the block, the cut out and hole would be changed accordingly; or if any modification is made on the cut-out, then the corresponding modification on the hole will be made automatically, unless the designer specifies not to do so. Furthermore, if there is a slot in the hole, then, if the size or location of the hole is modified, the proper modification would be made in the system by default.

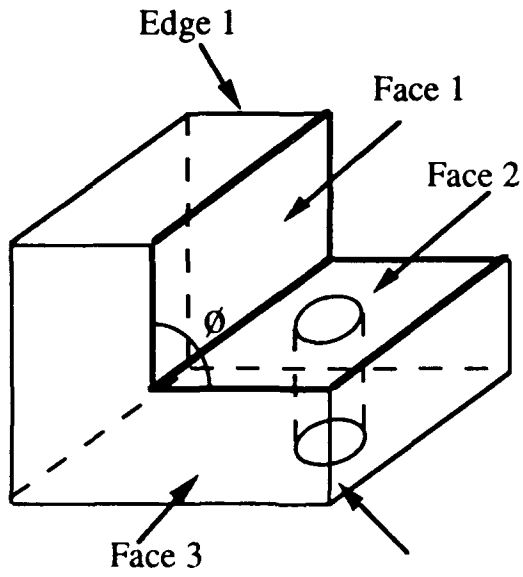



**Figure 15. Resulting Part**

In summary, if any modification (dimensions changing, repositioning, etc.) is made to a feature, the system would attempt to perform the corresponding modifications on all its subfeatures. Similarly, another two holes can be created and positioned on face 1 to complete the design as shown in Figure 16.

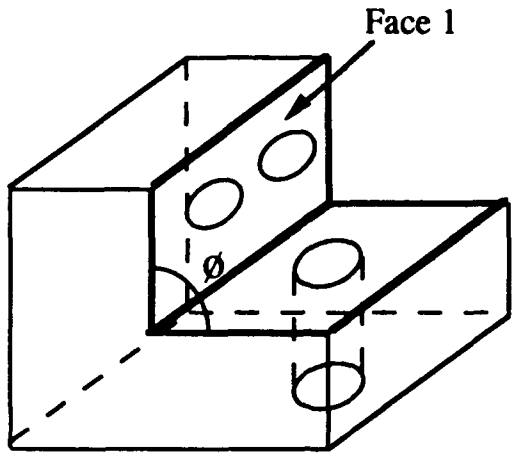



**Figure 16. Completed Part**

Besides the BRep geometry output, the system would also produce a CSG tree and a relational graph. Figure 17 illustrates the general form of a relation graph as described above.

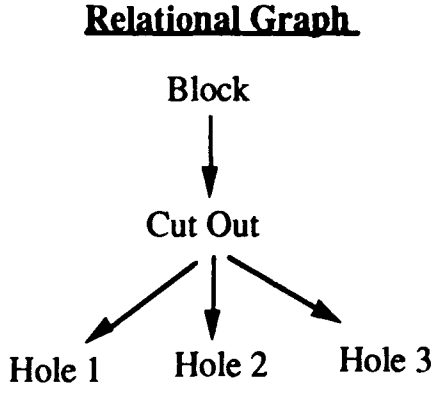



**Figure 17. Relational Graph**

## 7.2. Generalized CSG geometry engine

Most of CAD (Computer Aided Design) systems are CSG (Constructive Solid Geometry) based. Using a small number of primitive solids (cube, cylinder, tetrahedron, tube, sphere, etc.) as construction blocks, these CAD systems allow modeling of complex three-dimensional solids as planar-faceted polyhedra in a boundary representation. Although such modeling approaches are useful for many applications and very simple to use, they generally do not lend themselves to the design of sculptured surfaces (surfaces with double curvature), rapid prototyping, or design in abstract.

An algorithm for performing boolean operations (union, intersection, difference) on various representations is needed.

A GCSG (Generalized CSG) algorithm has been developed for supporting the incorporation of sculptured surfaces (e.g. Ferguson-Coons, Bezier and B-spline curves and surfaces, superquadrics) into a shape/feature based high productivity CAD system which allow users to conceptualize the design rapidly by using superquadrics and sculptured surfaces and to complete the design by using deformations and logical modifications (See Section 7.1.).

The general approach to accomplish the task of GCSG can be outlined as the following steps :

**Generalized CSG Algorithm Outline**

1. **Extraction::** If we are dealing with a primitive set that includes polyhedral (e.g. BRep and traditional CSG) or superquadric objects, then step one is to directly extract the underlying analytical representation of the superquadric and polyhedral object.

2. **Interpolation::** If a sculptured surface (e.g. Ferguson-Coons, Bezier or B-spline) is detected by the GSCG, then an interpolation algorithm would be used to calculate an analytical representation of the surface (by a set of control point coordinate and tangent vectors).

3. **Patch generation::** Convert the analytically represented sculptured surface or superquadric to a format for faceted approximation (polyhedral objects are already in a faceted format). This conversion is accomplished by generating a set of patches by recursive subdivision until a predefined tolerance is achieved.

4. **Triangulation::** Approximate each patch of surfaces by triangular facets to eliminate all curved surfaces.

5. **Refinement::** Collapse all the co-planar facets to achieve a more compact representation.

6. **Classification::** Perform reduction (determination of all possible intersection points between the two objects) and neighborhood classification (see the discussion on neighborhood classification) which was originally developed by Mantyla [Mantyla 88]. Then, depending on the operation desired (union, etc.), classify

facets, generate the resulting solid of boolean operations in BRep and produce the desired representation of the result (See [Mantyla 88] for an overview of the reduction and neighborhood classification algorithm).

Mathematical Background

The following sections overview the application of this general algorithm to the classes of solid representations that we would address in the Phase II effort.

1. **Superquadrics**. A superquadric is formed by the spherical product of two two-dimensional curves; one is defined horizontally, the other vertically.

Spherical product :

Given two two-dimensional curves

$$h(\omega) = \begin{bmatrix} h_1(\omega) \\ h_2(\omega) \end{bmatrix}, \omega_0 \le \omega \le \omega_1$$

and

$$m(\eta) = \begin{bmatrix} m_1(\eta) \\ m_2(\eta) \end{bmatrix}, \eta_0 \le \eta \le \eta_1$$

the spherical product x = m * h is defined as

$$\mathbf{x}(\eta, \omega) = \begin{bmatrix} m_1(\eta)\ h_1(\omega) \\ m_1(\eta)\ h_2(\omega) \\ m_2(\eta) \end{bmatrix} \quad \begin{matrix} , \omega_0 \le \omega \le \omega_1 \\ \eta_0 \le \eta \le \eta_1 \end{matrix}$$

Geometrically, **h**( w ) is a horizontal curve vertically modulated by **m(n)**; **m(n)** changes the relative scale of h, while $\mathbf{m_2(n)}$ raises and lowers it.

if

$$m = \begin{bmatrix} \cos \varepsilon_1 \eta \\ a_3 \sin \varepsilon_1 \eta \end{bmatrix} \quad \text{and} \quad h = \begin{bmatrix} a_1 \cos \varepsilon_2 \omega \\ a_2 \sin \varepsilon_2 \omega \end{bmatrix}$$

then

$$m * h = \begin{bmatrix} a_1 \cos^{\varepsilon_1}\eta \cos^{\varepsilon_2}\omega \\ a_2 \cos^{\varepsilon_1}\eta \sin^{\varepsilon_2}\omega \\ a_3 \sin^{\varepsilon_1}\eta \end{bmatrix} \quad \begin{array}{l} -\pi \le \omega \le \pi \\ -\pi/2 \le \eta \le \end{array}$$

Let $x = a_1 \cos^{\varepsilon_1}\eta \cos^{\varepsilon_2}\omega$

$y = a_2 \cos^{\varepsilon_1}\eta \sin^{\varepsilon_2}\omega$

$z = a_3 \sin^{\varepsilon_1}\eta$

the inside out function can then be written as

$$f(x,y,z) = \left[ \left(\frac{x}{a_1}\right)^{\frac{2}{\varepsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\varepsilon_2}} \right]^{\frac{\varepsilon_2}{\varepsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\varepsilon_1}}$$

if $f(x, y, z) = 1$ ---> $(x, y, z)$ is on
$< 1$ ---> $(x, y, z)$ is in
$> 1$ ---> $(x, y, z)$ is out

The shape and size of superquadrics can be modified by varying epsilon1, epsilon2 and the a1, a2, and a3 parameters. The global deformations (bending, tapering, pinching, etc.) can also be applied to superquadrics to generate desired shapes.

2. **Ferguson-Coons surface**. A Ferguson-Coons surface is a surface interpolated through a rectangular grid of control points. The control points define a set of Ferguson-Coons bicubic parametric patches which are blended to satisfy continuity constraints between adjacent patches.

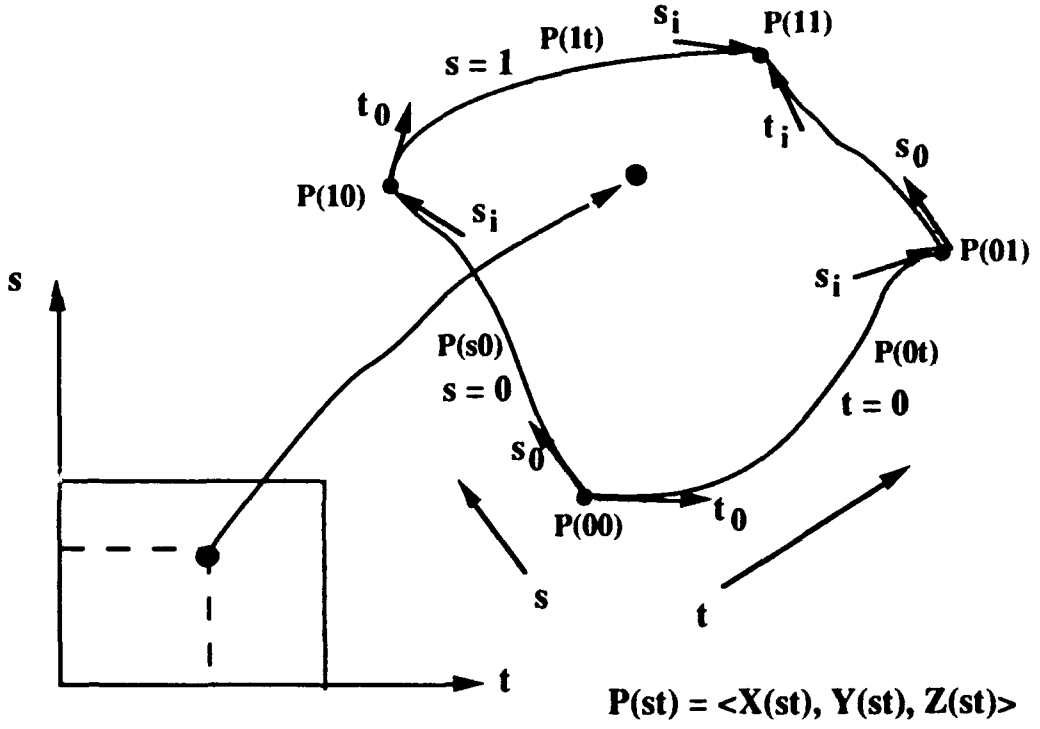
In ( s, t ) parametric space, each Ferguson-Coons patch is normalized over intervals s, t in [0,1] and is defined as:

$$P(s,t) = [s^3 \; s^2 \; s \; 1] \, NBN^T \, [t^3 \; t^2 \; t \; 1]^T$$

where

$$N = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

and B is a 4 by 4 matrix of coordinate and tangent information at control points. Figure 18 shows a Ferguson-Coons patch and its defining information; a six-patch FC surface (Figure 19).




FC bicubic patch

**Figure 18. Ferguson-Coons Patch and Defining Information**

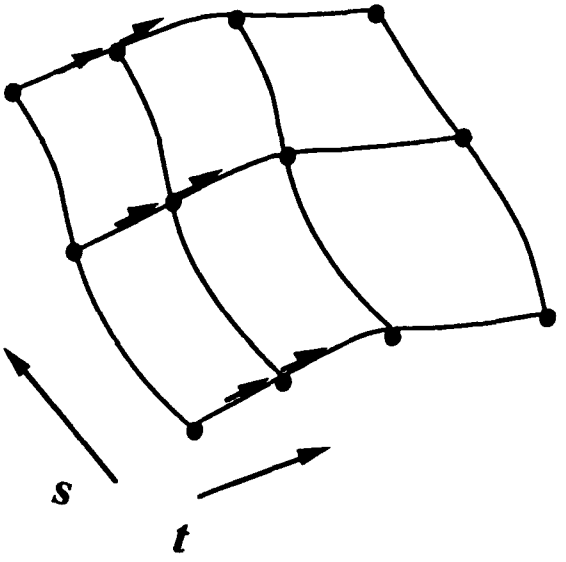



**Figure 19. Six Patch FC Surface**

3. **Bezier surfaces**--Bezier surfaces are also defined by a rectangular grid of points. In (s,t) parametric space, a Bezier surface is normalized over the intervals s,t in [0,1] and is defined as

$$Q(s,t) = \sum_{i=0}^{n} \sum_{j=0-}^{m} B_{i+1,j+1} J_{n,i}(s) K_{m,j}(t)$$

where

B is a control point,

$$J_{n,i} = \binom{n}{i} s^i (1-s)^{n-i}$$

$$K_{m,j} = \binom{m}{j} t^j (1-t)^{m-j}$$

4. **B-spline surfaces**--A B-spline surface may be defined as:

$$Q(s, t) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i+1, j+1} N_{i,k}(s) M_{j,l}(t)$$

where

$$N_{i,1}(s) = \begin{cases} 1 \text{ if } x_i \le s < x_{i+1} \\ 0 \text{ otherwise} \end{cases}$$

$$N_{i,k}(s) = \frac{(s - x_i) N_{i,k-1}(s)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - s) N_{i+1,k-1}(s)}{x_{i+k} - x_{i+1}}$$

$$M_{j,1}(t) = \begin{cases} 1 \text{ if } y \quad t < y_{j+1} \\ 0 \text{ otherwise} \end{cases}$$

$$M_{j,l}(t) = \frac{(t - y_j) M_{j,l-1}(t)}{y_{j+l-1} - y_j} + \frac{(y_{j+1} - t) M_{j+1,l-1}(t)}{y_{j+l} - y_{j+1}}$$

$x_i$ is the ith value in the knot vector in the s direction, $y_i$ is the ith value in the knot vector in the t direction and B is a control point.

The faceting process on superquadrics is done in the $\omega$ and $\eta$ space; triangular facets are generated by sweeping along two two-dimensional spaces. The sculpture facets are derived from the recursive subdivision algorithm (see the following Figure 20) with a specified tolerance value.

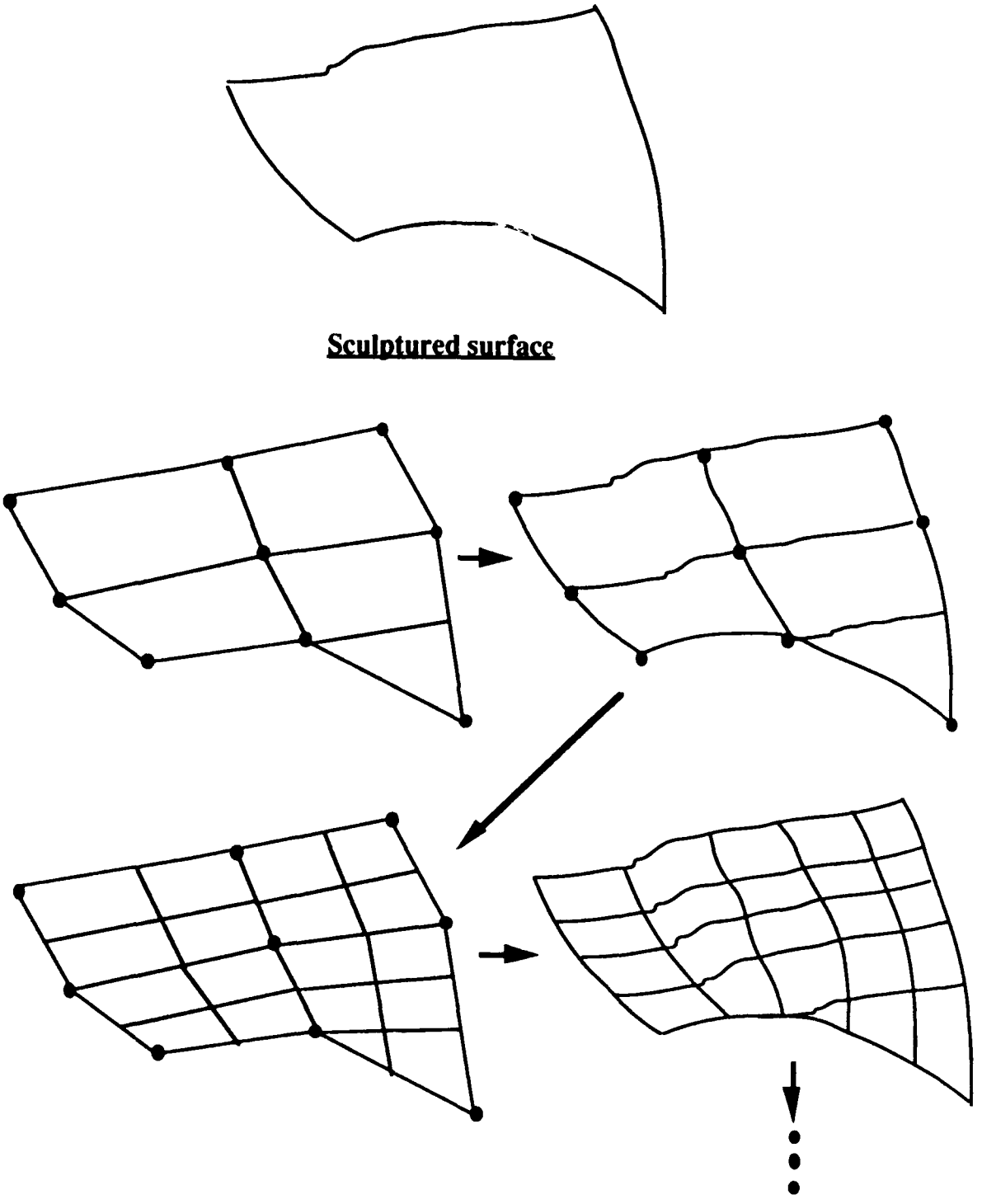



**Figure 20. Sculptured Surface Faceting using Recursive Subdivision**

After completion of the sculptured surfaces subdivision, superquadrics sampling, and polyhedral objects extraction, the faceting information is obtained. The faceting information is the input needed in the neighborhood classification process. The neighborhood classification algorithm is then being used to complete the task of GCSG and generate the resulting BRep solid.

## Neighborhood Classification Algorithm

In the first stage, the set operations (union, etc.) problem can be reduced into a collection of vertex neighborhood classification problems via Reduction Procedure.

### Reduction Procedure

Let A and B be two polyhedral objects.

1. Locate all pairs of edges eA of solid A and eB of solid B that intersect each other properly, i.e., at an internal point of both edges. Subdivide both edges at their intersection point, i.e., replace each edge by two edges and a new vertex lying at the intersection point. ( edge-edge intersection point).

2. Locate all edges of A that pass through a vertex of B. Subdivide all such edges at the intersection point. (edge-vertex intersection point).

3. Do step 2 symmetrically for edges of B and vertices of A.

4. Locate all coincident pairs of vertices vA of A and vB of B, and store them for later processing. (Of course, the resulting set includes at least all vertices added during step 1 through 3.) (vertex-vertex intersection).

5. Locate all edges eA of A that intersect a face fB of B properly, i.e., at an internal point of fB. Subdivide all such edges at the intersection point. (edge-face intersection).

6. Do step 5 symmetrically for edges of B and faces of A.

7. Locate all vertices vA of A that lie within a face fB of B and store the pair (vA,fB) for later processing. (This set will include all vertices added during step 5.) (vertex-face intersection).

8. Locate all vertices vB of B that lie within a face fA of A and store the pair (vB,fA) for later processing. (This set includes all vertices added in step 6.).

After the reduction step, the necessary information for generating the resulting solid can be obtained by executing one of two kinds of vertex neighborhood classification steps as follows (see [Mantyla 88] for details).

### Vertex Neighborhood Classifications

There are two kinds of vertex neighborhood classifications needed for the GCSG.

1. Vertex-face classification: The processing of a pair of a vertex of one solid that lies on a face of the other solid.
2. Vertex-vertex classification: The processing of a pair of coincident vertices of A and B.

Once all faces have been classified as "IN," "ON," or "OUT" through the vertex neighborhood classification, the resulting solid can then be derived using the following three equations :

$$\mathbf{A\ union\ B = A\ out\ B} \oplus \mathbf{B\ out\ A}$$

$$\mathbf{A\ intersect\ B = A\ in\ B} \oplus \mathbf{B\ in\ A}$$

$$\mathbf{A\ set\text{-}difference\ B = A\ out\ B} \oplus \mathbf{(B\ in\ A)^{-1}}$$

where $\oplus$ denotes the gluing operation in the context of solid modeling and $(B \text{ in } A)^{-1}$ denotes the complement of B in A.

For an illustration see the sequence of illustrations in Figures 21 through 24.

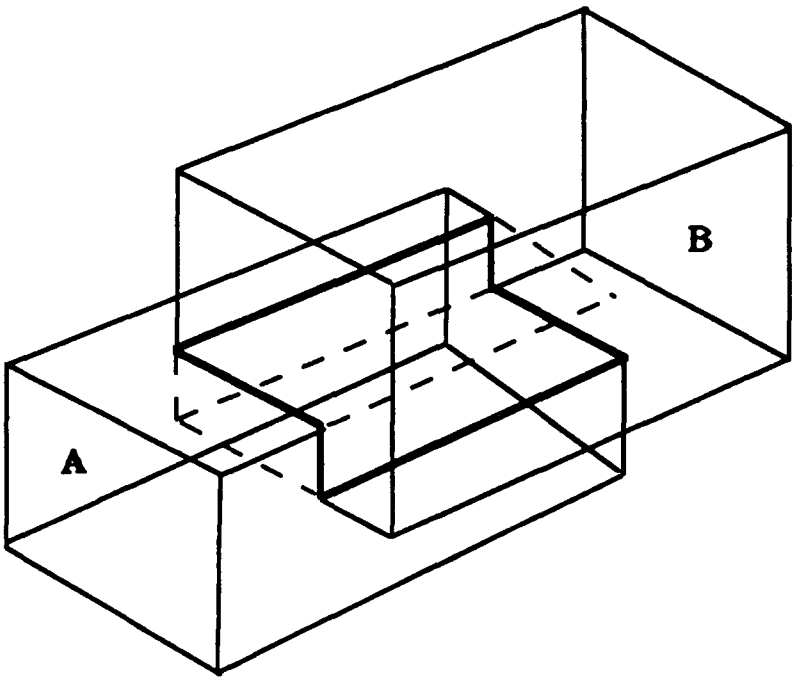



**Figure 21. Two Polyhedral Objects**

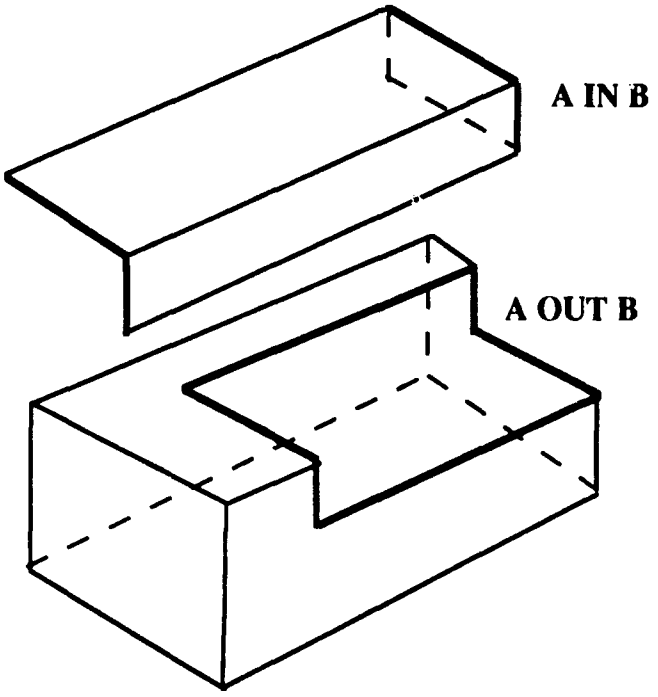



**Figure 22. A IN B and A OUT B**

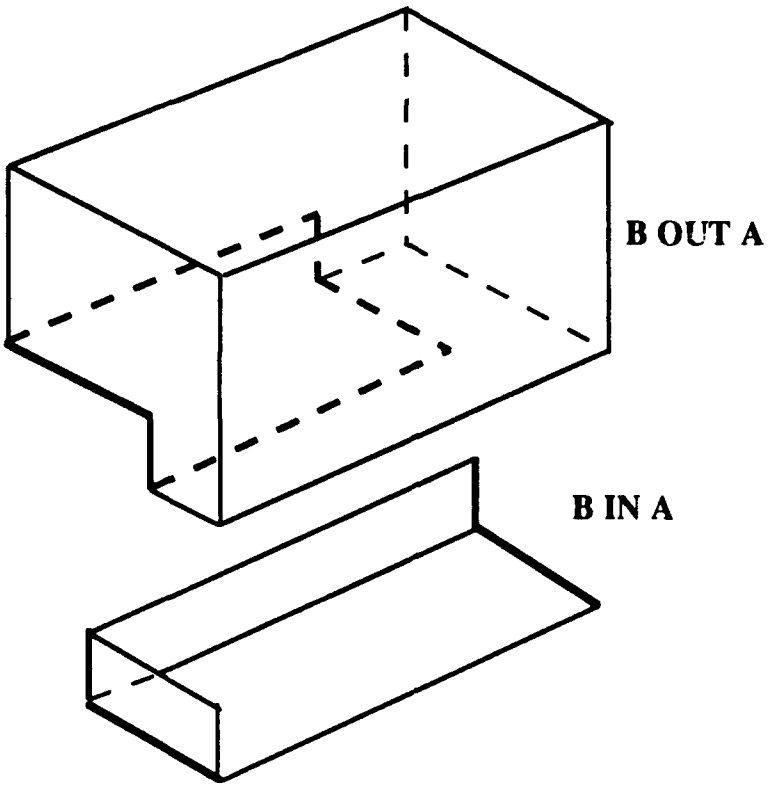
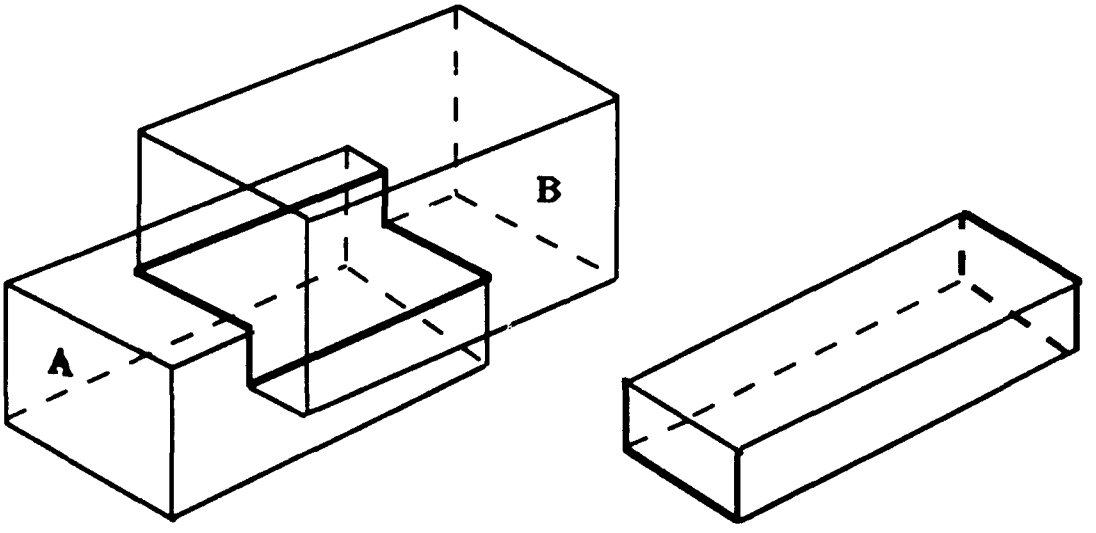



**Figure 23. B OUT A and B IN A**




A union B = A out B $\oplus$ B out A  A intersect B = A in B $\oplus$ B in A

**Figure 24. Illustration of Face Classification and Resulting Solid Generation**

**Generalized Constructive Solid Geometry.** The phrase "generalized constructive solid geometry" refers to the capability to combine definitions of assorted geometric solids which have been stored in different representations by performing boolean operations on the solid objects. For instance, one may want to intersect an object stored using a polyhedral representation with an object stored using a superquadric representation. It should be noted that the existence of multiple solids representations is only partly creditable to the historical use of a variety of CAD packages in engineering / manufacturing enterprises. It is also creditable to the greater efficiency or effectiveness of certain representations for specific purposes; thus, the use of multiple solid representations is *inherent* to the engineering process.

The CSG tree that we are considering in this exploratory work is comprised of boolean operations on the following solids and surface representations:

a. Polyhedral representations,
b. Superquadrics representations,
c. Surface generation (Bezier, Ferguson-Coon, and B-Spline) representations,
d. Sweeps (Generalized cylinder) representations.

Polyhedral representations are well-known and have a well-developed underlying theory. Experimentation in this project has focused on boolean operations involving the neighborhood classification and BSPT variations. The major result to date is that the neighborhood classification schemes appear to yield much faster boolean operations.

Boolean operations for solids using the Superquadrics representation have been developed using our own approach. At this point, the basic theory is well developed. However, the operations are complicated in the case of the superquadrics representation when the objects have been globally deformed, and even more complicated under local deformations. In addition, considerable work remains in finding sufficiently efficient and accurate methods for performing the operations.

Boolean operations among objects of like representation are in all cases simpler than such operations among objects of hybrid representations We have already developed some basic algorithms in this area; however, work is ongoing and constitutes a substantial effort to complete.

# 8. Advanced Engineering Modeling Support

One of the primary components in any design support or automated "designer" system is the engineering analytic model (s). One of the goals of DKMS is to also support the development / refinement of the models of the physical systems or processes used in that design process.

A large part of engineering design involves the manipulation of models. Yet most computer models today are coded in such a way as to be very resistant to change. Any changes to the existing models frequently require extensive recoding of very old and largely undocumented code. New model developments are equally difficult because the engineer has no means of expressing the model concepts to a computer in a way natural to him. What is needed is an intelligent model building system that knows about model concepts, solution technique types, and implementation methods. Phase II of this project will produce a set of utilities for constructing engineering models and for managing modifications to those models based on the KBSI Modeler capability. The resulting modeling support environment will provide capabilities in the areas of:

1) Qualitative simulation (curve based reasoning).

2) Bond Graphs for dynamic performance modeling of heterogeneous physical systems

3) Constraint management for supporting the derivation of complex systems models from first principles as well as test data results.

Knowledge Based Systems Engineer's Modelling Assistant [KBSI 89] was designed to aid in the development of analytical models for expert systems. Based on both bond graph modeling and constraint management methodologies for engineering performance modelling, it represents one of the most sophisticated support environments for such modelling. Its features include graphic, on-screen model generation, a relationship interface which ensures consistency and completeness, and direct access to a programming language. The tool was used to generate a simulation of the cool down of the interior of a car as well as the phenomenological models of all the components in the associated heating and air conditioning systems for the Chrysler Corporation.

Of the modeling support utilities to be provided by the DKMS, the most powerful is the constraint management. Design is constraint-oriented: much of the design process involves the recognition, formulation, and satisfaction

of constraints. The CMS (Constraint Management System) can aid designers to identify and explore the boundaries of the design space. It can help to determine which are the most important design parameters, specifications, and constraints and evaluate the global performance of the alternatives to select the most appropriate ones for detailed analysis and refinement.

The main functionalities to be provided by the CMS component of the DKMS include:

1. Causality and dependency determination via Bipartite matching,
2. Strong component (simultaneous constraints) detection,
3. Consistency verification,
4. Explanation and qualitative reasoning,
5. Solution sequence generation,
6. Sensitivity analysis.

Besides the engineering performance modeling support. the CMS will be incorporated into the container object representation mechanism as described in the following section.

## 8.1. CMS in containers

Composite objects, because of the ability to specify recursive template structures in their definitions, have a need for constraint management capabilities. These capabilities would be used to ease the specification of the instantiation logic as illustrated in the following example.

ICAD has :trials, and :query-trials keywords within the defpart macro that define a list of possibilities for an attribute and a test to determine which attribute is chosen. Each possibility is tested in left-to-right order, and the value of the attribute is the value of the first success. An example of the use of :trials is where the analysis of a list of catalogued parts might require the total mass of the object's parent. Since the total mass depends on which cataloged part is used, :trials is used to perform the analysis.

The syntax of :trials is as follows:

:trials ([:trial-attribute-name (trial-test list-of-possibilities)]

:trial-attribute-name is the name of an attribute trial-test is an expression list-of-possibilities is the list of values to be tried

(defpart table (box)

```
:attributes  (:leg-strength (* (the :leg :any :surface-area) 65.0)
                :length 30 :width 20  :height 20)
:trials  (:leg-type ((< (the :total-weight) (* 4.0 (the :leg-strength)))
                        '(AISC:w5x16 AISC:w6x20 AISC:w8x40)))
            :parts((table-top :type box
                              :height 1.0
                              :position (top 0.0))(leg :type (the :leg-type)
                              :orientation (:rotate :left) (2 2))))
```

When information about the table leg is required, the system has to calculate the leg type. The test uses :total-weight and :leg-strength which depend on the type selected. We are proposing to implement a more flexible form of this type of control structure using a primitive form of constraint manager. The traditional type of searching is computationally expensive as it does not use any searching strategy. As shown in the following example, the user can intercede by conditioning the search data where the cataloged data is sorted in ascending horsepower before the search is attempted.

```
(defpart mower (box)
      :query-trials  (:motor-hp ((> (the :motor-hp) 10)
                                    :horse-power
                                       '(:sort (:select motor
                                                   (eq (iq:the-element :vendor)
                                                       :westinghouse))
                                               ((< :horse-power))))))
```

Using a constraint management approach would allow the system to automatically compute a solution strategy based on the "knowns and unknowns" of a particular design session. It would also simplify the container specification as less procedural specification for instantiation needs to be supplied.

# 9. Services for Evolution Control within the DKMS

To provide support for design engineers in the control of the evolution of the design data, a design environment must:

1) Manage and propagate constraints specified in the product needs analysis or in design goal specifications and enforce those constraints on the evolving solid models.
2) Transparently control the configuration of the design artifacts.
3) Support rapid browsing and modification of the above information about an evolving design.
4) Support definition and management of perspectives.
5) Support definition and management of versioned objects.
6) Provide configuration decision management capabilities.

Such capabilities are required if current "computer aided drafting" (cad) and computer implemented engineering analysis programs are to evolve into effective Computer Aided Design and Engineering (CAD/CAE) **environments**. Current cad interfaces are so user abusive that even trivial parts can require weeks of drafting time to enter. Lack of computer support for determination of the effect of changes (at a semantic level, not just at an effected parts list level) causes poor disposition decisions that can result in major delays in a product program. Inability to enforce constraints across multiple representations of design data and poor decisions often result in a requirement for a complete reentry of the design data. This amplifies the negative impact of the poor cad interfaces. Configuration control is normally manually performed, resulting in an average of 8 man-hours required to locate the current released design data for modification.

## 9.1 Definitions

The details of our approach to perspective, configuration, and version management and control developed during Phase I are described in the final report. This approach is centered around the following definitions:

**Boot Layers:** A layer of the knowledge base that contains a set of assertions that describe the knowledge base, i.e., the boot layer contains environmental objects that describe the knowledge base. Therefore, the boot layer must be read in before the knowledge base can be accessed.

**Community Knowledge Bases:** Knowledge bases shared and maintained by a number of knowledge engineers or designers.

**Control Points:** Triggers (switch based upon event occurrence) plus a condition set. If the condition set is satisfied, a set of policies is invoked.

**Environment:** Provides a namespace in working memory for associating names and unique identifiers with objects from a particular knowledge base.

**Environmental Objects:** Knowledge bases, environments, and layers represented are instances of an Environmental Class.

**Environments:** The user works in a personalized environment. An environment provides a lookup table linking unique identifiers to objects in the connected knowledge bases. The user may indicate dominance relationships between selected knowledge bases. When an object is referenced, the dominance relationships determine the order in which knowledge bases are examined to resolve the reference. By making personal knowledge bases dominate over community knowledge bases, the user can override portions of the community knowledge bases in favor of his own knowledge bases.

**Global Name Table:** A table, accessible from any environment, containing all the environmental objects.

**Knowledge Base State**: A superclass of a knowledge base referring to an explicit set of file layers or knowledge base states.

**Knowledge Base:** Files built up of a sequence of layers of incremental changes. A user may choose any subset of layers of the knowledge base or share a communal database without incorporating the most recent changes. Thus, the user may refer to or restore earlier versions of the database.

**Layer:** A portion of a file containing descriptions of objects.

**Multiple Alternatives:** Environments provide fast access to alternative versions. A user can have any number of environments available at any one time since each environment is

isolated from the others, and information can only be transferred between environments explicitly through the knowledge bases.

**Policy:** Constraints that modify the system behavior.

**Unique Identifiers:** The ability to determine which layers are referring to the same entity is critical in shared knowledge bases. Therefore, unique identifiers have to be assigned to objects before they are written to a knowledge base.

**Updating Community Knowledge Bases:** Updating a Community Knowledge Base is accomplished in two steps. First, users make tentative changes to their own environment which can be saved in a layer of their own knowledge base. The knowledge base manager can later copy these layers into the community knowledge base. This separation of tasks encourages experimentation with proposed changes because the user does not have the responsibility for maintaining consistent knowledge bases for shared use by the community. This mechanism also allows individuals to compare each other's designs by exchanging layers of their personal knowledge bases.

**Version:** A defining structure of parent / alternative instances of objects.

**Version Pointer**: A generic reference between versioned and non versioned objects.

**Versioned Object:** Any container, design, knowledge unit, policy, control point, layer, etc., which maintains version pointers.

## 9.2 Configuration control and version management

Providing an environment that meets design support environment needs requires a set of tools that addresses the concerns of version management and configuration control. Discussions on how these concepts can be implemented and employed in a system requires an understanding of several other topics. These are control points, policies and constraints, and to some degree access control. In this section these concepts will be described in more detail and our approach to their use will be outlined.

Version management and configuration control are by their very nature linked. In an object-management system, an object would be represented by a set of versions of that object (either composite or non-composite), normally ordered by time. Creation of a new version for an object would be a function of configuration control within the object management system. Furthermore, it will likely be associated with some control point's set of policies and constraints.

In a design support environment, the primary reason for object versions is to maintain the design or decision history of an object. There are two levels of version management.

**Public-level:** In public-level version management, the version history of a managed object records the sequence of changes to the object as seen by all users (or classes of users) of the system. In this level of version management, an object is defined by policies and procedures of the organization or of the system management. In this level we would see:

1) Versions of objects representing changes to approved/released product designs. In the case of the DKMS, this would likely be seen in the objects maintained in the Community Knowledge Bases.

2) Versions of objects representing changes to the DKMS integration services components.

3) Versions of objects representing changes to the knowledge engineering support components.

4) Versions of objects representing changes to the life-cycle engineering knowledge bases.

5) Versions of objects representing changes to the engineering performance models.

6) Versions of objects representing changes to the analysis tools used to predict component or system performance based on the engineering models.

7) Versions of objects representing generic functions of shared services (e.g. user interface utilities, geometry engines, graphics display utilities, animation, simulation, model management, etc.)

**Private-level:** Private-level version management supports the representation of time- or sequence-dependent changes to an object at a level that is private to an application or specific user. Definition of an object at this level of version management is a function of the application. In application-level versioning, the versions of an object represent changes within an object relative to itself. An example of this level of versioning can be seen in the DKMS personalized user environments. In such a personalized environment the user would be able to invoke version control support to manage a set of visualization formats that he has evolved for particular engineering tasks. Engineering designer systems are also amenable to application-level version management. In these systems, version control can be used to maintain the history of the object design process. In such a designer environment, the user would be able to invoke version control support to record incremental changes that were made to an object (during design exploration or development) prior to its approval as part of an approved or released design.

In the DKMS, our design objective is to provide a reusable set of version management services that could be applied uniformly across the public and private levels as described above. Configuration Control for design artifacts in an object management environment can be provided in two ways.

1) First there is the "check-in / check-out" approach to configuration control. In this approach, the artifacts (versioned objects) are checked out by an authorized user. When this occurs, other users are locked out until the artifact is checked back in. For example, product design data would go through several states throughout its life cycle. Starting with a working version, it would progress to a released state and finally would become archived. The states are essentially versions of the object.

2) The other method is referred to as "percolation." In this approach, the evolving artifact is viewed as a collection of other objects (i.e., it is a composite object). Percolation strategies attempt to provide finer grained control over both the objects and their relations to other objects. The versioned object can be set up as a tree structure with the design artifact being the root of the tree and each node being a composite object. In this approach, a change to a low-level object in the structure that has

no impact on the container (or relations to other objects in the container) can easily be managed separately. Changes to low-level objects that result in the evolution of the container can result in new versions of all other objects within the version tree of that container. Design changes percolate back up the tree. In this approach, versioning is used to represent an object's compatibility with other objects because successive versions for a given object within the design may be unchanged.

An integral part of configuration management is change control. Here the term "change control" refers to what procedures the system will enforce when someone recognizes the need for a change, how to proceed with the change, and the procedures for recording a completed change. Change control includes facilities for requesting change, informing affected users of the need for a change, creating the change, and reporting the completed change. Message passing facilities are required with most stages of the change control procedures. It is the changes to an object that result in the need to maintain versions. Functions associated with the change control aspect of version and configuration management are triggered by control points.

By definition, control points are flags attached to the entrance and exit functions of commands, transactions, or service requests. Associated with control points will be a set of before, during, and/or after execution enterprise policies or constraints. In a system that employed control points, execution of a command would be a function call. The procedure would be:

1) The client (user / system / application) issues a request.

2) The system checks for control points associated with the before execution phase. If there is one or more, evaluate the prioritized list of policy or constraint functions.

3) Perform any actions indicated by the functions and return an "abort" or "continue" flag to the entrance function.

4) If "continue" initiate response to the service request.

5) If "during" constraints exist, initiate monitoring process to check for and enforce the specified constraints.

6) For either a normal or abnormal termination, check for control points associated with either the exit command or abnormal termination handler. If there is a control point, evaluate the prioritized list of policy or constraint functions associated with the control point, and perform the actions indicated by those functions.

Configuration management at both the public and private levels is built on top of the concepts of change control and version control along with control points and policies/constraints to manage the product life-cycle.

## 9.3 DKMS configuration control and version management

In an environment designed to augment the engineering design process, support for all these concepts would be necessary. The DKMS environment would have to provide version control services usable directly for the public and private version management areas described above. It must also support the definition of configuration management services as described in the previous section. The DKMS approach presented here is based on functionality similar to that described for the Engineering Environment Services provided by the Engineering Information System (EIS) prepared by Honeywell Systems and Research Center [EIS 89]. The requirements for the services described in that document were aimed at supporting an environment for administrative and electronic design information. We are extending the concepts to the entire DKMS environment, including the management of the evolving life-cycle engineering knowledge bases.

### 9.3.1 Product data evolution management

The term product data refers to the public product definition artifacts that are managed by the system. If the DKMS environment were used by an automobile manufacturing organization, these artifacts would be such items as complete vehicle designs, radiators, tires, etc. Other information maintained would be supporting data such as cost analysis reports, vehicle design requirements, etc. These items would all have versions and most of them would be composite objects. The default product data configuration management services provided by the DKMS would be based on the "check-in/check-out" strategy, with public-level version control. This default was chosen because it represents the approach most likely to be found in current corporations.

Consider, as an example, a product that undergoes four phases in its life-cycle. The first of these is an initial design phase. During this phase the product undergoes initial development, testing, and refinement. While the product is in this state, there will be frequent changes to the configuration and possibly parts of the design will remain undefined for a period of time. As this phase nears completion, certain portions of the design may become immune to changes. The phase ends sometime early in the product production phase. During the production phase, changes may still be made to parts of the design; however, there will be extremely stringent constraints on these changes. The next stage in the life-cycle of the product comes when production terminates. The product in this phase will likely continue to be supported by the organization. Once production of the product ends, the need for design changes becomes less likely; therefore change-control constraints would force very high level approval for any design change. At some point the organization will no longer support the product. This places the product design in the final phase of its life-cycle. The design data still needs to be maintained because there is much usable information in the design, and the organization would benefit by keeping the design available for review and copying. However, modifications will no longer be made to this particular design; it is frozen. These four phases that a product design goes through are called states. We call these states development, production, support, and retired.

The DKMS environment will provide functions which will allow an organization to define the descriptions for the objects which exist in its product data databases. The objects created will contain, in addition to the attributes that describe the product, version and state attributes. The functions provided by DKMS would be such functions as create *versioned-object*. A versioned-object would inherit version and time-stamp attributes. Simple versioned objects would likely be non-composite objects or components of the product that are out-sourced (i.e., components of a composite product that are neither designed nor produced by the organization). For an object that would pass through phases in its life-cycle, the DKMS environment must provide a function to create *state-object*. State-objects will likely be composite objects. These would be product components produced by the organization, but in particular a product which is a composite object, such as a vehicle produced by an automobile manufacturer. A state-object would inherit version, state, and time-stamp attributes. The time-stamp attribute not previously mentioned would be a means of assisting in a merge operation that would be necessary when multiple users are working on the same design. For an object to be properly described, it will be necessary to include the ability to describe objects as *component-parts* (i.e., list of objects that are part of another

object). Each object in a list of component-parts would have the component object in their *is-a-component-of* list. In addition to providing a facility for defining objects, functions will be provided for describing the states in which the organization's product data can exist. Functions will also be provided for defining what would constitute the termination of one state and the start of the next. In addition to the above described version management capabilities, additional functions to be provided by the DKMS environment include:

- Problem report generation,
- Change request processing,
- Message passing,
- Status report generators,
- Status / state browsing facilities.

Requirements for these and other configuration management functions require further definition and refinement. These issues will be addressed in the next phase of the DKMS project.

Controlled manipulation of the Product Data will be provided by the Knowledge Base which is described in the following section.

### 9.3.2 Knowledge base evolution control

In addition to maintaining a database(s) of product data, the DKMS must also manage a set of life-cycle engineering knowledge bases. These knowledge bases can be thought of as a database that would be composed of the rules, constraints, policies, and principles of life-cycle engineering in the organization. The knowledge base data will also have versions. These rules, constraints, policies, and principles of design influence the design process via the control points to which they are attached. Most important, the knowledge base maintains an additional version management data type that we call a context. The context version management capability allows one to control the "applicability" of knowledge and the validity of certain inferencing mechanisms. For example, the execution or application of a particular rule, or constraint, etc. doesn't always occur. For an illustration of this point, assume that the organization recently acquired a new cooling system analysis tool which requires more data than the old. All new designs must use this tool. A context would then be defined to allow the inference

that if the released version of the vehicle is later than some date, then the new analysis tool has been used in its design process. The context would determine which set of rules, policies, constraints, and principles of design are applied in a given situation.

Items in the Knowledge Base will be treated as versioned-objects and functions will be provided in the DKMS environment which will allow a qualified user of the environment to define objects of each of these types. The functions for defining objects in the Knowledge Base are the key to personalizing the system for different organizations. They allow the creation of versioned objects in the Knowledge Base that provide for:

1. User-defined rules and organization policies. Rules can be defined that limit user access to only certain applications.

2. User-defined constraints on objects in the Product Base. An example of a constraint could be projected production cost of object should not exceed x number of dollars.

3. User-defined policies such as change request must be approved by department manager, or change request requires approval by the division head.

4. User-defined context. In the previous example of policies, there appear to be two conflicting change request policies; however, the context could be "apply the first policy when the stage is *development* " and "apply the second when the stage is *supported.*" A context would also be used to control such procedures as which analysis programs to use or whether certain data is stored in a database or is calculated. Often the context will be associated with the value of the time-stamp attribute in a design object.

5. User defined principles of design.

Sets of the objects in the Knowledge Base will be associated with control-points. Because the objects in the Knowledge Base are separate entities, an object can be associated with multiple control-points. The definition of the control-points and the sets of objects to be associated with each would be defined by an authorized Knowledge Base manager. Definition of a control-point would be the equivalent to setting a flag on the entrance to or exit from a given function or command. A context would be a means of attaching different sets of objects to a control point based on some user-

defined situation. The context would also be used to enforce the use of particular design and analysis tools based on some user defined criteria.

For the objects in this Knowledge Base to be of maximum use, a command to the design system functions and applications would go through a command interpreter. The actual commands would be called by the command interface rather than by the user. For instance, the user wishes to run a cost analysis on a given design. The user would type a command such as *cost design x* . The command interface would:

1) Receive the command,

2) Check the version and state of design x,

3) Select the appropriate entry control-point(s) for the context of the design,

4) Check the set of policies/constraints associated with the entry control point,

5) Execute the functions associated with the policies/constraints,

6) Initialize the appropriate costing application,

- The user would:

  a) perform the costing process,

  b) terminate the process.

7) Select the appropriate exit control-point(s),

8) Check the set of policies/constraints associated with the exit control point,

9) Execute the functions associated with the policies/constraints, sending any required messages, change versions or state of the design object, etc.

The previous two sections have not directly considered the needs of the individual user of the system; they have been more concerned with the needs of the organization. However, proper handling of the organization's

needs for a product base and a knowledge base requires that consideration be given to the individual user of the system.

### 9.3.3 Personal design history

The DKMS will support the individual user in the definition and use of personalized environments in which he/she can define personalized interface presentations, rules, constraints, etc. These would be attached to control points in the same manner that those in the knowledge base are attached to control points. Another area that needs to be considered in a description of the individual user's environment is the ability of the user to experiment with the design of objects within his/her design domain without modification of the community databases. For instance, consider the designer of an automobile. This individual needs the freedom to experiment with vehicle body designs in order to develop ideas for future development. While being able to make multiple changes to an object within a specific domain, the user should retain the ability to back out of those changes at will. These topics will be discussed in this section.

The user's domain would be defined by the role type assumed. Role type is an aspect of access control, which is of vital importance in any multiuser system. The installations will be allowed to define different role or user types to which individual users can be assigned. Associated with each role type will be a set of allowable functions or operations that users of that role type can execute or use. The system manager will assign each user to one or more role types. A user can then login to the system, which would place the user into a private environment. After login, the user would select one role type to use. This procedure will provide the user with access to only those operations required. The environment will be designed so that an individual does not have to terminate a session and reenter the system to change roles. However, for the purposes of security, when a user exits one role type and starts another, the system will automatically clear the session (after prompting for the save operation). The user would then have a different set of commands available, some of which could be duplicates of the preceding set. With the definition of available functions for a role type, the objects in the product base that an individual is allowed to modify can be controlled. This restriction will provide extra protection for the product data. The engineering design support environment will provide an *exit-role* command in all role type descriptions. Ending a session would automatically execute the exit-role command.

If the functionality is available in a particular user's role type, then the user can define rules, constraints, control-points, etc. that would be applicable only in his/her environment. The creation and use of these control-points and policy sets, etc. is identical with those in the knowledge base; however, the use of individual control-points would be in addition to the organizational defined rule sets and would not be allowed to invalidate any requirement defined by them.

Whether the user is working on product data or experimenting with a design that may eventually become product data, the objects created and manipulated within a user's personal environment require versioning. In the individual's private environment, the primary reason for versioning is to maintain a history for changes made to an object with respect to other objects within the design.

The designer would make incremental changes to the object of design with each change recorded as a version of the total object. The designer could browse through these different versions selecting any one for further refinement (each refinement would represent a change and thus another version). Eventually one version would prove acceptable and it would return to the Product Base where its inclusion would produce a new version of the product. The private environment could then be purged of the excess versions or they could be retained for future reference. One facility that would have to be provided to the user would be a means of organizing the personal environment.

In this section we have presented an approach to evolution management that will be offered as one of the key integration services of the DKMS. In the next section, the DKMS approach to integration services in general will be described.

## 10. Product Designer Systems

As a platform for knowledge based design assistants, DKMS will provide design support systems that can assist in producing more nearly optimal designs in less time. The ultimate goal is to have these concepts incorporated into a "design support system" construction shell, i.e. a tool box designed to facilitate the construction of specific design support systems. Such a tool box will ease the development of design support systems able to use coherently a diversity of design information sources accessible through the DKMS. The benefits of design support systems built with this tool box include reduction in the development and modification time for products and systems for the customer. The generic nature of the

envisioned tool box will allow support of commercial product design support systems that would improve the international competitive stance of many U.S. industries.

Design information sources typically used by a human designer of mechanical parts include geometric data, engineering domain principles, material properties, cost data, analytic models, and product life cycle histories, among others. However, current automated design aid systems are unable to access all these information types in any integrated way, and these systems typically have no understanding of the information usage patterns and design development rationale employed by the human designer. One might say by rough analogy that current automated design aids are at the "calculator plus pen and paper" stage, not at the spreadsheet stage.

We have built / are building design support systems using a variety of these components. Such design support systems are individually useful; however, the greatest long term benefit to industry will be gained from developing a tool box of utilities for constructing specific design support systems.

## 10.1. Characteristics of designer systems

Our recognition of the opportunity to develop a generalized architecture for design knowledge management and concurrent engineering support is based primarily on our experience building "knowledge based systems for mechanical design," currently in use at Chrysler Motors for the design of engine box cooling systems, structural fasteners, engine valve train components, air conditioning systems, and cost estimation systems. Our understanding of the issues in the development of such designer systems is also based on research work in life-cycle engineering support for building architects. One important characteristic to recognize about "designer" systems is that they are rarely autonomous. Rather, they tend to provide more of a "design assistant" capability that improves the efficiency and productivity of the human designer. These systems usually provide a "manual" as well as "automatic" mode of operation. In the manual mode they provide design history management and high productivity interfaces to the traditional performance analysis tools. In the automatic mode, they provide services ranging from qualitative assessment (e.g. manufacturability, cost, reliability etc.) of proposed designs to computer-generated and analyzed system designs.

The designer system models both the domain reasoning and the design process reasoning of an expert engineer as he generates initial design specifications. Inputs to such a system typically comprise a description of the environment in which the system will operate from the perspective of the system, (i.e. the environment parameters that constrain the performance requirements on the system), a specification of test conditions (e.g. horsepower, speed, ambient temperature), and certain technical or administratively directed constraints on the system. In the process of producing acceptable design specifications, the system iteratively proposes specifications, tests these using an analytical performance prediction program, evaluates the test outputs, and revises the design specifications. The analytical program models state defining characteristics of the system in the context of the related environment. The Design Assistant does not search exhaustively but uses heuristics gleaned from the expert designer during the redesign step. Concepts derived from our experience with these systems that we expect to incorporate into the DKMS Designer Shell include:

1) The use of existing engineering analysis models as evaluators on a design. An important issue in the use of such models involves the acquisition of the portion of the designer knowledge base that defines how using the system will understand the capabilities, resulting outputs, and particularly the limitations and underlying assumptions of the analysis program.

2) The use of curve-based reasoning as part of the reasoning model. Engineers frequently think in terms of curves (or sometimes surfaces) when relating design parameters to performance expectations.

3) The use of truth maintenance techniques. Mechanical design engineers typically think in terms of alternative design proposals. That is, they will freely make and retract assumptions as the design process proceeds. Truth maintenance techniques are useful in managing the complexity generated by this situation in a complex design history.

4) Highly flexible user presentation. Designers do not take one prescribed path to a solution, but may change viewpoints readily. To support this process, an automated design support system must allow the user to choose to look at various aspects of the problem almost at will and in varying levels of detail.

5) The support for integration with product definition databases, including both geometry and non-geometry based systems.

6) "Special study" design, analysis, and presentation. The fact that a design satisfies requirements and constraints may not be enough to gain acceptance for the design in an organization; rather the design engineer will be expected to demonstrate the design rationale, show why the design is better than other satisfying designs, and answer "why not" questions about alternatives. This implies the ability to maintain the design rationale, to do comparative analysis on test results, to do sensitivity analysis on designs, and to produce a suitable presentation of the results.

7) Use, at varying levels of detail, of engineering models used for performance analysis. A design support system should accommodate system specifications at varying levels of detail, e.g. a component may in one case be modeled as simply an output and in another as a set of parameters manipulated by a routine simulating performance.

## 10.2. Components of designer systems

The principal components of a designer system are represented in Figure 25. These components are not shown in any particular structure primarily because the structure of interconnections is generally left up to the session user. Our experience has been that the cognitive process of the life-cycle engineering activities is a rationalized, exploratory, learning process generating its own internal structure as it proceeds. Any attempt to impose a defined pattern on such a process by a tool may work well for one problem instance and then fail miserably on another (even similar) instance. Therefore, the DKMS approach to this problem is to build from the systems that have been implemented to develop more of an "object" based approach to the development shell components. Under this approach, these components become programmable services in the respect that a "designer" system developer would be able to tune the generic services to accommodate the needs of a particular domain. He could even provide a structured interaction for new users. However, as soon as the users had progressed to the point of understanding the protocols of the interactions between the objects, they could quickly develop their own specialized interfaces. One important note on these components is that, in any one application, there are almost always multiple domains whose knowledge is a part of the services provided by the "Engineering Domain Knowledge

Representation" component. This will be even more the case in the Concurrent Engineering Applications that are anticipated in the DKMS.

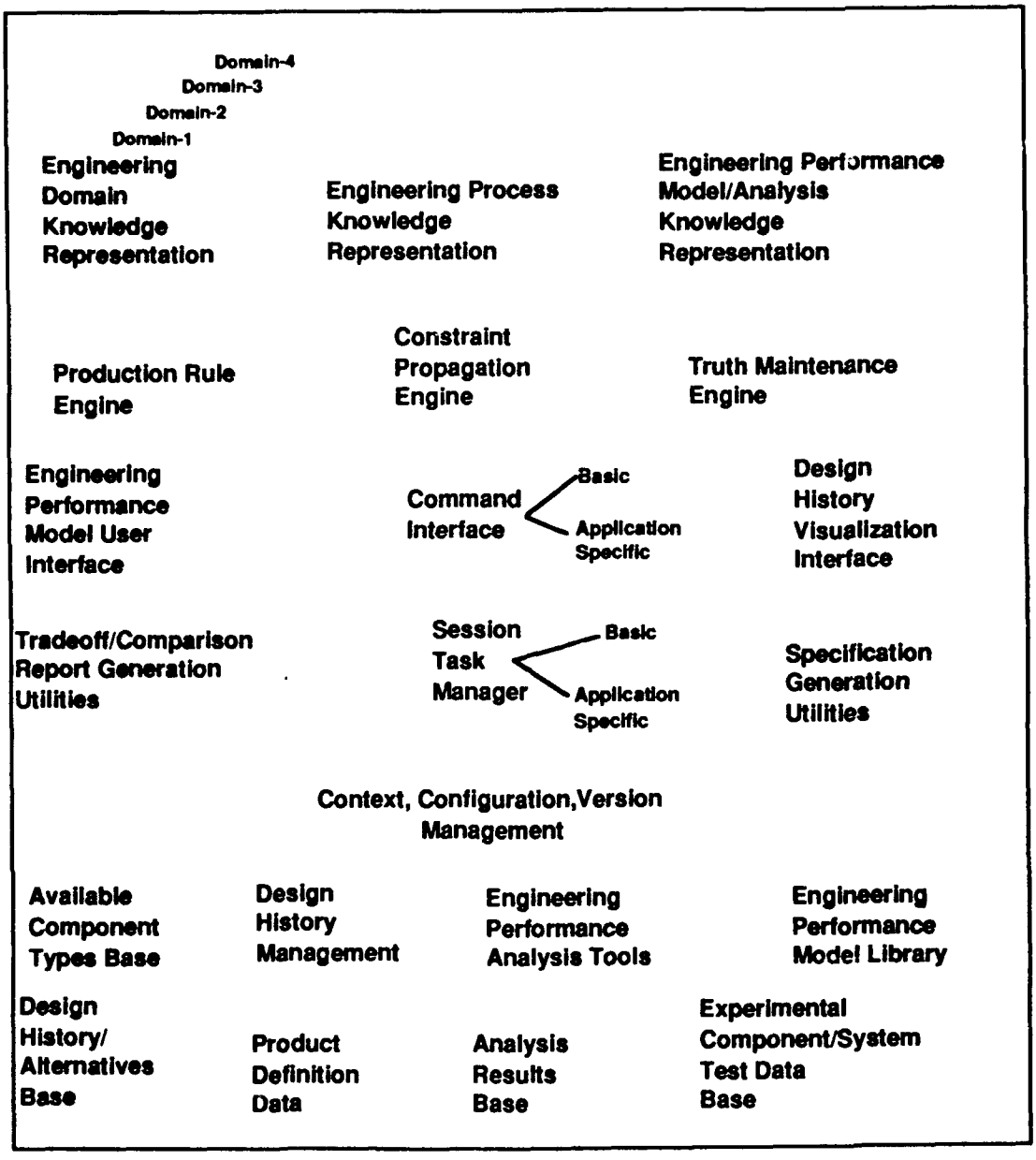



**Figure 25. Generic Component Objects in Designer Systems**

## 10.3. Knowledge acquisition approaches for designer systems

Knowledge acquisition in the life-cycle engineering areas described above can be time consuming and tedious. Of all the traditionally studied problems of knowledge acquisition (internment of the process as well as the domain knowledge, etc.), the most serious problem with knowledge application in the life-cycle engineering domains is the lack of (sometimes any) written corpus of cases that can be analyzed. Part of this problem is due to the previously noted use of "languages of thought" which are often semi-formal symbol systems. These symbol systems carry a rich semantics and generally (from a computerization point of view) a complex syntax. Complexities range from the definition of an entire sublanguage to the use of complex two-dimensional (graphics/iconic based) languages. Part of the problem then of knowledge acquisition in these domains is the problem of assisting the domain experts in a formalization of their languages of thought. Another is the creation of note-making environments that can be mixed into early prototypes of a designer system that human experts can use to capture their comments as they use the system. We have also experimented with the delivery of sophisticated user interfaces to existing engineering performance modeling analysis programs as a first step. These systems offer the human user a direct benefit and hence an incentive for use. Into these systems then we can integrate session monitors (flight recorders of a sort) and the note-making capabilities described above. Once in place, the information collected by these systems can be periodically recovered and studied by the knowledge engineers.

To expect to produce anything other than "crude" automated knowledge acquisition aids is believed to be presumptuous to a high degree over the life of the DKMS. However, there are utilities that can greatly assist the human knowledge engineers in their acquisition, organization, and analysis of the domain knowledge in the life-cycle engineering knowledge bases.

## 10.4. Knowledge application approaches in designer systems

There are four basic approaches to the application of acquired engineering knowledge in designer systems.

1) Heuristic guided search using generate and test methods.

2) Design option generation through backsolving of the performance models followed by heuristic based option ranking methods.

3) Structured selection using constraint tree representation of the design knowledge.

4) Constraint satisfaction supporting a combination of structured selection and heuristic guided search.

The DKMS designer system development shell utilities for control must support these application approaches.

The first approach is illustrated in Figure 26. Each node in that figure represents a possible rule generated design configuration. Those nodes that are connected with links represent possible "follows from" relationships. That is, one node is generated based on the analysis performed on its parent nodes. While there is no requirement for a strict hierarchy in such an application approach, our experience has rarely seen different. It is also important not to misinterpret the illustration as though each generation is based strictly on the results of an analysis of its direct parent. In fact the "next step" decision is generally based on every node generated to that point. As can be seen from the figure, there are nodes in the design space that may not be generated at all by a particular execution of the designer system. This can be the result of the "learning" process that the system undergoes as it solves a particular instance of the design problem (program learning can be as faulty as the human counterpart). The thicker links represent a level of heuristic based "belief" on the part of the "designer system" that pursuit of that particular link is more advantageous (such heuristic-based beliefs can also be erroneous).

The second common knowledge application strategy uses analytic methods and quantified constraints to generate the entire set of acceptable solutions. The design knowledge in these cases usually takes the form of the constraint set or heuristics for rank ordering the resulting set of acceptable solutions. The application of the configuration analysis techniques is reserved for the more detailed evaluation of only the top ranked techniques.

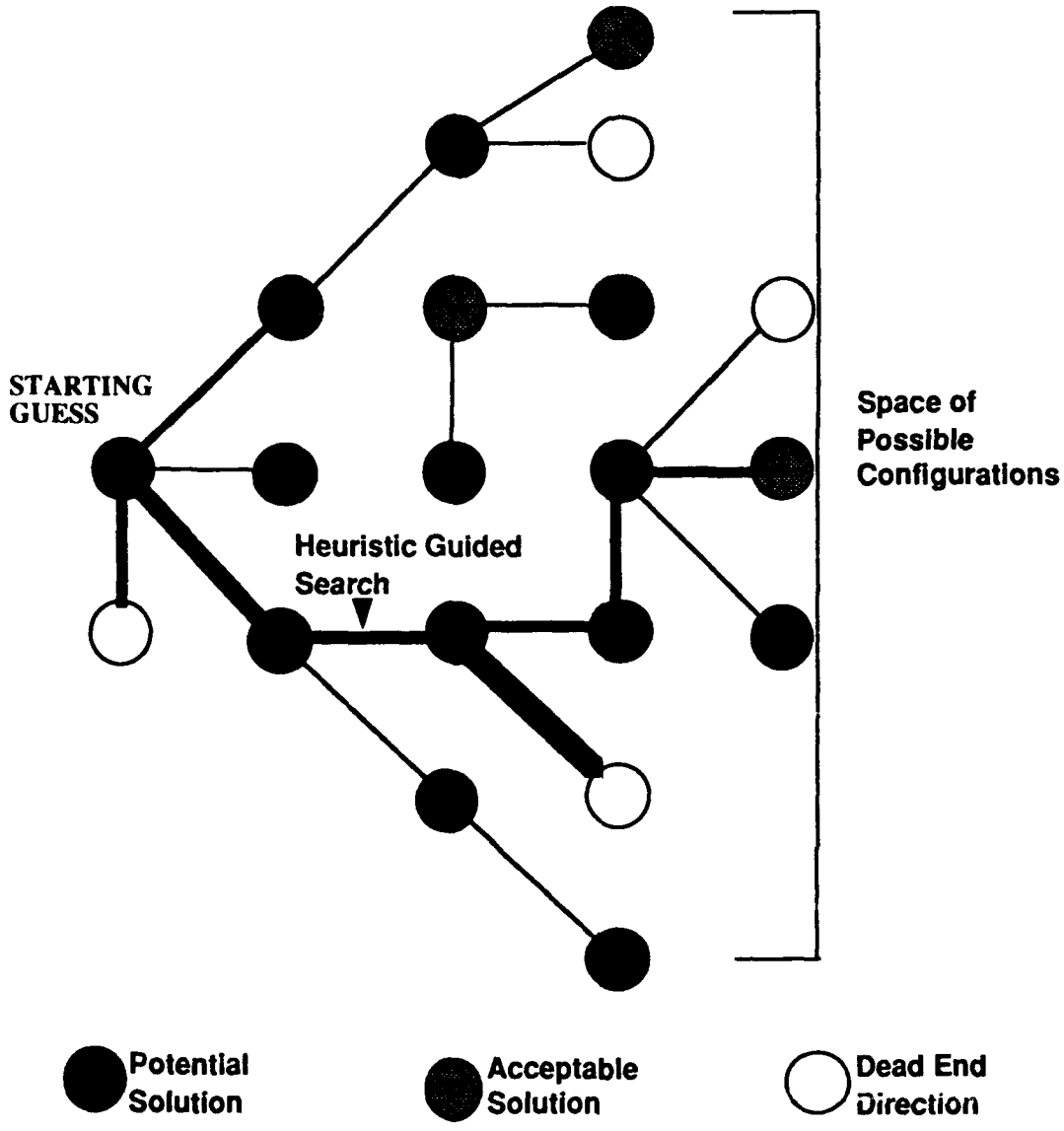



Figure 26. Heuristic Guided Generate and Test

## 10.5. Generic utilities required to develop designer systems

Concepts derived from this work that will apply to the DKMS concept include the following:

1) User interface / composite part representation construction utilities. The automated design of mechanical systems frequently involves a large representation structure for the objects being designed and a

sophisticated user interface for intelligent management of the input of object descriptions. A utility being developed facilitates the rapid prototyping of these interfaces and automatically generates the representation structures (which are accessible either to a rule-based reasoner or to analytic programs) as the input formats are defined.

2) Knowledge-base structures for supporting reasoning based on the identification of situation types in the design process. Frequently a designer will make design decisions based not only on the current state of the designed object and domain principles, but also on the current "situation" as determined by the history of design changes and evaluations to date.
3) Utilities for defining control structures to manage the interplay of numeric components, heuristic components, display and communication components, and so on.

### 10.5.1. Knowledge engineering utilities (representation and reasoning methods)

By far the most common knowledge representation scheme used to date in the designer systems we have constructed is the production rule representation with an associated set of logical consistency constraints managed by an assumptive based truth maintenance system. As described in the previous section, there are serious shortfalls in this area when confronting knowledge domains which have extensive geometry concepting or recognition components. In these situations, the only options in the past have been to:

1) Develop linear language encoding (names or descriptors) for the "20%" of the commonly occurring or most important shapes and use the traditional symbolic processing facilities.

2) Hard code sophisticated (but generally limited) geometry processing and reasoning mechanisms (as in the AI CAD/CAM system described below).

3) Avoid the problem as too difficult for current methods.

With the addition of the shape representation / reasoning capabilities of the Generalized CSG and the container objects, the option will exist for the direct representation of shape indexed knowledge representation.

The DKMS will also provide support for the traditional production rule and belief maintenance facilities as a part of the DKMS. The production rule facilities will be based on a Rete net processor with additional language facilities that we have found essential to the construction of designer class systems including:

1) Seamless integration with the host language environment.

2) Rule language facilities for both formulation of relational and object oriented database queries.

3) Rule language facilities for the convenient processing of the results of such database queries.

4) Rule language and processing facilities for matching against objects in the execution domain without the direct incorporation of those objects into the working memory of the rule processing system. This capability avoids the problems of data redundancy and currency control.

Three types of belief maintenance facilities will be supported in the DKMS including a) justification-based, b) logic-based, and c) assumptive-based capabilities. In addition to the standard first-order-propositional logic support in the logic-based truth maintenance facilities, label-based numeric equality and inequality support will also be supported.

### 10.5.2. User interface construction utilities (visualization of the design history)

Based on our experience in the development of designer systems, the sophistication, intelligence and flexibility of the user interface is most often the determining factor of the acceptability of the system to the target end user. Besides the familiar desktop, windowing, menu, mouse, and presentation manager capabilities, the DKMS tool kit will provide for the development of specialized visualization support for:

1) Browsing a visual representation of the design history.

2) Multiple simultaneous display of text/tabular results of the execution of performance analysis runs.

3) Ad hoc specification and simultaneous display of multiple graphs and charts for comparative visualization of the results of execution of performance analysis runs.

4) Visualization of the base of components, design sessions, rule sets and other information, knowledge or processing resources to support the end user management of these resources.

Our current approach to the provision of these facilities is based on the extensive use of the New-Flavors objects and the Symbolics Presentation and window manager systems. During the course of Phase II the CLOS (Common Lisp Object System) and CLIM (Common Lisp Interface Manager) capabilities will be reaching maturity and we plan to convert to these utilities to allow delivery of the resulting capabilities on a wide variety of platforms. We are also continuing to monitor the progress of the AFHRL IMIS model based user interface strategies as a design philosophy for the underlying architecture for the user interface construction utilities [Gunning 89].

### 10.5.3. Engineering artifact management

The ease of use of engineering designer systems, the magnitude of the possible design alternatives and choices that are generated, to say nothing of the volume of complex interlinked data resulting from the online rationale capture, make issues of configuration, version, and evolution management a critical element in a successful DKMS. To provide the capabilities to generate such a rich environment without the corresponding assistance in managing the resulting complexity is an invitation to disaster and disappointment. The DKMS evolution management services described in this report are sufficient (based on our experience) to keep track of this complexity. However, additional design work on the needed summarization, visualization, and knowledge-based support requirements to realize a truly effective man-machine interface for addressing this problem must be pursued in Phase II of this project.

## 10.6. Support for interacting / integrated designer systems

One of the implications of the establishment of a DKMS capability in an enterprise is the inevitability of the desire to construct not only individual designer systems but also federations of interacting cooperating designer systems. The integration services provided by the DKMS provide many of the needed base capabilities for construction of such cooperating systems.

However, there are several specialized utilities beyond those integration services that address problems of the nature of modeling the negotiations and compromising associated with team engineering of complex systems. The DKMS designer shell must provide support in the following areas to enable the construction of this class of truly simultaneous engineering applications.

1) Facilitator construction.

2) Decision scenario specification and implementation.

3) Blackboard construction.

Facilitators are knowledge-based components that support the interaction of a primary designer with the other primary designers in a cooperative design scenario. The role of the facilitator is to relieve the designer of the need to understand the complexity of the cooperative design process, thus allowing reuse of the primary designer in many different cooperative sessions. The facilitators' knowledge base includes knowledge of:

1) The decision process.

2) The other agents involved in that process.

3) A model of its primary designer's activities.

4) Negotiation procedures for the critical state variables that define the coupling between the components being designed.

5) Error / failure mode diagnosis and recovery procedure knowledge.

The facilitator concept has been successfully implemented in complex cooperating systems involving over 15 knowledge based agents with critical fault tolerant capabilities [Mayer 87]. However, the design of such agents can often be as complex as the primary knowledge sources they serve. Hence special support is required in the DKMS development shell for decision scenario specification, blackboard construction, and knowledge source analysis.

## 11. Design Knowledge Application Support--(Advanced CAE/CAM)

To the concurrent engineering market, the proof of viability of a concept is in the actual demonstration of capabilities that escape their current grasp. We propose that along with the information integration platform, CAD innovations and knowledge representation schemes will be investigated. We will assemble three demonstration capabilities in the following areas:

- Manufacturability checking,
- Process planning based manufacturing cost estimation,
- Engineering modeling support.

These capabilities will be built from stand-alone applications which KBSI already has under development in each of these areas. KBSI's philosophy in designing the engineering and manufacturing support systems has been to develop systems that serve in an advisory role to the human expert so that the system will accommodate the expertise of the user. With respect to the process planning system of AI CAD/CAM, the goal of the system is to effectively capture the knowledge of the process planner so that the tedious and repetitive procedures (i.e., NC code generation and verification) can be automated. Achieving this goal will free the expert to experiment with new tool and fixture designs. In other words, the expert is able to devote more time to the more creative, and potentially beneficial, aspects of process planning.

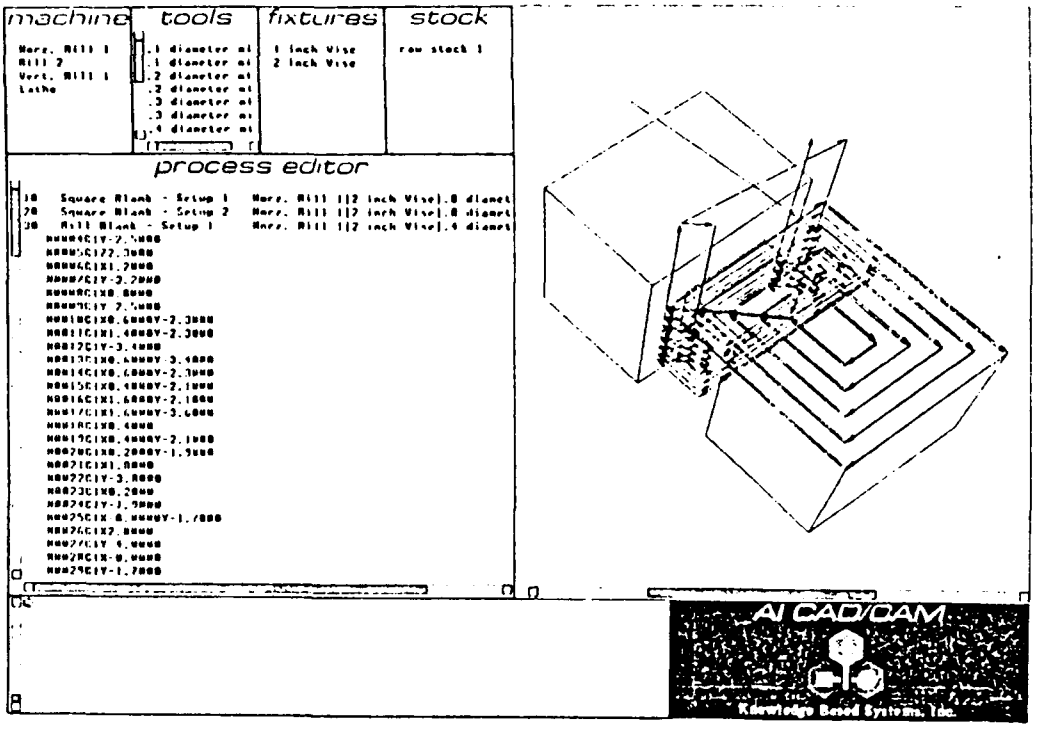



**Figure 27. AI CAD/CAM Process Plan Editor**

Figure 27 presents a view of the Plan Editor utility of AI CAD/CAM. This interface allows the user to edit or browse through the computer generated plans, while the Graphic Interface Manager provides visual cues in the form of three dimensional models of the state of the plan the operation on which the user has positioned the cursor. In the example shown in the figure, the user is viewing a tool path associated with the NC code generated for a particular operation in the process plan. Visual aids like this, as well as other plan analysis features of the system, assist the user in determining if any modifications must be made to the process plan. If a plan is modified, the new plan is resubmitted to the process planner to determine if there is some reason why the modifications should not be allowed.

## 11.1. AI CAD/CAM planner concept of operations

Key to the AI CAD/CAM philosophy in process planning support is the division of the problem into multiple interacting and overlapping major components that mimic the functionality of the human planner. The functions duplicated include:

- Understanding the finished part requirements,
- Manufacturability assessment,
- Process Plan generation,
- Process Specification generation,
- Plan verification,
- Plan simulation.

The process planning system of AI CAD/CAM accepts solid model files and transforms these into an object oriented half-edge and Binary Space Partition Tree (BSPT) data structure. The decision support geometry reasoning tools used in the determination of process suitability are centered around the generation of convex enclosing objects, prismatic decomposition, and orthonormal visibility analysis. The enclosing objects include the convex hull, smallest enclosing box, and principal orientation box of the part. If no single machine process is available for machining the set of prisms and residue elements of the part, a structured minimal representation of those elements is handed over to the planner system. Otherwise, the opportunistic nature of the AI CAD/CAM planner immediately proceeds to the process specification stage, where the tool approach angles and milling planes are then analyzed, and a process specification is determined based on the minimum number of setups needed. The system then generates NC code for machining the part using the maximum volume removal criterion to consolidate the tool passes over the feature space.

Figure 28 shows the current level of implementation of the generative component of the AI CAD/CAM concept from a networking/hardware viewpoint.

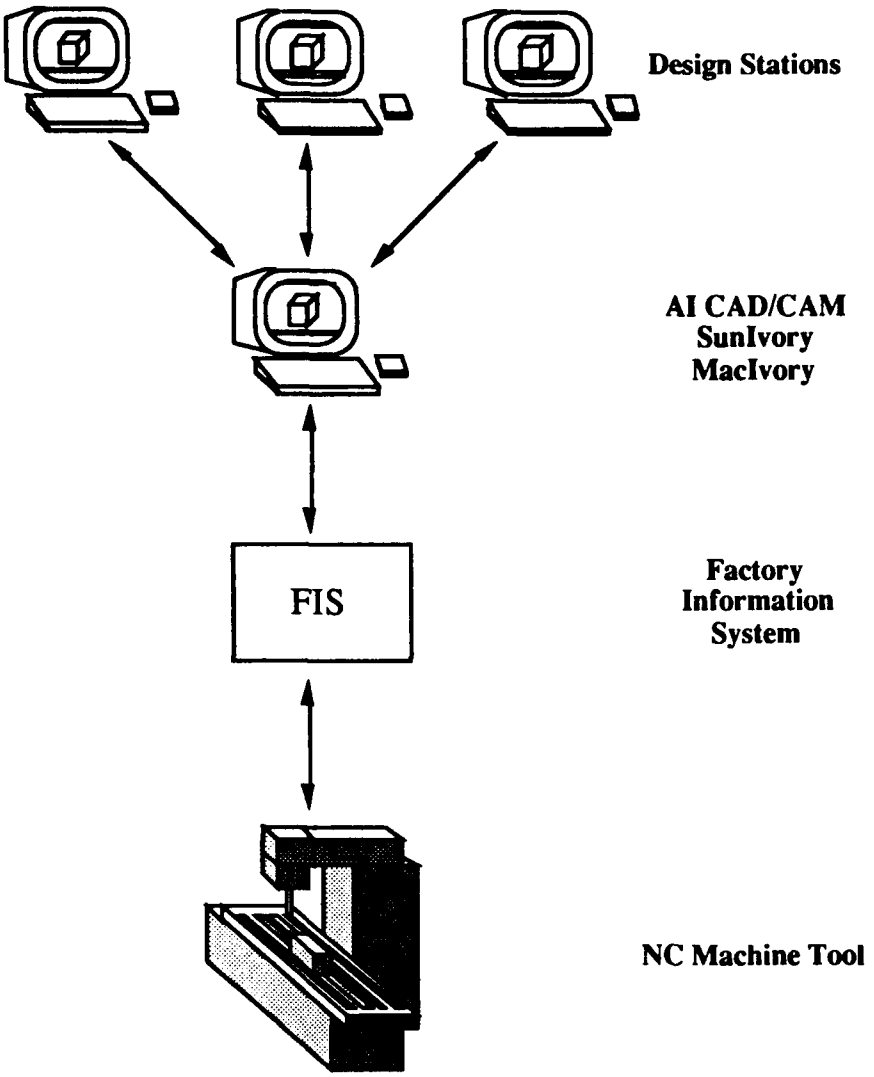



**Figure 28. Current Implementation of the Generative Component of AI CAD/CAM**

This planning capability demonstrates the importance of the DKMS integration facilities. The information and knowledge necessary to produce an effective process plan comes from many sources. Initially, the prism generation and analysis required to fully understand the part in question are performed through interaction with the Generalized Constructive Solid Geometry. To produce an effective process plan requires product data definition information to be available on the part as well as on machines, tools, and fixtures. This information is maintained in a variety of manufacturing databases and is accessible by the planning system through the Common Data Manager. Without means of effectively integrating all

this information, the capabilities of the process planner, as well as other portions of AI CAD/CAM, would be severely limited.

## 11.2. General algorithms and approach

In duplicating the six functions normally performed by the human expert, several interacting utilities had to be developed. To fully understand the part in question, a prism generation and analysis system was built. The geometric reasoning capability built into this system allows the planner to know what material must be removed from the stock material. The prism analysis also provides the capability to relate or link prisms together in an intelligent fashion so that operations in the process plan may be combined. By viewing the prisms from all possible machining directions, the system can determine if every prism is accessible by the spindle from at least one of the machining directions.

Once the geometric processing is completed, the information is passed onto the process planner. This utility is a rule-based system that produces a sequence of operations that will remove the necessary volume from the stock material. The rules in the system incorporate the knowledge of the expert and enforce the practices to be followed in producing a process plan. Throughout the plan generation, the planner is advised by a set of domain specific critics. The tolerance critic, tool critic, setup critic, and machine critic all provide the plan generator with feedback on how "good" was a choice made by the plan generator. The planner then stores this feedback from the critics in a truth maintenance system network. This network is used to prune the search of the plan generation by preventing choices from being repeated. In addition, the network provides a line of reasoning from which an explanation of the decisions made by the planner can be derived.
The next component of the system is the process specification. It is in this unit that the NC code generation is performed. However, this unit is capable of modifying the process plan so that more efficient tool paths may be produced.

Finally, using solid models of the tool, spindle, fixture, and stock, the NC code undergoes verification. For the entire plan, the tool path is followed to determine if any invalid collisions occur between the objects. The only valid collision is one where the tool intersects with the stock material during a cutting operation. The intersection is calculated first by generating the swept volume of both the tool and spindle for a tool

movement. The following sections present more detail on the components of the process planning tool of AI CAD/CAM.

### 11.2.1. Manufacturability determination

The algorithm used for machinability checking utilizes the properties of point visibility, point orthonormal visibility, facewise complete visibility, facewise partial visibility, and solid visibility from the convex enclosing object to determine the suitability of a part to a process. Since there are infinite points on the boundary of a part ***S***, it is not always possible to check the visibility of all boundary points. In practice, facewise visibility, instead of pointwise visibility, is checked for machinability. That is, if every part face can be visible from some enclosing object face, then the part is acceptable as visible to a three-axis machine with one repositioning axis. A Machinability Theorem was developed based on orthonormal visibility, providing a mathematical foundation for machinability. Using the Machinability Theorem in conjunction with practical constraints on the available machining tools and machine, a cutting strategy can be derived.

### 11.2.2. Prism generation and prism analysis

After generation of the prisms, the system enters the prism analysis phase. It is at this point that distinction is made between residual prisms (those prisms that represent material to be removed from stock) and part prisms (those prisms that make up the finished part). The analysis then concerns itself with only the residual prisms and arranges a hierarchy of prisms for each of the possible machining directions. At the lowest levels, these hierarchies consist of the residual prisms. But as the tree is traversed upward, each level represents a union of the prisms at the next lower level. This analysis allows the system to determine if any relationships exist between prisms visible from a certain machining direction. Should these relationships exist, the planner would be able to take advantage of this to produce more efficient and effective plans.

### 11.2.3. Process plan generation

The generative planning problem can be formulated in the following way:

**minimize**(tool changes, setup, fixturing)
**subject to**
    **Constraints**
    Machinability Theorem
    Physical Constraints

Coordinate Constraints
Tolerance & Dimension Constraints
Shop Constraints
Tooling Constraints

The CAD interface, along with the machined attributes extraction facility and the primitives required for the machinability checker and the NC code generator provide a solid platform for the development of the Master Planner. Using the information from the Minimum Enclosing Box (MEB), the 3-D Convex Hull, and the Principal Orientation Box (POB) calculations, we can determine hidden faces and optimal machining directions as well as fixturing orientations. With this logical information we can then view the problem of plan generation as a two-phase planning process. The first phase involves the selection of a range of operations for accomplishment of the required material removal and a set of machine tools for each operation. This phase of the planning process attempts to make local or static decisions and record the options available at each decision point. This first phase applies the expert planner's knowledge of which types of machining processes can be used to remove material of various forms at particular rates and with certain types of tolerances.

The next phase in the planning process starts with the generation of an implicit sequencing of the operations. We are using an implicit sequencing approach to avoid the combinatorial explosion inherent in the possible number of sequences which could be generated for any part. The second phase begins to apply the knowledge which the human planner has acquired concerning how operations and plan segments can be assembled into a coherent plan. The internal representation of the coherent plan will take the form of two directed networks of nodes. Each node in the plan network will contain a plan step or a group of plan steps. Each arc in the plan network will represent a set of constraints between each pair of plan net nodes. Constraints are recast as sequence constraints to enable the use of temporal logic.

e.g.: Drill-hole-a must be before during or meets taphole-a

drill-hole-a (b d m) taphole-a
mill-face-b (b) drill-hole-a

Associated with each arc and node in the plan network will be a node in the second network which record the rationale or justification for the plan network element. This second network maintains the justifications or

assumptions which must be true for a plan node or arc to be included in a valid plan.

Taken together these two networks define implicitly all the feasible plan structures. The generation of an instance of a plan would result from a traversal of the plan network extracting a set of nodes with valid justifications. To maintain plan validity, the three main truth maintenance paradigms are used, namely assumptive, logic, and justification based. For the Phase II DKMS planning system, we plan to use an assumptive-based paradigm for the overall planning strategies, a justification-based system for the actual planning, and a logic based system to critique the plan.

The master plan is represented as a set of Plan periods, consisting of operations.

For example:

**(MASTER PLAN**
(Plan Period 1 (operation 1)
(operation 2)...

(operation n))

(Plan Period 2 ... )
.
.
.
(Plan Period M... ))

A Plan Fragment is defined as a subset of a Plan Period. This structure is important for both the Plan Editor / Browser and the actual execution of the Plan to allow plan fragments to be rearranged because the environmental constraints have changed or the user decides to intervene.

### 11.2.4. Process Specification

#### 11.2.4.1. NC Code Generation

Before NC code is generated, the set of tool approach directions for the features are determined. By minimizing the number of set ups, a subset of these tool approach directions may be selected for NC code generation.

The following algorithms have been implemented for generating the NC code.

a) Global Tool Path Generation for roughing
b) Local Tool Path Generation for roughing
c) Local Finishing
d) Tool Path to G-Code translation

#### 11.2.4.2. Global Tool Path Generation

The Global Tool Path generation is used for cutting planes perpendicular to the milling plane. For example, it can be used for cutting generalized pockets with walls parallel to the milling plane, and floor perpendicular to the milling plane.

#### 11.2.4.3. Local Tool Path Generation

The Local Tool Path Generation is used for cutting faces that are not perpendicular to the milling plane. For example, a pocket with an inclined floor can be decomposed into a global operation and a local operation. Figure 29 illustrates the machining of an inclined plane.

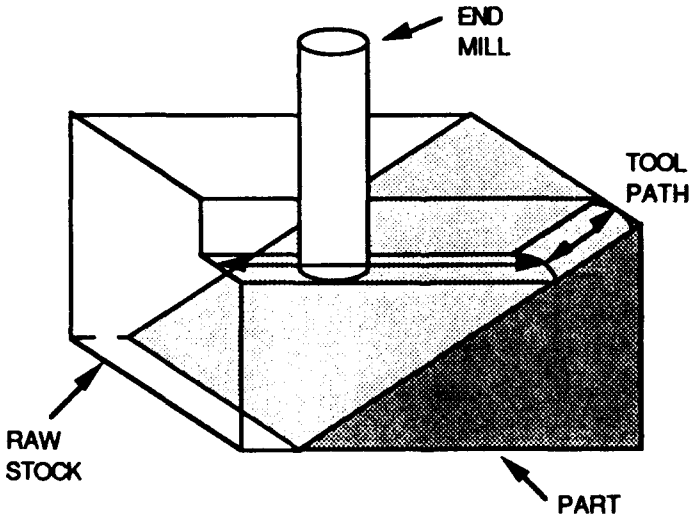



**Figure 29. The Machining of an Inclined Plane**

#### 11.2.4.4. Local Finishing

Local finishing is used for milling out the roughing steps left by the local tool path operation. Because the operation is taking place on an inclined surface, care must be taken to ensure that the tool passes are close enough

together so that the grooves that will be formed are within the surface finish tolerance specified.

#### 11.2.4.5. Tool Path To NC Code Translation

The tool path tours that have been determined by the procedures described above have to be translated into a format compatible with the NC Machine. Thus, the tool path is translated into low level NC code, which will control the milling machine.

### 11.2.5. Process verification

In the machine shop, process verification requires a test run of the NC code using a wax stock to determine if any collisions would occur between the equipment and the stock or fixtures. If any problems occur, the NC code must be modified and another verification step performed. However, with AI CAD/CAM, this verification process is run online so that no test runs need be performed.

Initially, every object used in the machining process (tool, tool holder, spindle, stock, fixture, machine bed) is represented as a solid model. After every movement of the tool/tool holder/spindle arrangement, a solid model representing the swept volume of each of these objects is calculated. Then, with the set operators provided by the Generalized Constructive Solid Geometry, the swept volume solids are intersected with the stock, fixture, and bed models to determine if any collisions occur. Of course, the situation would determine what collisions would be tested for. For example, the tool can only collide with the fixture or bed or with the stock on a fast tool movement (G0 movement). If a collision is detected, the user is notified that the NC code is invalid and the user can then make appropriate adjustments.

### 11.2.6. Process simulation

In a similar fashion to process verification, the tool cutting can be simulated. Once a tool movement is determined to be valid (no collisions) during the process verification, the model of the tool swept volume is subtracted from the model of the stock material. From this operation, the simulator is able to present an intermediate representation of the stock as the NC code is being processed.

This facility is used in conjunction with the Process Verification. Because operations on the solid models are expensive, as the process is being

verified, intermediate information is written to a file so that the simulation can be played back later. This allows the simulation to run quickly and allows the same simulation to be played multiple times.

This off-line simulation also indicates the importance of the Generalized Constructive Solid Geometry. The process verification step requires the solid models to be represented as Binary Space Partition Trees (BSPTs). The BSPT is a complicated data structure. By storing key information off-line, the process simulation is able to use the standard BREP to regenerate the objects in the simulation. The GCSG allows the components of the system to use the representation that is the best for those operations.

## 12. DKMS Platform Architecture

To provide the type of information and knowledge management support and control required, KBSI is proposing a platform architecture based on a services approach to integration. This represents a new way of looking at the integration problem. In essence, we are suggesting that rather than focusing on the construction of an "integrated system" we should focus on the "integration services" that the DKMS platform as well as the functional applications will provide. In previous approaches to integrated systems architectures, the burden was assumed to be on the platform to provide the integration support desired. The prevailing wisdom on achieving concurrent engineering integration within an organization is that it occur in three ordered waves:

a) Data Integration,

b) Application Integration,

c) Function Integration.

For this to occur, comprehensive standards have to be set up a priori and universally accepted by the organization. Thus, applications would have to meet these standards before being accepted by the organization. These traditional approaches have severe problems (which is probably why after 11 years there are still no significant applications of this approach). One of those problems is the need to define a priori the comprehensive standards. This presumes that an organization (or a group of analysts within that organization) can foresee (a) the integration services required and (b) the relative demands for those integration services. Presuming that we could solve (a) with a yet to be discovered appropriate "crystal ball" without

knowing the answer to (b), the ISO style integration approaches force an equivalence of integration support across all needs. This implies a massive overhaul of existing legacy systems and unjustifiable modifications by vendors of their existing systems to achieve even a minimal level of integration support. Thus, this "socialistic" approach just won't work. What is clearly needed is a "capitalistic" approach where such services can be incrementally introduced as the user demand forces suppliers of the needed services to emerge. The key is the establishment of the appropriate guidelines and structures for the service contract specification, service protocol, service advertisement, and contracting so that (a) once the expense of setting up a service has been incurred that service is available to all subscribers and (b) the service brokerage evolves in an organized fashion.

Under the "services" concept, the platform focuses on advertisement, specification, and facilitation of "integration services." In the simplest realization of this concept, an application (or user) could request information services. Such requests would be advertised across the "services" network. Applications capable of responding to the services request would, based on availability, bid to provide the service. This simple approach is much like the Information Automat concept proposed by Max Wilson [87]. Our approach is made possible by the advances in both the conceptual understanding and implementations of the object oriented paradigm (Wilson's concepts were based on transaction processing paradigms). In fact, many of the capabilities of the "Integration Mechanism" component of the ISO genre of integrated architectures would be carried over under the notion of a planner (or general/systems contractor). Such a mechanism would allow the generation of a "service request" plan (or proposal) which would essentially be a design of a means to service a complex (or unusual) request by employment of a number of advertised services. An important characteristic of this approach is that the integration planner only need be involved with those more complex service requests. This is in contrast to the CDM (conceptual data model) processor approach of the ISO. In fact, there would be nothing preventing applications (or users) from serving as their own general contractors (i.e. having a hard coded proposal as a part of the application code) running only the risk of not knowing what are the latest services available.

It is our thesis that the integration of an engineering application into a concurrent engineering platform should be viewed as an opportunity to provide greater functionality to the system by providing new services and resources. The opportunity to incorporate new services into a system must be addressed at three classes of services:

a) Passive services {Data, Information, Knowledge}

b) Active services {Application, Function, Inference }

c) Control {Configuration, Versioning, Context }

As these classes do not have strict ordering, vendors may use their discretion to advertise services in terms of generic capabilities that can be rendered by their applications. Thus, an object may issue a service request to a Service Integration Manager (SIM) that will post the request(s) directly to blackboards serviced by domain brokers. The brokers put together service package proposals by assembling resources and methods, and costs by broadcasting requests to a network of applications in their domain. These domain brokerages can include network service, data service, version control negotiation, geometry, etc. The brokered service package includes a time window in which the services can be initiated. There are at least two varieties of bid and proposal support that must be provided by the SIM. The first variety is referred to as the verified sources variety that would be implemented in a fashion similar to the ISO CDM processing scheme using a directed message passing paradigm. The second variety (referred to as "open bidding") would employ a stock exchange (blackboard) style approach. The SIM then selects a package from amongst the service proposals based on criteria such as duration and cost and initiates the service. Figure 30 displays the operation of the SIM.

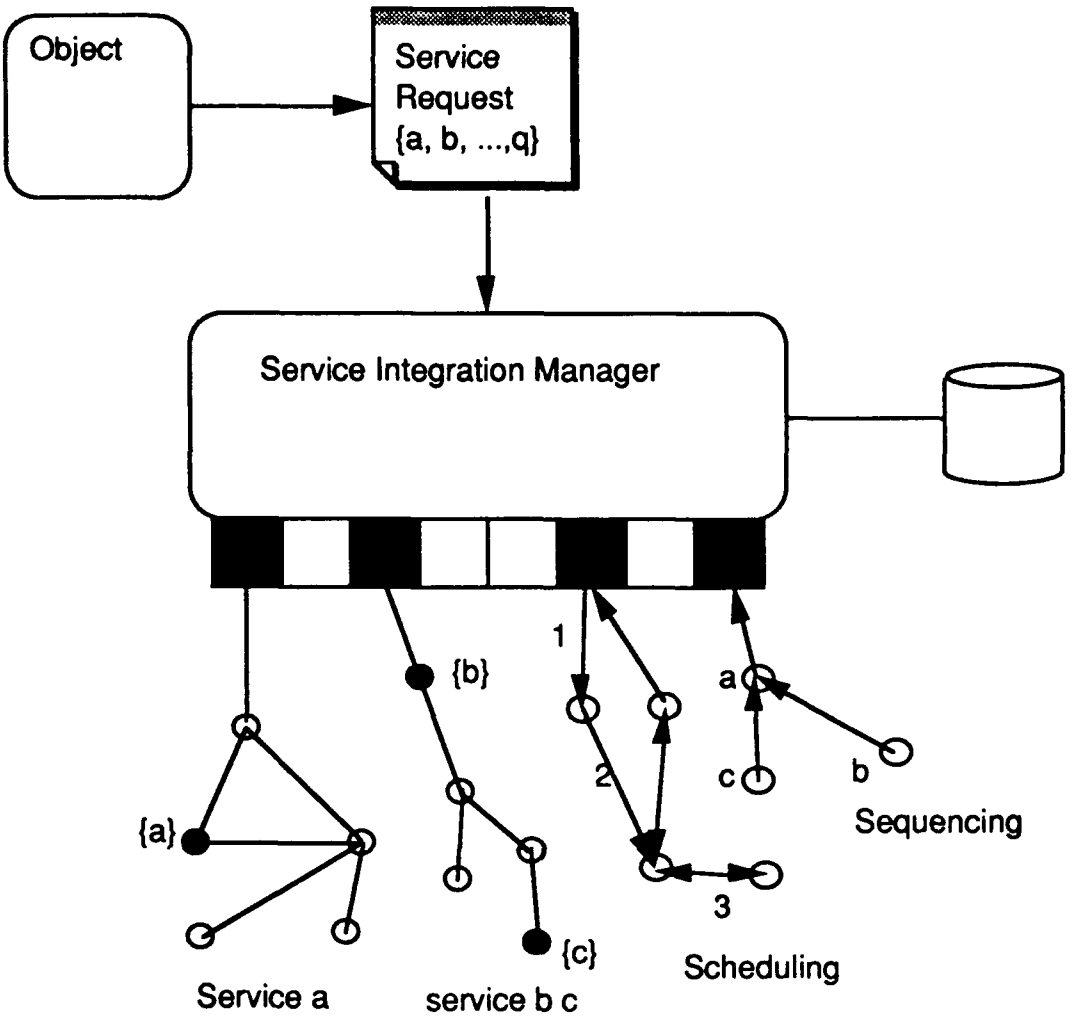



**Figure 30. Service Integration Manager**

The Planner components of the SIM would be based on the following three premises:

- Knowledge Driven Service Support Planning
- Description Based Information and Knowledge Integration Mechanism
- Object Level Life Cycle Data and Life-Cycle Knowledge Evolution Control

The first premise, Knowledge Driven Service Support Planning, is simply the requirement that the planning component of the integration services

system have an understanding of the processes that it will be planning services for (and with). The information integration component of this knowledge base will be built up from an IDEF3 specification by the managers and technical leaders at a site. This resulting knowledge will then be loaded into the system platform. With this knowledge, the system can provide the administration of the design data development process (as well as the life-cycle engineering knowledge evolution process) that was described above. The knowledge base integration component of the DKMS will be described in a similar fashion (IDEF3) but the source of these descriptions will come from the design experience and rationale capture mechanisms as well as from experience captured from the manufacturing, maintenance, and operations personnel.

The second premise is that the system must have description-based integration service mechanisms. Information integration is necessary to provide complete and accurate access to (and control of) the life cycle data of the software system. The number and variety of tools used in the design activities throughout a product life cycle makes an integrated homogeneous (i.e., one supplier, one hardware platform that does all) environment impossible. Instead, tools must have the ability to access data and information that are created by different tools. In addition to the ability to provide active information sharing between tools, information resources for making complex scoping, design, and change management decisions must be supported. This implies the provision of ad hoc search and query support with both browsing and user defined presentation generation (reports, hypernotes, illustrations, etc.). However, we must accommodate an evolution to this goal. With the integration services approach we can initially support design tool interfacing and as the demand dictates and the vendor accommodates, evolve to more functional levels of support. For example, initially we may have to convert an entire CAD file from one system to the other to query for information about a particular model. If, however, the users only require answers to directed questions (such as "what is the normal vector at a particular point"), then the vendor could provide this capability as an exported service in later releases of his product. To accomplish this, support service descriptions must be defined for each vendor or legacy tool as well as information descriptions for those life cycle artifacts developed or managed outside of a particular tool. These descriptions must be used as an active component of the advertisement, planning and contracting elements of the service environment. The integration services planning mechanism will then have the ability to use these descriptions to support access/control of the data produced by a specific tool. This mechanism must also have the ability to recognize

which object is being accessed so that the appropriate framework constraint can be applied.

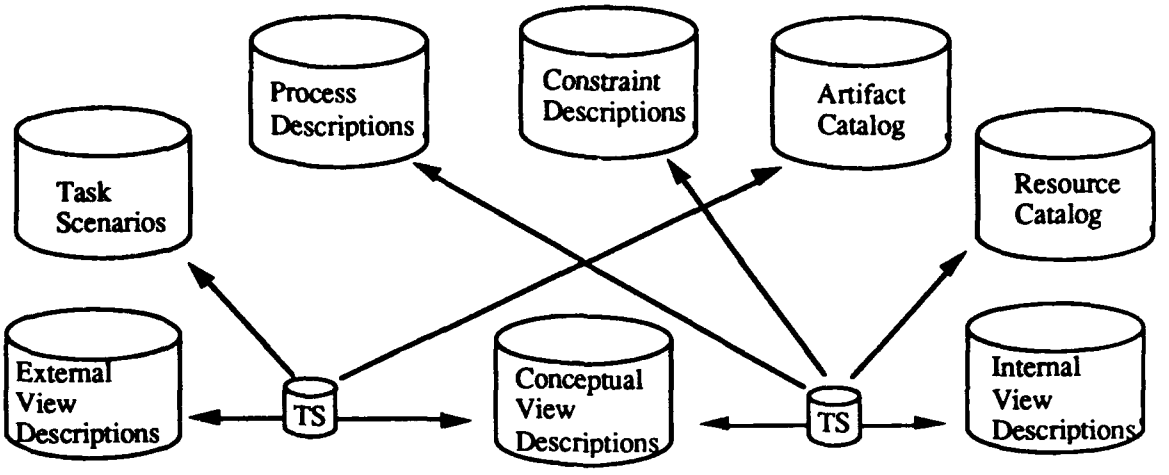



**Figure 31. Multi-schema Architecture Approach**

This information integration planning support provided by the integration services component in the DKMS will be accomplished using a multi-schema architecture implementation of the ISO conceptual schema standard as seen in Figure 31. The schemas in addition to the three standard schemas depicted in that figure are knowledge bases that define the life cycle information managed by the proposed framework programmable platform. They not only specify what data exist but also how to access the data, what constraints exist on the data, and how the data is/can be used. This complete description of the information making up the system is a major means of providing the type of control that the programmable framework system will give.

Effective integration support depends on the third premise of Object Level Life Cycle Data Evolution. Previous integration and technical data management/control strategies have taken one of two approaches:

1) Data translation standards from one format to another.

2) Object definition and control at the file level.

Both of these ideas can be supported by the integration services approach. Data translation standards can be set up initially to enable application / tool interfacing. A preplanned communications link between two tools can easily be established. The result of the traditional integration architectures (which supported such links) was that any tool that was to be integrated

must have these communication links set up. Using the planner services of the DKMS, established communication and translation services will be able to be combined into powerful new interfaces on demand. This approach delivers both the performance and the appearance of integration without radical modifications to the vendor programs and without a combinatorial growth in interface program development.

File level control (while an appropriate first step) is not fine grained enough. For example, in a system design, it is often useful to refer to a specific requirement in the system requirements document. In an integrated system that operates at the file level, this reference could only be made to the requirements document. If this requirements document were changed, the entire document would have to be parsed to determine if the change affected the design. However, the services approach will provide intelligent support to this traditional approach as required.

As Life Cycle Data and life-cycle engineering knowledge begin to be represented and controlled at the object level, these two methods would be obsolete. No format translation methods between tools would be required because the integration services mechanisms would have the ability to access all versioned objects directly. The second problem of having to parse the entire file would be eliminated because a notification would be sent only when the specific requirement object referenced by the design document when modified.

Clearly, this is the best way to proceed. However, it requires that the tools and utilities that make up the design support environment be based on these premises. The scope of this project is to address these issues of a platform for a federated system of knowledge based, data based and analysis applications. The result will be a system that incorporates these technologies to provide a suite of integrated CAE tools along with the knowledge based data administration and seamless configuration control capabilities.

Figure 1 of this report displayed a traditional view of the architecture that will be pursued in the development of the prototype DKMS. That is, it illustrates the formation of an "integrated" system from a set of integration services. The lower half of the diagram illustrates the internal components of the Information Integration Services and the Knowledge Management Services system. Above that is the Dynamic Object Manager that provides the necessary local services to access both the required elements of concurrent engineering knowledge and the product life cycle data at the local object level. Then, in the center of the diagram is the the GCSG and

the container object system. These service components support the backbone of the knowledge representation and shape based reasoning capabilities required of the DKMS. The remaining components of the architecture are typical tools and utilities that a design development environment should provide.

The power of the "services" approach to integration is better illustrated by the introduction of legacy (or existing commercial) engineering CAD based applications into the DKMS platform. Figure 32 illustrates the architecture of such a typical application [Krause 89]. The view taken in the construction of this illustration is that there are core reusable functions that are integrated together into the application by the construction of application specific interface layers on top of the functional levels. Under the "services" approach to integration, the level of integration of such a system will be largely defined by the level of access to these underlying functional methods that a legacy system provides.

The components of a typical Engineering Application

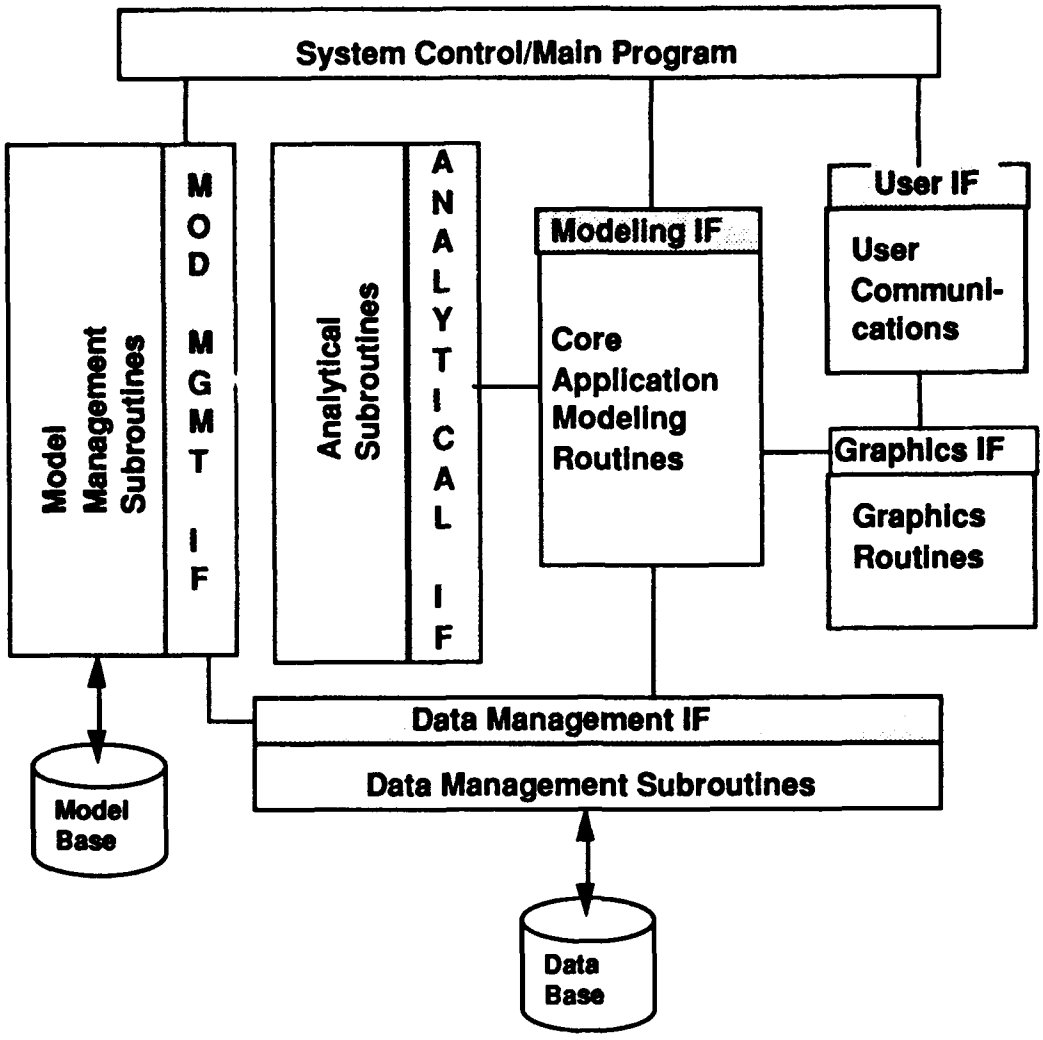



**Figure 32. Legacy System Architecture**

Figure 33 illustrates the assimilation of such a legacy system into the DKMS. The primary determiner of the level of passive, active, or control services provided by that system will be determined by the number of service contracts that the system can satisfy through the advertised protocols of those services. It is clear that, initially, a legacy system may not offer any services whatsoever. Its existence in the system will be justified solely on the particular function that it provides to the direct users. In fact, in some cases this may be the terminal stage of such a package. However, in the fiercely competitive world of CAD/CAE/CAM applications, such an isolated tool has a low probability of survival. It is

likely that at least data access services will be requested of the application. The opportunity offered by the DKMS is that these integration requests can be handled on a case-by-case basis by modular extensions (essentially additional interfaces to the underlying generic functions of the application).

Services management unit layered on top of engineering application

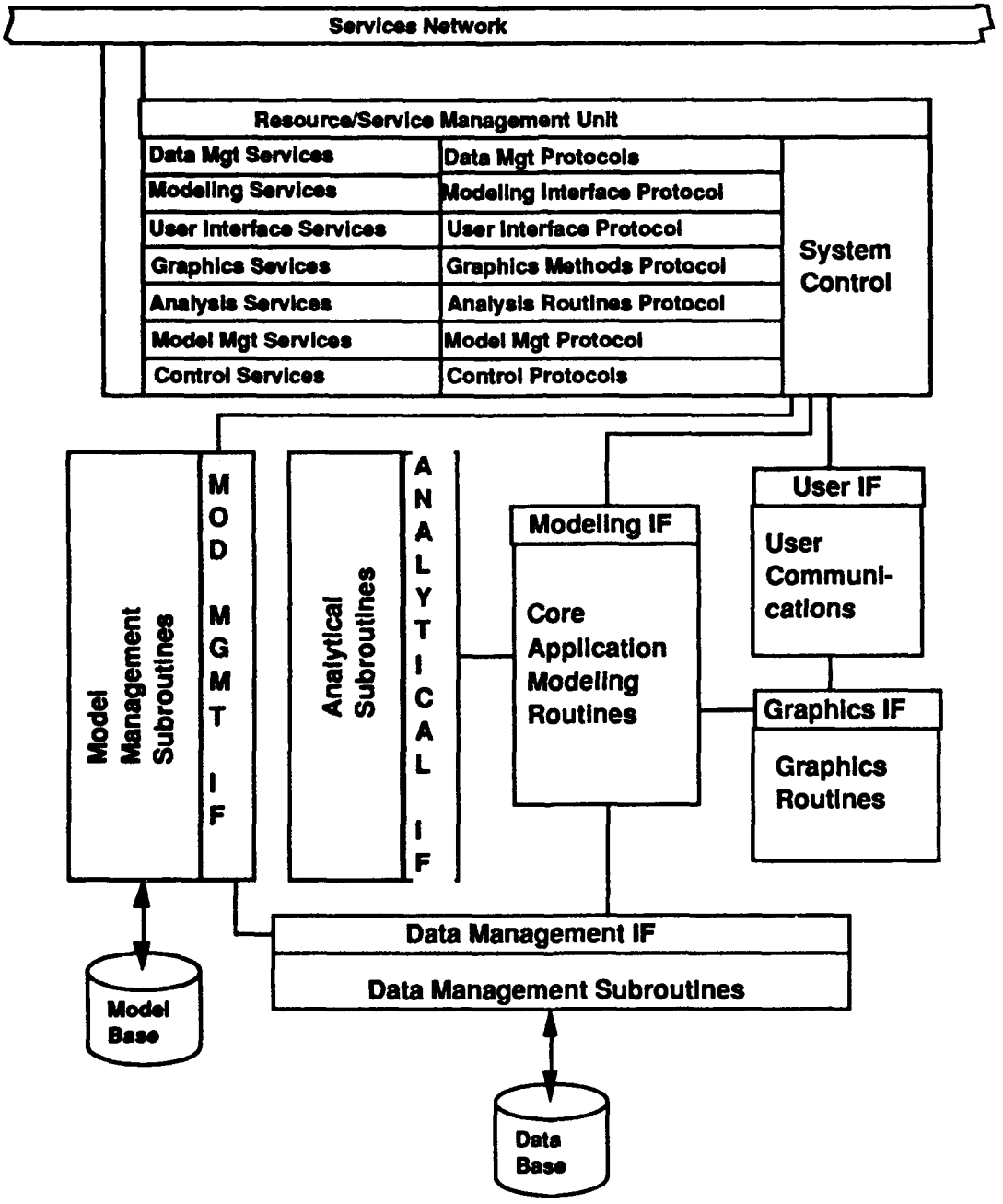



**Figure 33. Installation of Legacy Application into Services Network**

A major task within the next phase of this project will be the further definition and implementation of this services approach to integration. As mentioned previously this approach can build directly off the initiatives and research results of the ISO conceptual schema architecture community. For example, standards such as the evolving PDES and the ISO STEP would serve as powerful mechanisms for specification of both service contracts and protocol ontologies.

## 13. Related Work

Many of the insights and concepts behind the DKMS have resulted from our experience in the development of knowledge-based designer, concurrent engineering, and information integration systems. The following provides a brief overview of several of the salient experiences drawn on in the course of development of the DKMS concept.

### Manufacturing Engineering Experience

AI CAD/CAM KBSI is developing AI CAD/CAM composed of (1) a superquadrics based CAD interface, (2) manufacturing process plan support, (3) design support, and (4) engineering modeling support. AI CAD/CAM is a platform and a collection of integrated tools that provide intelligent support for automated design and manufacturing engineering. The focus of the Intelligent Assistant for CAD/CAM is concurrent engineering of mechanical systems. The goal of AI CAD/CAM is to provide engineering decision support, intelligent CAD interfaces, and manufacturing plan generation support in disciplines ranging from product design to NC code generation and validation. The system is based on a layered architecture supporting geometric reasoning, design object management and control, object-oriented database integration support, and applications development tools. AI CAD/CAM provides the means to integrate engineering support systems from the design to the manufacture of a product. The current focus of AI CAD/CAM centers on manufacturing engineering, but other key areas of development include a high productivity CAD interface as well as Design and Engineering Modeling Support.

Die Process Design and Cost Estimation (Proteus). This prototype system, developed for Chrysler Manufacturing, addresses the problem of automated reasoning about part features directly from the geometric representation. The system is based on superquadrics based geometry models of part and die geometry. The superquadrics based models have the

advantage that they are mathematically complete and provide a symbolic (rather than point, line, curve sets) representation of part features.

The designer interface with Proteus allows the molding of a superquadric primitive into a model of the part which the designer wishes to construct. The CAD interface will allow the designer to "fit" a superquadric to an existing part geometry representation from an existing CAD system. Once the designer completes the representation of the geometry of the desired part, the knowledge based component applies rules for costing and process planning based on the features of that part. The cost estimation facility provides manufacturing cost advice to the designer for sheet metal parts. This cost advice includes consideration for the reuse of existing dies (with modifications) as well as the cost of the required new dies. The process planning advice identifies what dies are required to fabricate the part as well as what types of press equipment can be used to apply those dies.

### Integration Platforms--Space Station Software Support Environment (NASA)

Knowledge Based Systems, Inc. is involved in an integration platform project with NASA Johnson Space Center. The Advanced Software Development Workstation (ASDW) Project is conducting research into development of advanced technologies for Computer Aided Software Engineering (CASE). The ASDW has as a primary goal the provision of concepts and technology for future NASA systems including the Space Station Software Support Environment (SSE), Space Shuttle support systems, and other NASA initiatives. This goal provides an avenue where future requirements for software support environments can be defined and demonstrated. Under this project, we are creating an integrated environment for the development of computer software systems. Like the DKMS, this environment will provide complete control over the development and use of the software systems life cycle data artifacts.

Control of the software development process is an important aspect of any software development system. For a development environment to provide intelligent coordination and control throughout the software development process, the environment must have some means of understanding the intended development process and knowledge of past experiences with similar processes. The Ada APSE concepts, as reflected in the current Space Station Software Support Environment, attempts to accomplish this by defining a "project instance" as part of the setup of an SSE environment. However, this "project instance" falls short of describing the methodology

used in the development process and provides no means for the accumulation of experience within the organization.

KBSI's system will make use of a Framework programmable platform. Within this environment the actual development process can be modeled by the project manager to include the unique characteristics of each project within the context of the development experience at a site. This Framework (organization experience base and project specific characteristics) will then be installed within the ASDW at that site to define (1) the method, tool, and computer resources, (2) the personnel roles within the project, (3) the tasks that must be performed to complete the project, and (4) the configuration control method to be applied to the products of those tasks. Once the instantiation is performed, the environment will have complete control over the development process and the life cycle artifacts produced by that process. We intend to demonstrate that such an environment can support extremely sophisticated Ada development tools.

The ASDW component proposed, (referred to as the Framework Programmable Software Development Platform in this report), will provide a programmable platform of CASE tools for the development of complex software systems, with an emphasis on Ada software development. Through the use of an IDEF3 description (the proposed form of the framework definition) of the process that development should follow, the platform will be configured to make use of the tools and resources necessary for that project. This system development process definition is referred to as a framework. It includes not only the life cycle phases, tasks, milestones, and documentation artifacts, but also:

a) Descriptions of the procedures for analysis, decision making, and configuration control,

b) Callouts for the application of specific methods (methodologies),

c) Definition of common information/data across the different methods,

d) Descriptions of how the results of using the methods will be applied,

e) Role definitions and personnel assignments to roles.

It is envisioned that each site will have its own Framework definition, though software job shops may have several since a framework definition is sensitive to the system software type involved. This framework concept will allow the flexibility required to properly define the development process for a specific project instance, but also provide carry over of the experience base from project to project within an organization. The installed framework will provide the necessary control over the project system development to ensure development success as well as consistency between projects required to allow multiple project coordination, management consistency, and personnel portability.

In addition to the programmable framework, a major element of this effort will be a suite of integrated CASE tools to provide the necessary capabilities for developing Ada based software systems. Particular emphasis will be given to Ada specific (upper)CASE tools that provide Ada code generation from IDEF requirements and object based system design models.

## Engineer's Modelling Assistant Experience

One of the primary components in any design support or automated "designer" system is the engineering analytic model (s). One of the goals of a generalized architecture for building expert systems to support the engineering life cycle would be to also support the development / refinement of the models of the physical systems or processes used in that design process. Knowledge Based Systems Engineer's Modelling Assistant was developed to aid in the development of analytical models for expert systems. Based on both bond graph modeling and constraint management methodologies for engineering performance modelling, it represents one of the most sophisticated support environment for such modelling. Its features include graphic, on screen model generation, a relationship interface which ensures consistency and completeness, and direct access to a programming language. The tool was used to generate a simulation of the cool-down of a car's interior as well as the phenomenological models of all of the components in the associated heating and air conditioning systems for Chrysler passenger vehicles.

## Knowledge Based Design Automation Experience

Our recognition of the opportunity to develop a generalized architecture for design knowledge management and concurrent engineering support is based primarily on our experience building "knowledge based systems for mechanical design" which are in use at Chrysler Motors for the design of

engine box cooling systems, structural fasteners, air conditioning systems and cost estimation systems.

Cooling System Design Assistant. This system, developed in cooperation with Cooling Systems Engineering at Chrysler, generates design specifications for engine box cooling systems. The system models the design process reasoning of an expert cooling systems designer as he generating initial design specifications. The inputs to the system comprise a description of the vehicle from the perspective of the cooling system, i. e. the vehicle subsystem parameters that constrain the performance requirements on the cooling system, a specification of test conditions (e.g. car speed, ambient temperature), and certain technical or administratively directed constraints on the system. In the process of producing acceptable design specifications, the system iteratively proposes specifications, tests these using an analytical performance prediction program, evaluates the test outputs, and revises the design specifications. The analytical program models thermal and mass airflow characteristics of the cooling system in the context of related vehicle subsystems (e.g. transmission, air conditioning system, engine) and has been in use in the cooling systems laboratory for some time. The Cooling System Design Assistant does not search exhaustively, but uses heuristics gleaned from the expert designer in making the redesign step. Concepts derived from this system that we expect to be incorporated in the DKMS Designer Shell include:

1) The use of existing engineering analysis models as evaluators on a design. An important issue in the use of such models involves how the using system will understand the functions, and particularly the limitations and underlying assumptions, of the analysis program.

2) The use of curve based reasoning as part of the reasoning model. Engineers frequently think in terms of curves (or sometimes surfaces) when relating design parameters to performance expectations.

3) The use of truth maintenance techniques. Mechanical design engineers typically think in terms of alternative design proposals, that is, they will freely make and retract assumptions as the design process proceeds. Truth maintenance techniques are useful in managing the complexity implied in this situation.

4) Highly flexible user presentation. Designers do not take one prescribed path to a solution, but may change viewpoints readily. To support this process, an automated design support system must

allow the user to choose to look at various aspects of the problem almost at will, and in varying levels of detail.

Fastener Selection. This system is being developed in cooperation with the Fastener group in Chrysler Engineering. The purpose of the system is to assist structures and body engineers in the selection / design of an appropriate joining approach (as well as selection of fasteners to be used in a joining approach) given the geometry, materials, and function of the mated parts. The fastener design system also takes into consideration product life cycle considerations including:

1) Manufacturing considerations.

2) Assembly considerations.

3) Maintenance considerations.

4) Corrosion, vibration, tolerance, and safety considerations.

The users of this design system have the option of describing their joint by application area / function, by mechanical properties (tensile joint), by creation of the geometry of the joint using a solids modeling system, or by a combination of the above. If the designer can include the application area characterization, then the fastener advisor can propagate many of the product life cycle considerations from its knowledge base. Otherwise, these considerations will be prompted from the user. The initial prototype of the fastener advisor was constructed on a Symbolics Lisp machine. It is currently being revised and expanded to operate on a Compaq 386 class microcomputer. This enhanced version will include the capability to interface through IGES files with traditional CAD environments including 3-d CSG and form feature modelers.

Climate Control Design Assistant. The Climate Control Design Assistant is being developed as an aid to generating design specifications for automobile interior air conditioning systems. The system can be used in one of several modes:

1) To generate a design for a new vehicle.

2) To evaluate design changes proposed by a vendor.

3) To evaluate effects of design changes to other subsystems (e.g. an engine change) on the air conditioning system.

The system incorporates two engineering analysis programs one which models the performance of the air conditioning system components and one which models the interior comfort conditions of the vehicle over time under a test sequence involving varying speeds and initial heat conditions. Performance goals, in contrast to the cooling system case, are frequently specified in relative terms, e.g. "an average interior temperature over a 60-minute test cycle that is 6 degrees less than the 1986 K body car."

Concepts derived from this system that we expect to be incorporated in the DKMS Designer Shell include those listed in the Cooling System section plus the following:

1) "Special study" design, analysis, and presentation. The fact that a design satisfies requirements and constraints may not be enough to gain acceptance for the design in an organization; rather the design engineer will be expected to demonstrate the design rationale, show why the design is better than other satisfying designs, and answer "why not" questions about alternatives. This implies the ability to maintain the design rationale, to do comparative analysis on test results, to do sensitivity analysis on designs, and to produce a suitable presentation of the results.

2) Use at varying levels of detail of engineering models used for performance analysis. A design support system should accommodate system specifications at varying levels of detail, e.g. a component may in one case be modeled as simply an output, in another as a set of parameters manipulated by a routine simulating performance.

Cooperating Knowledge Based Systems. The goals of this project with Chrysler Engineering are twofold:

1) To integrate design systems in various laboratories where there are dependencies between the subsystems being designed, and

2) To develop utilities for constructing future design systems in Engineering where common elements of such design systems can be identified.

The Cooling System and Climate Control System designers serve as the focus of this development. The two systems form a particularly useful testbed because the engine box cooling system and the air conditioning

system in a vehicle have tightly coupled dependencies and because the two design problems present considerable similarities in terms of design method, use of engineering analysis programs, and design presentation as numeric or symbolic parameters (neither employs a geometric model).

Concepts derived from this work that will apply to the DKMS concept include the following:

1) User interface / composite part representation construction utilities.

2) Knowledge base structures for supporting reasoning based on the identification of situation types in the design process.

3) Utilities for defining control structures to manage the interplay of numeric components, heuristic components, display and communication components.

## 14. Summary and Conclusions

The goal of Phase I of this project was the definition of the structure and key algorithms for a design knowledge management system (DKMS) that will support:

1) High productivity CAD system that will reduce the time for concept entry, particularly the entry of form feature geometry data and the generation of design tradeoff model input data derived from that product geometry data. The principal elements of this system include:

   a) Form Feature based user interface,

   b) Geometry concepting support based upon a multi-geometric model constructive solid geometry engine (GCSG).

2) Shape and Feature based concurrent engineering knowledge representation and associative retrieval mechanisms. This facility will enable delivery of qualitative assessments of design options by providing the ability to associatively index historical knowledge structured around form feature interpretations of the evolving product geometry.

3) Container object knowledge representation mechanism for the representation/encapsulation of design knowledge for automated application as well as reuse.

4) Multi-schema DKMS platform (framework) architecture to support:

   a) Information integration of "design decision support tools" with traditional engineering, manufacturing, and field support databases,

   b) Management and retrieval support for concurrent engineering knowledge stored in heterogeneous representation forms,

   c) Configuration, versioning, and perspective definition and control.

5) Complete development shell for "designer" systems which automatically makes the desired design tradeoff decisions.

Phase I resulted in the definition of a new model of the design process based around both life-cycle activities and cognitive primitives. Phase I also produced the base requirements for, and a conceptualization of, a services-based integration architecture for the DKMS platform, the container object system, the form / feature CAD interface, the GCSG, the knowledge-based designer development shell, and the shape/form geometry reasoning system (SGRS). Geometric modeling algorithms were developed for the GCSG and the SGRS. Finally a Phase II plan and proposal was developed.

In Phase II of this project, KBSI will:

1) Complete the detailed design for the DKMS and all of its subsystems.

2) Build and unit test the components of the DKMS.

3) Establish at least three beta test sites where each subsystem will be implemented and used.

4) Integrate the subsystems into a complete DKMS environment.

5) Develop a marketing plan and sponsor group for the Phase III commercialization of DKMS

The key technology issues to be addressed in Phase II include:

1) Efficient implementation algorithms for the GCSG and SGRS.

2) Implementation languages, integration service processors and version management elements of the DKMS platform for the multi-representation knowledge base integration.

Potential pitfalls and barriers that must be managed during the Phase II development include:

1) Maintaining compatibility with the evolving standards.

2) Maintaining in depth knowledge of government and private initiatives in integration schemes and mechanisms.

The proposed DKMS would be usable as a base concurrent engineering platform for almost any engineering automation effort in the Government, Defense Contractor, and Commercial industrial sectors. It would provide a quantum improvement over any available design automation concept available today. It could also serve as a means to promote better university/industry/government ties in that it would provide a direct vehicle for moving design, manufacturing, and field experience into the academic classroom. It is anticipated that initial installations would be in (1) very aggressive small business communities which must leverage scarce resources, (2) DoD installations where management of knowledge bases over long life-cycle weapon systems is a critical issue, (3) NASA Space Station and deep space missions where knowledge bases will span multi-careers, and (4) commercial industries where reduction in product development time is critical to the maintenance of a competitive position. The results of this project will be focused primarily on mechanical device design support. By having industrial, university, and government beta test sites as a part of Phase II and because of the modular nature of the approach we have taken, there is an opportunity for commercialization of the DKMS a piece at a time. This modularity, combined with the feedback from the beta test sites, also will allow KBSI to be able to fund selected pieces of this commercialization itself.

The technology resulting from this project will provide support for design engineers to better integrate the trade-off of various design attributes such as performance, cost, schedule manufacturability, and supportability. It will also result in significant reduction in the engineering cycle time and

manpower requirements for both initial product design and sustaining engineering of a product. The resulting system will truly provide a framework for concurrent engineering initiatives beyond those to be demonstrated in this Phase II effort. By supporting the knowledge based delivery of manufacturing and maintenance experience to the initial product designers, an order of magnitude reduction in redesign and engineering change requests will be realized. Finally, because of the integration support and high productivity CAD support offered by the DKMS, it will find use in many areas other than just product design. For example, in manufacturing planning and production planning, access to the design rationale and design knowledge bases will allow automation of a number of these "manufacturing engineering" activities as well as significantly reducing quality and reliability problems in the final product.

## 15. Bibliography

[Allen 1983] Allen, J.A., Maintaining Knowledge about Temporal Intervals, Communications of the ACM, November 1983, Volume 26, No 11, pp. 832 - 843.

[Allen 1987] Allen, R.H., Boarnet, M.G., Culbert, C.J., and Savely, R.T., "Using hybrid expert system approaches for engineering applications," Computers 2 (1987) pp. 95 - 110.

[Archer 1984] Archer, L.B., "Systematic method for designers," in: N. Cross (Ed.) *Developments in Design Methodology*, (John Wiley & Sons, Ltd., 1984) pp. 57-82.

[Armstrong 1982] Armstrong, G. T., 1982. A study of automatic generation of nonevasive NC machine paths from geometric models. Ph.D Dissertation, University of Leeds.

[Armstrong 1984] Armstrong, G. T., Carey, G. C. and de Pennington, A., 1984. "Numerical code generation from a geometric modeling system." In Pickett, M. S. and Boyse, J.W. (Eds), *Solid Modeling by Computers*. Plenum Press, New York.

[Barr 1981] Barr, Alan H., "Superquadrics and Angle-Preserving Transformations", *IEEE Computer Graphics and Applications*, 1(January 1981) , pp.11-23.

[Barr 1984] Barr, Alan H., "Global and Local Deformations of Solid Primitives", *Computer Graphics*, Volume 18, No. 3, July 1984.

[Bsharah 1986] Bsharah, "AS-IS Engineering Support IDEF0 Model (ESO)", NA-86-1604-L, North American Aircraft Operations, Los Angeles, CA, September 1986.

[De Floriani 1989] De Floriani, L., "Feature Extraction From Boundary Models of Three Dimensional Objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 8, August 1989.

[DeKleer 1983] DeKleer, J., Brown, J. S., "The Origin, Form, and Logic of Qualitative Physical Laws," In: Proceedings of the Eight International Joint Conference on Artificial Intelligence, William Kaufman, Los Altos, CA, 1983.

[Descotte 1985] Descotte, Y. and Latombe, J. C., 1985. "Making compromises among antagonist constraints in a planner," *Artificial Intelligence*, Volume 27, No. 2, pp. 183-217.

[Dixon 1986] Dixon, J.R., "Artificial Intelligence and Design: A Mechanical Engineering View," Proceedings AAAI-86, 2(1986) pp. 872-877.

[Dyer 1983] Dyer, M.G., *In-Depth Understanding*. MIT Press, Cambridge, MA, 1983.

[EIS 1989] "Specifications for Engineering Information Systems Volume I: Organization and Concepts," F33615-87-C-1401, United States Air Force, Air Force Systems Command, Wright Research & Development Center, Wright-Patterson Air Force Base, OH 45433; prepared by Honeywell Systems & Research Center, Minneapolis, MN, Xerox Advanced Information Technology, Cambridge MA, TRW, Redondo Beach, CA, October 1989.

[Faux 1979] Faux, I. D. and Pratt, M. J., *Computational Geometry for Design and Manufacture*, Wiley, New York, 1979, p.230.

[Forbus 1984] Forbus, K.D., "Qualitative Process Theory," AI-TR-789, MIT., July 1984.

[Franklin 1981] Franklin, W. R., and Barr, A. H., "Faster Calculation of Superquadric Shapes," *IEEE Computer Graphics and Applications*, Volume 1, No. 3, 1981.

[Friedell 1984] Friedell, M., "Automatic Synthesis of Graphical Object Descriptions," *Computer Graphics*, Volume 18, No. 3, July 1984. pp. 53-62.

[Friedman 1969a] Friedman, G.J., and Leondes, C.T., "Constraint Theory, Part I: Fundamentals", *IEEE Transactions on System Sciences and Cybernetics*, Vol. ssc-5, No.2, Apr., pp. 132-140.

[Friedman 1969b] Friedman, G.J., and Leondes, C.T., 1969, "Constraint Theory, Part II: Models Graphs and Regular Relations", *IEEE Transactions on System Sciences and Cybernetics*, Vol. ssc-5, no.1, Jan., pg. 48-56.

[Friel 1988] Friel, P. Griffith, "Modeling Design Reasoning in Automotive Engineering," PhD Dissertation, 1988, Texas A&M University.

[Goel 1989] Goel, V., Pirolli, P., "Motivating the Notion of Generic Design with Information Processing Theory: The Design Problem Space," AI Magazine, Vol 10, No. 1, Spring 1989.

[Grayer 1977] Grayer, A. R., 1977. "The automatic production of machined components starting from a stored geometric description," In, McPherson, D. (Ed), *Advances in Computer-Aided Manufacture.* North-Holland, Amsterdam.

[Gunning 1989] Gunning, D., "A general Framework for Describing Human-Computer Information Systems" Technical Report, AFHRL Wright Patterson Air Force Base, OH 45433, 1989.

[Harmon 1986] Harmon, G., Change in View. The MIT Press, Cambridge, MA., 1986.

[Henderson 1984] Henderson, M., R., "Extraction of Feature Information from Three Dimensional CAD Data," Ph.D dissertation, Purdue University, 1984.

[Henderson 1985] Henderson, M. R., 1985. Extraction and organization of form features, In Proceedings of Prolamat 1985, pp.131 141.

[Hobbs 1986] Hobbs, J., "On the Coherence and Structure of Discourse," in: The Structure of Discourse. L.Polanyi, (Ed), Ablex Publishing Corporation, Norwood, NJ, 1986.

[IISyCL 1989] IISyCL Constraint Language, submitted to AFWAL/LDL , WPAFB, for approval, December 1989.

[Jared 1985] Jared, G.E.M., Shape Features in Geometric Modeling, in Pickett, M. S. and Boyse, J. W., (Eds), Solid Modeling by Computers, Plenum Press, New York, 1985.

[KBSI 1989] Lockledge, J.C., Engineering Modelling Assistant, Final Report, KBSI Internal Technical Report, KBSI-89-023, 1989.

[Keen 1989] Keen, A.A., Mayer, R.J., private communication, 1989.

[Kim 1989] Kim, W., Lochorsky, F.H., eds., *Object-Oriented Concepts, Databases, and Applications.* ACM Press, New York, 1989.

[Kuipers 1984] Kuipers, B., "Common Sense Reasoning about Causality: Deriving Behavior from Structure," Artificial Intelligence Vol. 24, 1984.

[Krause 1989] Krause, F.L., Beinert, M., Fortmann, K., et al., "System Architecture for CAD/CAM Integration," Final Report, Engineering Office of Scientific Affairs, Chrysler Motors Corporation, Highland Park, MI, 1989.

[Kypriano 1980] Kypriano, L. K., Shape Classification in Computer Aided Design, Ph.D. Dissertation, University of Cambridge, United Kingdom, 1980.

[Laughton 1985] Laughton, S., "Explanation of Mechanical Systems Through Qualitative Simulation," AITR85-19, AI Laboratory, University of Texas at Austin TX, December, 1985.

[Mantyla 1988] Mantyla, Martti, An Introduction to Solid Modeling. Computer Science Press, Rockville, MD, 1988.

[Mayer 1987] Mayer R. and Underbrink A., "Autonomous Underwater Vehicle Knowledge Based Control System Design," AUV Phase I Final Technical Report, SEA Grant Project #R/C, Grant Number NZ85AA-D-SG128, July 1987.

[Pentland 1986a] Pentland, Alex, "Recognition by parts", Technical Note No. 406, SRI International (December 16, 1986).

[Pentland 1986b] Pentland, Alex, "Perceptual Organization and the Representation of Nature Form," *Artificial Intelligence* 28, 1986, pp. 293-331.

[Ramey 1983] Ramey, T. L., "Guidebook to Systems Development," Internal Research Report, Hughes AirCraft Co., El Segundo, CA 1983.

[Requicha 1988] Requicha, A. A. G., and Vandenbrande, J., "Automated systems for process planning and part programming," *Artificial Intelligence Implications for CIM*, Ed. Andrew Kusiak, IFS/Springer-Verlag, 1988, p. 299-326.

[Serrano 1988] Serrano, D., 1988. Constraint management in conceptual design. Ph.D. Dissertation, MIT.

[Simon 1967] Simon and Shuster, "The Way Things Work: An Illustrated Encyclopedia of Technology" New York, 1967.

[Su 1989] Su, Chuan-Jun, 1989, An Enclosing Object Based Automatic NC Code Generation System. Ph.D Dissertation, Texas A&M University.

[Tilove 1980] Tilove, R. B., "Set Membership Classification : A Unified Approach to Geometric Intersection Problems", *IEEE Trans. Computers*, Volume C-29, No. 10, Oct. 1980, pp. 874-883.

[Ting-jung 1989] Ting-Jung Fan, Gerard Medioni, and Ramakant Nevatia "Recognizing 3-D Objects Using Surface Descriptions," IEEE Transactions on Pattern Analysis and Machine Intelligence, November 1989, Vol. 11, No. 11.

[Voelcker 1978] Voelcker, H., et al., "The PADL-1.0/2 System for Defining and Displaying Solid Objects", *Computer Graphics*, Volume 12, No. 3, August 1978, pp. 257-263.

[Webster 1987] Webster, D. E., "Mapping the Design Representation Terrain: A Survey," MCC Technical Report Number STP-093-87, MCC Austin TX, July 1987.

[Wilensky 1983] Wilensky, R., Planning and Understanding: A Computational Approach to Human Reading. Addison-Wesley Publishing Company, Redding, MA, 1983.

[Wilson 1987] Wilson, M.L., Information Automat: Concept Definition Facility. IA Systems, Inc., San Jose, CA, 1987.

[Woo 1982] Woo, T., C., "Feature Extraction by Volume Decomposition," in Proc. Conf. CAD/CAM Technol. Mechanical Eng., M.I.T., 1982.