

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

EXPERT SYSTEMS IN ENGINEERING DESIGN:
A PROTOTYPE APPLICATION FOR INJECTION MOLDING OF PLASTIC PARTS

by
Sally Jean Steadman

A dissertation submitted to the
Department of Mechanical Engineering and
The Graduate School of The University of Wyoming
in Partial Fulfillment of Requirements for the Degree of

DOCTOR OF PHILOSOPHY
in
MECHANICAL ENGINEERING

Laramie, Wyoming
December, 1994

UMI Number: 9524549

UMI Microform 9524549

Copyright 1995, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized
copying under Title 17, United States Code.

UMI

300 North Zeeb Road
Ann Arbor, MI 48103

Steadman, Sally J., Expert Systems in Engineering Design: A Prototype Application for Injection Molding of Plastic Parts, Ph.D., Department of Mechanical Engineering, December, 1994.


Recent developments in expert system shells have the potential to markedly impact the use of knowledge-based expert systems for complex tasks like engineering design. The knowledge required in an engineering design application is categorized and representations are formulated for each of the knowledge types. A prototype expert system implements each of the knowledge representations, integrating external knowledge sources -- a solid modeler and a materials database -- with a hybrid expert system shell.

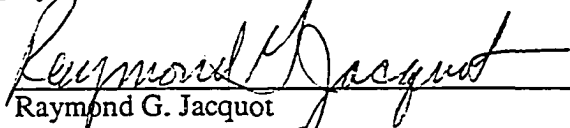
The context chosen for the prototype expert system is the design of an injection molded plastic part; a subproblem, the design of a cantilever snap joint to join two components, is representative of engineering design problems. A designer, using a solid modeling system, develops a conceptual design and then invokes the expert system to determine geometric parameters for the design. The object-oriented, rule-based expert system integrates various knowledge sources for injection molding: heuristic rules, design specifications, geometric configurations and constraints, analysis software, and a material properties database. The expert system, using these knowledge sources interactively with the designer, determines the feasibility of the conceptual design, and modifies the design, iteratively, until an acceptable design is formulated.

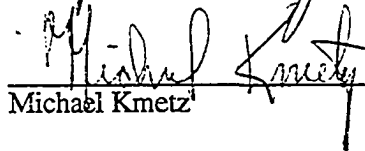
The prototype system has illustrated the utility of expert system shells for engineering design problems. Expert system shells offer rich development environments with interfaces to programming languages (and hence to a multitude of existing computer-aided engineering software systems), access to databases, and graphical capabilities to assist in developing user interfaces. Expert system shells deal effectively with the complexity of engineering design, and they provide a design engineer, familiar with the heuristics of the problem, with an easy-to-use tool for rapid development of a design aid.

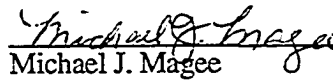
TO THE GRADUATE SCHOOL:

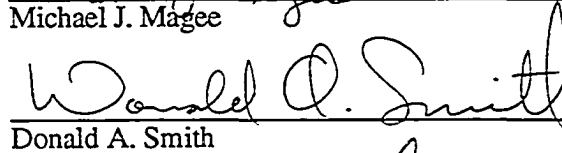
The members of the Committee approve the dissertation of Sally Jean Steadman presented on October 11, 1994.


Kynric M. Pell, Chairman


Raymond G. Jacquot


Michael Kmetz


Michael J. Magee


Donald A. Smith


David E. Walrath

APPROVED:


Kynric M. Pell, Head, Department of Mechanical Engineering


Thomas G. Dunn, Dean of Graduate School

ACKNOWLEDGMENTS

I would like to acknowledge the assistance provided by Bill Palsulich, Mold Engineering Manager for Cobe Gambro Hospal Medical Inc., in sharing his injection molding expertise and contributing to the knowledge base for this research. Further, I would like to recognize the foundation for this research, provided by Dr. Michael Kmetz, Integrated Design Engineering Systems. I would also like to thank Dr. Kynric Pell for his valuable support and encouragement throughout this project.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION.....	1
Engineering Design	3
Status of Expert Systems in Design.....	3
Research Objectives.....	4
Research Focus.....	5
Prototype Development.....	7
2. KNOWLEDGE -BASED EXPERT SYSTEMS: AN OVERVIEW	9
Architecture.....	12
Tools for Building Expert Systems	16
A History of Expert System Applications.....	18
Summary.....	20
3. BUILDING A KNOWLEDGE-BASED EXPERT SYSTEM	21
Characteristics of Expert System Problems.....	21
Tasks	22
Knowledge Acquisition	23
Choosing a Tool.....	25
Developing a Prototype	27
Validating the System.....	28
Summary.....	28
4. EXPERT SYSTEMS IN ENGINEERING	29
Problem Solving	29
Design Methodology	30
Expert Systems in the Design Domain	33
Expert System Implementations: Design Applications.....	35
Research Areas.....	43
Summary.....	44
5. PROTOTYPE DEVELOPMENT.....	45
Choosing an Expert System Building Tool	46
Kappa PC Developers Environment.....	48
Application: Cantilever Snap Joints.....	49
Knowledge Representation	52
User Interface	55
Inference Strategies.....	59
Prototype Evaluation	61

6. INTEGRATION OF EXTERNAL KNOWLEDGE SOURCES	63
External Interface Capabilities.....	65
Solid Modeling Software	66
Database Software	67
Hardware / Software Environment	71
Summary.....	72
7. RESULTS AND CONCLUSIONS	73
Results	74
Conclusions	76
Future Research.....	78
APPENDIX A: Listing of Classes (including Methods), Instances, Rules, Goals, and Functions.....	81
APPENDIX B: Rule Trace Examples	105
APPENDIX C: IDEAS Snap Feature	137
APPENDIX D: Edit Program	139
APPENDIX E: User Evaluation Form.....	141
REFERENCES	142

LIST OF TABLES

	PAGE
Table 2.1 Characteristics of Conventional Programs vs. Expert Systems	10
Table 2.2 Historical Expert System Development.....	18
Table 5.1 Materials.....	53
Table 6.1 PROSPECTOR Data Sheet	69,70

LIST OF FIGURES

	PAGE
Figure 2.1 Architecture of an Expert System	12
Figure 3.1 Programming environments	25
Figure 3.2 Expert System Shells.....	26
Figure 4.1 Design Process: An Iterative Model.....	30
Figure 5.1 Graphical Presentation Tools	48
Figure 5.2 Representative Snap Joints.....	50
Figure 5.3 Cantilever Snap Joint Geometry.....	50
Figure 5.4 Object Hierarchy	52
Figure 5.5 Feature Selection.....	55
Figure 5.6 Design Interface	56
Figure 5.7 Initializing Cantilever Configuration.....	57
Figure 5.8 Entering Geometry Data.....	57
Figure 5.9 Design Results.....	58
Figure 6.1 System Approach	63
Figure 6.2 Cantilever Snap Joint Object.....	66

CHAPTER 1

INTRODUCTION

The computer has become an essential tool for the engineer. Computer-aided drafting (CAD), finite element modeling (FEM), and solid modeling, along with special purpose analysis programs are known as computer-aided engineering (CAE) tools. These tools are used in every facet of the engineering process -- design, analysis, simulation, and manufacturing, and are implemented at two distinct levels depending on the capabilities of the hardware / software.

The lower level implementation of the CAE tools exists on DOS based, PC compatible personal computer systems, and generally includes software with restricted capabilities for the engineer: CAD, 3-D wireframe and/or surface modeler, and FEM. These systems are primarily used by smaller engineering firms. Larger firms, on the other hand, utilize more sophisticated implementations on UNIX based workstations, which are generally faster than personal computers and offer a broader range of features and capabilities. Software restricted to mainframe computers in the past now runs on workstations. Typically, CAE software in the workstation environment is an integrated system based on a three dimensional solid modeler. A mechanical designer, using an integrated tool such as Structural Dynamic Research Corporation (SDRC) I-DEAS™ or Dassault Systemes CATIA™, creates a three dimensional model. This model is then used as input data for the engineering analysis incorporated in the integrated design software. For example, the model can be analyzed using FEM or dynamic simulation techniques.

Regardless of the level of the CAE implementation, the key to productivity for both the designer and for the entire product development team is the degree of integration of the CAE tools. The computer-based design representation needs to be integrated with the analysis and manufacturing tools or exported directly to these tools, with minimal user intervention. Manufacturing firms are currently using solid models to automatically

generate machining code for milling machines, turning centers, and other computer controlled manufacturing equipment.

Another, emerging, key to increasing productivity is concurrent engineering (also known as simultaneous engineering). Concurrent engineering decreases the time required to develop a product by considering the manufacturing process early in the design of the product, or *concurrently* with the product design. In fact, this approach takes into account not just the functionality of the product, but its quality, manufacturability, testability, and maintainability.

Recent developments in Artificial Intelligence (AI), and more specifically in knowledge-based expert systems, promise to significantly extend the use of CAE tools in the interface between design and manufacturing. Traditional programming concepts and algorithmic procedures do not lend themselves to this interface; the field of AI is attempting to produce new technology to address these new concerns. Not only are expert systems a part of AI, but AI also includes natural language processing, image processing, robotics, and neural networks. However, the research presented here is limited to knowledge-based expert systems.

A knowledge-based expert system (KBES) differs from conventional software in several important ways. One definition widely used for expert systems is:

. . . interactive computer programs incorporating judgment, experience, rules of thumb, intuition, and other expertise to provide knowledgeable advice about a variety of topics (Gaschnig, Reboh, and Reiter 1981, 7)

Expert systems are symbolic processors, in which the knowledge base, or expert information, is separate from the methods for manipulating the knowledge base.

Generally, programming languages or tools incorporate the methods used to manipulate the knowledge, so the developer concentrates on constructing the knowledge base, and not on the procedures for processing the knowledge. An expert system uses the knowledge base to *reason* about a problem in a manner similar to the process used by an *expert* in solving the problem.

ENGINEERING DESIGN

Engineering design is a creative process, best conducted by a knowledgeable designer with years of experience. Through his experience, the designer has developed a set of design guidelines, or heuristics, that he applies to new design situations in developing a conceptual design. Often, for an experienced designer, this conceptual design is near to the final, optimal design solution. Using analysis tools, the design is evaluated, and if the original design specifications have not been met, the design is modified. An expert designer will again use his judgment and expertise to modify the design. The modified design is evaluated, and this iterative process continues until an acceptable design is accomplished. Sometimes the specifications must be relaxed in order to arrive at an acceptable design; again, the expert uses his knowledge to adjust the specifications.

A difficulty encountered in applying expert systems to a design problem is acquiring the expert knowledge for the system. Often an expert cannot express how, or why, he does something; typically, he has not thought about the processes he uses to solve a problem.

Engineering design is clearly becoming more of a team effort because the amount of data and the scope of considerations involved in a significant project transcend both the breadth and depth of any one individual's experience. The segmentation of a design project, for the numerous designers working on the project, is facilitated by database structures and file management systems incorporated in the integrated software. The team concept is often informal in smaller firms, but can be very formal and highly documented in larger firms.

Problems encountered in mechanical design share some common characteristics. They often involve a choice of manufacturing processes and a wide choice of materials, and the mechanical designs are often fairly complex, three-dimensional artifacts. Mechanical design software attempts to integrate both the material property data and manufacturing process simulations in order to assist an individual designer.

STATUS OF EXPERT SYSTEMS IN DESIGN

Early expert systems have been applied to problems such as an advisor for a finite element program, and monitors or controls for manufacturing and chemical processing. However, engineering design differs from these types of problems in two basic ways: the

diversity of the information (or knowledge) and the complexity of the engineering systems. In engineering design there is no *one*, correct solution, but usually an optimum solution can be identified by applying constraints like economics, physical limitations, and manufacturing considerations.

Few expert systems exist for the engineering problem solving tasks of planning and design, and most of the ones that do exist have been implemented using programming languages or environments. Representative implementations for these tasks are discussed in Chapter 4. Programming languages or environments have not promoted the use of expert systems for design problems since they are relatively difficult to use, and are particularly onerous to the typical engineer with limited programming skills. Tools are available that are appropriate, reasonable to use, and that facilitate rapid development of expert systems for complex tasks such as engineering design. Expert system shells fit these requirements, but have not been traditionally applied to engineering design problems.

The knowledge required in a mechanical engineering design problem is a combination of design rules and guidelines, analysis software incorporating engineering models and governing equations, and database information about material properties and specifications. The knowledge is provided by multiple sources, requiring a variety of specialized knowledge representations, and needs to be integrated for fully functioning systems.

Current expert system implementations make little use of data generated in existing applications. Computer-aided design and solid modeling systems are widely used by engineers, and produce geometric and feature databases. Databases for material selection are also important tools for design, as well as the analysis information generated by software such as finite element modeling. Since all of these tools produce data that can significantly enhance the capabilities of an expert system for design applications, they should be integrated with the expert system.

RESEARCH OBJECTIVES

This research investigates the feasibility of applying knowledge-based expert systems to engineering design problems. A variety of tools currently exist for the expert system developer, ranging from programming languages which require considerable

understanding of the fundamental theory involved in expert system programming, to expert system shells, which can be thought of as high level expert system languages. Expert system shells appear to offer rich development environments with interfaces to programming languages, access to databases, and graphical capabilities to assist in developing user interfaces. Expert system shells are easy-to-use tools for the typical engineer with limited computer skills, and provide a viable tool for developing expert systems.

The goal of this research is to develop a generic approach, or template, for expert system applications, based on expert system shells, that can be used by engineers in day-to-day applications. To accomplish this goal, the steps in developing an expert system application for engineering design problems must be formulated. The following tasks for expert system development are explored, and formalized, in this research:

- investigate the use of expert system shells for design problems
- categorize the knowledge required to solve design problems
- formulate representations for the knowledge
- integrate the expert system with external databases and solid modeling software
- develop interactive capabilities, as well as graphical interfaces.

RESEARCH FOCUS

This research focuses on manufacturing processes, which are integrally involved in product design. Since the specific details of the manufacturing process impact the appearance, strength, and long term stability of a product, the process needs to be considered during the product design. Thus the designer needs to have detailed knowledge of the specific manufacturing process.

Manufacturing processes have recently evolved from processes used since the industrial revolution for the traditional materials of metals, metal alloys, and wood. Plastics have been used since WWII; followed by composite materials in the last two decades. The associated manufacturing methods of injection molding, blow molding, and thermal forming, which did not exist prior to the 1950's, are responsible for a major portion of today's consumer goods. However, the number of designers experienced with these new materials and methods has not kept pace with the penetration of these materials

into the marketplace. Therefore, designers experienced with the traditional materials, as well as novice designers, require assistance employing the newer materials in their designs. The difficulty of using these materials in product design is compounded by the vast number of plastic materials available, which currently exceeds 18,000. The existing materials are being alloyed and blended by suppliers to create new materials at a very rapid rate.

The design area selected for this research is injection molded part design. Previous work done in this area (Kmetz 1986) provides the foundation for this research. Kmetz developed software for adjusting conceptual part designs, using a set of generally accepted rules applied by plastic designers. His work incorporated the rules in algorithmic procedures and did not use an expert system approach. His application was also limited to those rules which can be implemented in algorithmic procedures. A major source of his design information is in the design handbooks which are generally available from individual material suppliers. These handbooks contain the experience of *expert* designers. Another source of information required for his work is the material property information which can be found in suppliers' manuals and databases and in independently published materials.

A successful plastic product begins with a good part design, which is a result of a thorough knowledge of design as well as an understanding of the process and material being used. (Beall 1990) In feature design, a complex part is decomposed into its basic elements: the nominal wall, projections off the nominal wall, and depressions into the nominal wall. The nominal wall can be simplified to a set of flat plates, no matter how complex the shape is. All projections -- reinforcing ribs, pegs, gussets, snap joints -- can be addressed with similar guidelines. Likewise, all depressions are viewed as similar design problems. Other plastic part features are combinations of these three basic elements; therefore, guidelines can be used to design each basic element, and the elements assembled to create complex geometries. Design guidelines often vary depending on the materials chosen for the plastic part, and pertain to the moldability of a part.

The scope of the research presented here is reduced to a manageable level, but demonstrates the viability of expert systems for design applications, by limiting the expert system application to one basic element of a plastic part. The design of a specific

projection, a cantilever snap joint for joining two injection molded plastic parts, involves the knowledge sources found in a generic design problem, and was selected as the focus of this effort.

To design a snap fit, the experienced plastics designer uses a representative set of the knowledge used in plastic part design problems. Heuristics, or the design rules, are the basis for a conceptual design, or initial configuration. The conceptual design depends on the material selected for the part and related material properties, and the functionality of the part, i.e., the specifications for the design. Governing equations assist the designer in determining the appropriate geometric relationships. Representations for the various knowledge sources will be developed, providing a template for design problems in general.

PROTOTYPE DEVELOPMENT

The feasibility of using an expert system shell for engineering design problems can be explored by building a prototype expert system. The first step is to identify a design problem for the prototype implementation. This problem should be representative of typical engineering design problems, to demonstrate the viability of expert systems as CAE tools in the mechanical design area. It should also be confined to a fairly narrow domain, to facilitate the implementation of the expert system. The design should involve each of the knowledge types in an engineering design problem, to develop a template that can be used in other design applications. If each knowledge type is incorporated in the prototype, extensions are easily made to the template for more complex problems. The context selected for this research, injection molded part design, fits these specifications.

Another important task in developing the prototype is selecting an appropriate tool for the expert system implementation. Many expert system shells are available, offering a range of features and capabilities. The prototype should demonstrate the ease with which a typical engineer can develop an expert system for design applications.

A survey was conducted to identify expert system shells which provide the necessary development environment for this research. The criteria used in evaluating various products included ease of use, range of available features, implementation platforms, and cost. The tool selected was Kappa PC™, available from IntelliCorp, an early leader in developing software for expert system applications. Kappa PC runs on IBM

PC[®] compatible hardware. Another product available from IntelliCorp, ProKappa[™], is a workstation version of an expert system shell. Although IntelliCorp offers both products, they were developed independently, and are not completely compatible. Therefore a system developed on a PC cannot be migrated to a workstation without conversion efforts.

An important aspect for software acceptance is the user interface. A graphical interface can make a system easier to learn and easier to use; extensive explanation facilities build a user's confidence in the system, and thus promote its acceptance. The expert system should accommodate use by novices, as a tutor, and by experts, as a design aid. An interactive interface allows the user to participate in the design process, instead of merely observing the results.

The resulting expert system is a computer-aided engineering design aid. The expert system, using the knowledge sources interactively with the designer, assists the design engineer in developing a conceptual design and determining its feasibility. The expert system described here not only incorporates design rules (both heuristics and governing equations), but interfaces to a materials database and to a solid modeling package. The expert system iteratively evaluates and modifies the design, if necessary, until the specifications are sufficiently accommodated. The prototype system does not identify an optimal solution, but this functionality can be easily incorporated in the expert system by including additional rules that address constraints related to conditions for optimal design.

The prototype expert system is implemented in a fairly narrow domain. To be an effective design tool, the research must be extended from designing a basic feature to designing more complex parts and their corresponding mold designs, incorporating sophisticated analysis techniques for flow within a mold and structural properties. Extending the expert system to other manufacturing processes will produce an even more valuable tool. However, the value of this research is in establishing the guidelines, or templates, for developing expert system tools for the design process.¹

¹ This research has been accepted for publication (Steadman and Pell 1994; Steadman 1994).

CHAPTER 2

KNOWLEDGE-BASED EXPERT SYSTEMS: AN OVERVIEW

Artificial Intelligence (AI) is an area of computer science dealing with the emulation of human thought processes. AI is concerned with understanding human problem-solving strategies and incorporating (or simulating) these strategies in computer programs. Knowledge-based expert systems (KBES) are a specific application of AI. Edward Feigenbaum, generally regarded as the father of expert systems, defines an expert system as (1981, 221):

an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution.

Another definition is:

. . . solves real-world, complex problems using a computer model of expert human reasoning, reaching the same conclusions that the human expert would reach if faced with a comparable problem (Weiss and Kulikowski 1984, 1).

However, the most widely accepted definition is given by Gaschnig, et al. (1981):

Expert systems are interactive computer programs incorporating judgment, experience, rules of thumb, intuition, and other expertise to provide knowledgeable advice about a variety of tasks.

These definitions also apply to many existing computer programs, which are not usually thought of as expert systems. Most authors make this distinction by defining an expert system to be a program in which the knowledge base, or expert knowledge, is separated from the methods for applying the knowledge, i. e. the inference mechanism, reasoning mechanism, or rule interpreter. In fact, Feigenbaum, McCorduck, and Nii (1988, 7) state that the power of an expert system depends on the amount and quality of the

knowledge it possesses, not on the particular formalisms and inference schemes it possesses.

Other characteristics of expert systems include (Adeli 1988, 6; Fenves 1986, 3; Maher 1987, 5):

- knowledge-intensive programs
- knowledge usually divided into many separate rules
- highly interactive
- user-friendly, intelligent, user interfaces
- explanation facility for reasoning
- incremental growth capability
- knowledge is readable and understandable

Expert systems can provide advice, answer questions, and justify their conclusions. The differences between conventional programming and expert systems are summarized in Table 2.1 (Maher 1987, 4).

Table 2.1. Characteristics of Conventional Programs vs. Expert Systems

CONVENTIONAL PROGRAMS	EXPERT SYSTEMS
Representation and use of data	Representation and use of knowledge
Knowledge and control integrated	Knowledge and control separated
Algorithmic (repetitive) process	Heuristic (inferential) process
Manipulation of large databases	Manipulation of large knowledge bases
Programmer ensures uniqueness and completeness	Knowledge engineer relaxes uniqueness and completeness constraint
Midrun explanation impossible	Midrun explanation possible
Numerical processing	Symbolic processing

Solving complex problems involves a large knowledge base and extensive searching of that knowledge. A human expert rapidly narrows the search by recognizing patterns and using appropriate heuristics, or rules of thumb. With the technology currently

available, expert systems are limited to narrow, highly specialized, well defined domains (contexts). They are not able to reason broadly over a field of expertise. With future improvements in computer memory size and speed, this limitation will gradually disappear, and expert systems will be applied to wider domains, and more complex problems. Expert systems are currently expensive to implement, requiring significant investments of human and capital resources. These costs will also diminish with technological advances.

Expert systems are employed in many engineering fields for a variety of reasons. They are used to compile and archive knowledge from employees and external experts to develop intellectual capital for a firm. Expert systems can be available any time of the day or night, not just during business hours; access can be distributed to many employees and locations. They provide consistent answers, and can be updated with new expertise as new policies or methods are implemented. They do not bias judgments, or jump to conclusions, but systematically consider all possibilities. They attend to details, and may produce several solutions for a particular set of conditions.

However, expert systems cannot reason from axioms or general theories, or by analogy. They do not learn, and they lack common sense. The performance of an expert system rapidly deteriorates when it is extended beyond the narrow task that it was designed to perform. (Harmon and King 1985, 7)

Companies using expert systems have measured both qualitative gains and quantitative gains. Qualitatively, expert systems have improved not only the quality, but the consistency of designs and their compliance with standards, and have encouraged innovation among the users of the systems. Quantitatively, less time is spent in bookkeeping tasks resulting in more productive time for engineering and designing tasks; design data are available earlier in the product cycle, and can be used in downstream tasks such as detailed documentation preparation, material specifications, and job costing. In measuring productivity, one example is given by the Babcock and Wilcox Power Generation Group in the design of heat transfer components where expert systems have decreased, by two-thirds, the time to model the components; in addition, detail drawings are automatically generated along with other manufacturing documents. (CIME 1989)

ARCHITECTURE

An expert system consists of three main components: a knowledge base, an inference engine (search strategy), and a domain (or context). Additional components may include a user interface, an explanation facility, and a knowledge acquisition facility.

Figure 2.1 illustrates these components.

The knowledge base consists of the facts and the heuristics about the domain. The heuristics include rules of thumb, and the strategies limiting the search for solutions in large problem spaces, which are usually empirical, and are based on experience and intuition, not mathematical or scientific proof. The inference engine controls the reasoning operations, i.e., it is the executive for the expert system. The inference engine *fires* (applies) the rules, and may alter the knowledge base.

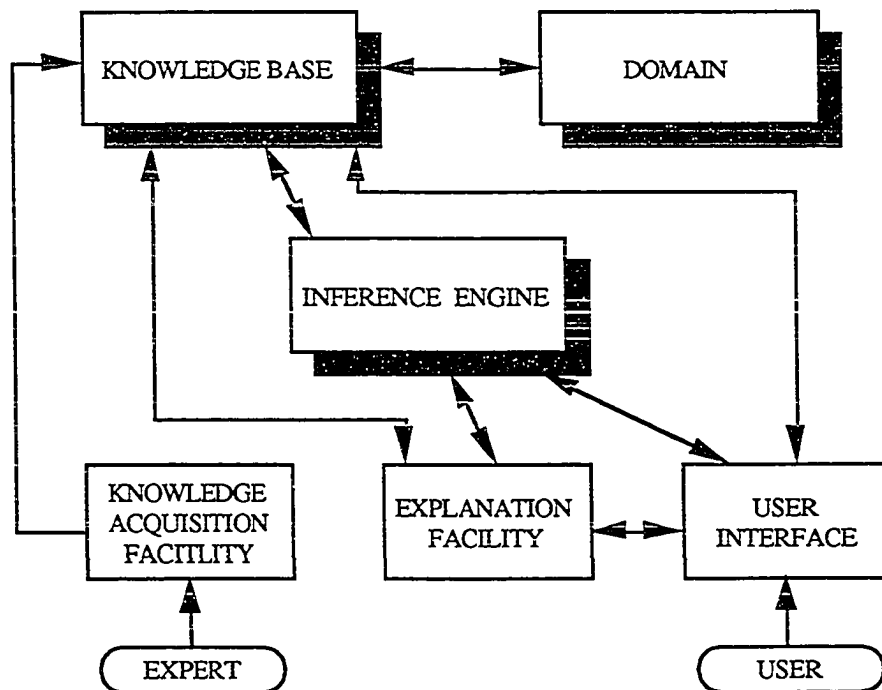


Figure 2.1. Architecture of an Expert System

The user interface should provide for several user-modes: client -- getting answers and explanations to problems, tutor -- improving or increasing the system's knowledge, and pupil -- harvesting the knowledge base for human use. (Michie 1980, 370)

KNOWLEDGE BASE. The knowledge can be represented with various schemes: production rules, predicate logic, semantic networks, frames, and object-oriented frameworks. When several, independent, experts cooperate to solve a problem, a blackboard model is used. A brief discussion of these schemes follows.

Production Rules. Production rules are IF-THEN (or condition-action, or antecedent-consequent) statements, where satisfaction of one or more conditions results in one or more actions. Each rule is an unordered, data sensitive unit, contrasted with the sequenced instructions of procedural languages. The conditions are stored in a database, and the actions modify the contents of the database when they are invoked. (Newell and Simon 1972), (Davis and King 1977)

Predicate Logic. The simplest form of logic is propositional logic. Propositions can be either TRUE or FALSE and can be connected by logical operators (and, or, not, implies, equivalence) to form a propositional calculus of constants, functions, and predicates. Predicates are used to represent relationships, e.g., SUM (A, B, RESULT). Predicate calculus is a structured extension of propositional calculus employing variables and quantifiers (all and some). It introduces specific roles for the elements of the propositional calculus and allows for deductions to be calculated; the characteristics of a particular object can be deduced from more general statements about the attributes of some or all objects in a set to which the object belongs. (Dym and Levitt 1991)

Semantic Networks. A semantic network is composed of a set of nodes, representing objects and their descriptors, and a set of links (semantics) connecting the nodes, representing the relations among the nodes. Commonly used links are is-a and has-a links. Semantic networks have been used primarily in natural language research. (Quillian 1968)

Frames. A frame (or schema) is used to describe an object, and is a special case of a semantic network. It is composed of slots which store information about the object. This information may be default values, pointers to other frames, sets of rules, or procedures.

Frames may be linked in a tree-like structure (network), thus allowing inheritance of slots and their values from one level of nodes to subsequent levels. (Minsky 1975)

Object-Oriented Frameworks. Object-oriented programming involves the use of objects, which are extensions of frames tightly coupled with operations (methods). Each object is described by a number of attributes, which may be integer or real values, strings, or complex data structures. The behavior of an object is defined by methods, or procedures which manipulate the state of an object. Objects interact with each other by sending messages to execute one or more methods. Objects are arranged in a hierarchy of classes and subclasses having similar attributes, with lower classes inheriting methods and attributes from higher classes. Subclasses are specializations of their parent classes. (Stefik and Bobrow 1985)

Blackboard. The blackboard model was developed to provide a reasoning mechanism when multiple knowledge sources exist. The blackboard serves as the location for posting communications (messages) between the various knowledge sources. It also keeps track of the current state of the problem. The blackboard model is generally used for complex problems that must be partitioned into subproblems (knowledge sources). (Nii 1986)

The knowledge representation scheme should be chosen as the first step in implementing an expert system. In order to choose an appropriate scheme, the knowledge engineer must first organize the knowledge, gaining a familiarity with the domain. Some general guidelines in choosing the appropriate representation are:

- simple production and logic systems are good for poorly understood domains, where the knowledge structure cannot be well described
- structured production and frame systems increase run-time efficiency and reduce the effect of the volume of knowledge on run-time, but are more difficult to implement
- logic systems are more difficult to implement for mathematical expressions.

INFERENCE ENGINE. The inference engine selects which rules to examine (in either a forward or backward direction), evaluates the rules, generates new facts or retrieves facts needed by rules, to generate solutions for a set of conditions. When more than one rule is eligible for firing, several options are commonly implemented for conflict

resolution: breadth-first, depth-first, and best-first. Other strategies used to select which rules to examine include assigning rule priorities, using the timing of candidacy, the textual position, and the applicability of the rules to the task at hand. (Winston 1984)

Forward and Backward Chaining. In a forward chaining strategy, the rules are searched to reach conclusions from information provided by the user, facts in the knowledge base, and previous conditions. As conclusions are reached, premises or other rules are satisfied, and the search continues until no more conclusions are reached. Since this strategy works from the data to the goal state, it is also called data-driven. Forward chaining is appropriate for problems where the solution is chosen from a very large number of potential solutions, and a small amount of information from the user is available.

In a backward chaining strategy, a goal is selected and then the rules are searched for those rules whose consequent actions match the goal. Backward chaining is also called goal-driven, and is appropriate for problems with a limited number of solutions, or when all the available data does not need to be analyzed.

Most real problems use a combination of both strategies. A fully integrated system allows the expert system developer the flexibility to solve complex problems.

Search Options. The hierarchy of rules can be arranged in a search tree where the search for a solution is a traversal through the tree. A search can identify a single path through the tree, or exhaust all of the possible paths (or solutions) through the tree. Terminating the search when an acceptable solution has been identified is much more efficient, but does not necessarily identify an optimum solution.

In a breadth-first search, the nodes (or rules) are searched layer by layer, one layer at a time. Thus all of the rules at a given depth are examined to see if they match the conditions for the solution, before any of them are expanded. Breadth-first search is most effectively used when most of the solutions are at relatively shallow depths of the tree. When the solutions are fairly deep in the tree, breadth-first requires extensive processing of a large number of layers before any solutions are identified.

For each node in a depth-first search, a path to a lower node is picked, ignoring all alternatives at the same level, thus shooting straight down the tree along any path. When a branch terminates, another path is found, until all alternative paths have been located.

Depth-first search can also be fairly expensive when the solution is located along the last paths identified, or when the first paths are relatively long.

In a best-first search, the next node added to a path is the "best" node available. All the available nodes are examined, and the node to expand is selected according to some criteria. Best-first search is more likely to find the shortest paths than other methods, since it always chooses the node closest to the solution criteria.

TOOLS FOR BUILDING EXPERT SYSTEMS

The expert system developer has a range of tools available to use for the representation and control of the knowledge: programming languages, programming environments, and expert system shells. These tools provide varying levels of support for explanation facilities, graphics, and other features influencing the ease of use of the expert system. Many of the tools provide for interfaces to existing databases, Computer-Aided Drafting (CAD) packages, and to the multitude of analysis software (such as finite element modeling). These interfaces, as well as facilities for knowledge acquisition and uncertainty management significantly impact the ease of development of the system.

PROGRAMMING LANGUAGES. Procedural languages like FORTRAN and BASIC are very effective for programming mathematical, algorithmic tasks, but are not particularly useful for symbolic reasoning. LISP (LISt Processing) and PROLOG (PROgramming LOGic) are generally used by AI programmers. PROLOG, used mainly by European and Japanese programmers, contains constructs to manipulate logical expressions, while LISP has operators to facilitate list processing. C is emerging as an alternative to LISP, due to its portability and ability to interface with existing analysis programs, which are usually written in FORTRAN or C.

PROGRAMMING ENVIRONMENTS. A programming environment is closely associated with a particular language, and contains chunks of the code (similar to subroutine libraries) that are useful for particular tasks. Most environments can also be classified as hybrid tools. Hybrid tools combine a rule-based approach with procedure-oriented programming and object-oriented programming. These tools are well suited to engineering problems, which are generally complex problems requiring a variety of representation schemes.

EXPERT SYSTEM SHELLS. Expert system shells facilitate the rapid development of expert systems. They incorporate specific knowledge representation schemes, inference mechanisms, and control. Early shells were developed by *stripping* the knowledge from an expert system. Many of the available commercial shells have facilities to interface to existing databases and to procedural languages, as well as extensive graphics capabilities.

KNOWLEDGE ACQUISITION. Knowledge acquisition is the transfer of problem-solving expertise from some knowledge source -- human experts, textbooks, databases -- to a program. This expertise is a collection of facts, procedures, and judgmental rules about the domain, and is often very difficult to either extract from a human expert or to represent in a knowledge representation. This task can be automated with inductive inference methods that generate new rules from training examples. The research in this area is in its infancy, however, and has exposed difficulties in achieving consistency, correctness, and completeness in knowledge bases. Computer aids do exist to assist in knowledge acquisition: knowledge-base editors and interfaces, explanation facilities, and knowledge-base revision. Sophisticated editors are being developed that facilitate instruction and check for semantic inconsistencies. These editors, along with a facility to explain the basis for reasoning, affect the acceptance by the user and/or the expert. Semantic consistency checks and automated testing help in updating the knowledge base, to minimize introducing new errors into the expert system. (Buchanan et al. 1983, 149 -157)

UNCERTAINTY MANAGEMENT. The knowledge in the expert system may not be exact. Several methods are commonly used to deal with uncertain or incomplete knowledge: certainty factors, Bayes theorem, and fuzzy logic. Certainty factors are informal measures of confidence; Bayes theorem provides a method for calculating probabilities; and fuzzy logic applies to sets of information with unsharp or 'gray' boundaries. (Bonissone and Tong 1985, 241 - 250)

HARDWARE REQUIREMENTS. Early expert system implementations were on hardware devoted to artificial intelligence tasks, such as Symbolics, LISP Machines Inc. (LMI), or XEROX AI. These specialized machines are relatively expensive and are not very useful for general purpose computing tasks. Expert systems are also available on mainframes, minicomputers, workstations, and PC level machines. Some systems are

available on all levels of hardware, an advantage in developing a system for distribution. A system can be developed on a VAX class machine with a rich development environment, and then implemented on PC class machines at relatively low cost.

PERFORMANCE. Expert system performance is inversely proportional to the number of elements being reasoned about, and is dependent on the knowledge representation scheme, how structured the knowledge base is, and obviously, the hardware chosen. Expert systems are knowledge intensive and have considerable memory requirements. Misusing a tool, i.e., using forward chaining in a backward chaining environment, can significantly impact the performance of the system. In general, an efficiently written production system is more efficient than a hybrid tool using rules.

The performance of a system is also dependent on human factors: ease of use, familiarity, understandability. Its productivity is associated with the ability to provide assistance. Other factors influencing performance are portability and extensibility.

A HISTORY OF EXPERT SYSTEM APPLICATIONS

A brief discussion of some early expert system applications illustrates the historical development of expert systems. Table 2.2 summarizes these applications. More recent works are outlined in Chapter 4.

Table 2.2. Historical Expert System Development

SYSTEM	DATE	DEVELOPER
DENDRAL	1965 - 1979	Buchanan & Feigenbaum Stanford Heuristic Programming Project
MACSYMA	1968 - 1982	Engleman, Martin, & Moses MIT
HEARSAY-I & II	1970 - 1976	Erman, Hayes-Roth, Lesser, & Reddy Carnegie Mellon University
INTERNIST	1974	Pople & Myers University of Pittsburgh
MYCIN	1976	Shortliffe Stanford Heuristic Programming Project
PROSPECTOR	1978	Duda, Gaschnig, Hart, et al. Stanford Research Institute (SRI) International
SACON	1978	Bennett & Engelmores Stanford Heuristic Programming Project
PUFF	1979	Kunz, Aikins, Shortliffe Stanford Heuristic Programming Project
R1 (XCON)	1981	McDermott Carnegie-Mellon University & DEC

DENDRAL originated the fundamental concept of expert systems, manipulating large amounts of expert, heuristic knowledge in a computer program. The program is designed for use by organic chemists to infer the molecular structure of complex organic compounds from their chemical formulas and mass spectrograms. The heuristic knowledge of expert chemists is incorporated into a rule-based system. DENDRAL's success proved that expert systems could be developed and launched researchers on the study of knowledge-based systems. (Buchanan & Feigenbaum 1978)

MACSYMA is a large, interactive computer system designed to assist mathematicians, scientists, and engineers in solving complex mathematical problems. Inputs to MACSYMA are formulas and commands, and outputs are solutions to symbolic problems. MACSYMA is widely used by researchers in government laboratories, universities, and corporations. (Rand 1984)

HEARSAY-I & II are speech understanding systems. Each knowledge source contributes information to a common working memory, or blackboard. HEARSAY-II demonstrated how multiple knowledge sources could be integrated in very complex problem solving. (Erman 1980)

INTERNIST assists a physician in making multiple and complex diagnoses in general internal medicine given a patient's history, symptoms, or laboratory test results. The system is one of the largest medical expert systems developed, and therefore uses a structured approach for the knowledge base. INTERNIST must consider not only a very large number of diseases, it must also consider all the possible combinations or interactions among these diseases. Because of the structure and size of the knowledge base, the program does not perform very well; additional development has been done with the successor, CADUCEUS, to make the program more attractive to physicians. (Pople, Myers, and Miller 1975)

MYCIN is the most famous of the early expert system projects. It diagnoses blood and meningitis infections and recommends appropriate drug treatment, on the basis of an interactive dialogue with a physician about a particular case. Each rule has an associated certainty factor, indicating the expert's level of confidence in the rule. It also has an explanation facility to justify the inferences made by the system. MYCIN exemplifies the

essence of a typical expert system. The developers subsequently built EMYCIN -- an *empty* MYCIN, or MYCIN without its knowledge base. EMYCIN contains all the machinery needed to reason about a knowledge base and to conduct consultations with a user. (Shortliffe 1976)

PROSPECTOR aids the geologist in finding ore deposits from geological data. A combination of rule and semantic networks are used to represent the knowledge. The system contains a knowledge acquisition system (KAS) to facilitate the acquisition of knowledge. Information is either requested from the user, or it can be volunteered. (Duda, Gaschnig, and Hart 1979)

SACON advises engineers on the use of the finite element structural analysis program MARC. SACON was developed using EMYCIN to evaluate EMYCIN's environment for diagnostic applications in other domains. (Bennett and Engelmores 1979)

PUFF diagnoses the presence and severity of lung disease in a patient by interpreting measurements from respiratory tests administered in a pulmonary function laboratory. PUFF was built to demonstrate the practicality of using the shell EMYCIN to prototype additional systems. (Kunz et al. 1978)

R1(XCON) assists in configuring VAX computer systems for Digital Equipment Corporation, and is the largest, most mature rule-based expert system in operation. From a customer's order, R1 decides what components must be added to produce a complete operational system and determines the spatial relationships among all the components. It also outputs a set of diagrams of these relationships. It was developed using a programming environment tool, OPS5. (McDermott 1982)

SUMMARY

AI research has been underway for more than three decades, but it has only been since the late 80's that its impact has been measurable. The most notable and visible results are in the area of expert systems, the implementations of which have exploded in the past several years. To effectively use expert systems, we must understand their capabilities and limitations; they are not the solution to every problem. They are, however, a viable technology providing a new approach for solving many decision problems.

CHAPTER 3

BUILDING A KNOWLEDGE-BASED EXPERT SYSTEM

The first step in developing an expert system application is to determine if the problem is suitable to an expert system. Some problems are better solved by conventional programming tools; other problems exhibit characteristics that are better solved by expert systems. Once an expert system approach is selected, the process of implementing the system, or knowledge engineering, begins. The developer, in this case also called a knowledge engineer, is responsible for acquiring the knowledge and embedding it in an expert system. The knowledge engineer must choose an appropriate tool for the expert system implementation, and then develop a prototype to test the implementation.

CHARACTERISTICS OF EXPERT SYSTEM PROBLEMS

Problems to be solved by expert systems share some important characteristics (Dym 1985, 18; Winston 1987, 15 - 16):

- the domain knowledge is highly subjective, judgmental, and rich in reasoning
- the knowledge cannot necessarily be coded or organized
- an expert is much better at solving the problem than an amateur
- the problem is clearly defined, in a fairly narrow domain; the expert system's complexity will naturally grow as the system evolves
- adequate data is available
- at least one expert is available, and committed, to the project and can explain the reasoning used in solving the problem
- the task is not too easy nor too difficult for the expert to solve; it should take a human expert from 1-12 hours to solve the problem.

Conventional programming techniques have not been successfully applied to problems exhibiting these characteristics, and expert system implementations will not be successful unless these criteria are met.

Even though expert system developers experience some of the same problems that conventional system developers experience, some myths about expert systems have arisen (Fox 1990, 13 - 16). Among these myths are that expert systems do not make mistakes, small prototype systems can be scaled up into full-scale solutions, expert systems can be easily verified and validated and are easy to maintain, and that if an expert exists, an expert system can be created. These myths are worth noting, in order to avoid making mistakes, and thus to build more effective systems.

TASKS

The types of problems that have been solved by expert systems can be classified as: interpretation, diagnosis, monitoring, control, prediction, repair, instruction, planning, and design. These tasks can be grouped into derivation problems and formation problems.

DERIVATION. Most of the early expert systems solved derivation problems. The outcome, or goal, exists in the knowledge base and the solution is to identify the path to the goal. Typical tasks are:

- *Interpretation*. Analyzing data to determine the meaning. The data is often unreliable, erroneous, or extraneous.
- *Diagnosis*. Identifying problem areas or faults based on potentially noisy data. Often the first step is to *interpret* the data which can be incomplete, inexact, or from faulty sensors.
- *Monitoring*. Interpreting signals continuously, or intermittently, and warning when intervention is required.
- *Control*. Adjusting or regulating a system based on signals *monitored*.
- *Prediction*. Inferring likely consequences from given situations.
- *Repair*. Acting to rectify faults in a system. The first step is to *diagnose*.
- *Instruction*. Identifying deficiencies in a student's problem solving knowledge and recommending actions.

FORMATION. Most engineering problems are formation problems, complex procedures where the solution is not already in the knowledge base. The solution space is generally very large, and methods must be implemented to prune the number of likely

outcomes from the solution space. Typical tasks are:

- *Planning*. Creating a program of actions to achieve a goal, subject to specified constraints. Excessive use of resources should be avoided.

- *Design*. Creating objects that satisfy certain specifications. In large design problems, the task is usually divided into a number of subtasks that interact with one another. Priorities must be established for resolving conflicting goals.

In formation problems, since the exact solution does not necessarily exist in the knowledge base, it must be generated by the inference mechanism using the knowledge base. A generate and test method is often used; *all* possible solutions are generated, then tested, until a solution is found that satisfies the goal condition. Another method, problem reduction is also used for formation problems. Problem reduction involves factoring the problem into subproblems (subsystems). Formation problems usually use a hierarchical approach to develop a plan at successive levels of abstraction. They frequently involve backtracking, when no solution exists along the current path, and constraint handling for interaction between the subsystems.

KNOWLEDGE ACQUISITION

The process of extracting knowledge from an expert (or source of expertise) and transferring it to an expert system, knowledge acquisition, is an important and difficult problem. Knowledge acquisition plays a major role in designing an expert system, and is viewed by many authors as a bottleneck in the construction of expert systems.

Buchanan (1983, 140 - 149) has described the following elements as part of the knowledge acquisition process:

- Identification of experts, resources, and knowledge engineers
- Conceptualization of tasks and subtasks, and the techniques used by the expert
- Formalization of concepts by mapping them into representation schemes
- Implementation by encoding knowledge, and iteratively acquiring and testing the system's expertise
 - Testing and refinement of the prototype, by an expert; exhaustive testing is infeasible, due to the combinatorial explosion of the possible solution states.

Eliciting knowledge from the expert is a time consuming process. Often an expert can tell you *what* he does, but not *how* he knows to do it. Encouraging the expert to describe his expertise in the most natural way may help elicit knowledge about the procedures he uses in his problem solving. The expert tends to keep the whole problem in mind, and can find it hard to focus on one sub-issue when several, related sub-issues are also present. Once a prototype system is implemented, it becomes difficult for the expert to distinguish between fundamental problems in the knowledge base and superficial problems in how the program presents information to the user. Several methods have been used in knowledge acquisition (Hart 1985, 456 - 460).

- Interview. The knowledge engineering explores, with the expert, the kinds of data, knowledge, and procedures needed to solve specific problems. This process is difficult to structure, and since the expert is often not explicitly aware of the methods he uses, he often loses interest in the process.
 - Protocol analysis. An expert examines documented cases and talks about them. This is more structured, thus reducing some of the problems with interviews. A variation of this method involves watching the expert solve real problems on the job.
 - Induction. A set of specific examples, a training set, is used to automate the induction of rules or patterns, i.e., machine learning.
 - Repertory grid technique. An expert produces examples and two valued attributes for the examples. The grid is a cross-reference between the examples and the attributes. This technique helps the expert structure and classify the knowledge.
- In each of these techniques, the knowledge engineer often needs to guide the expert in formulating the data and procedures that will produce a relevant, and useful, knowledge base.

To implement the expert's knowledge in the expert system, the knowledge engineer will need to choose an appropriate representation and inference strategy. He must be familiar with the various knowledge representation schemes and inference strategies in order to choose the schemes that best fit the problem.

CHOOSING A TOOL

A multitude of expert system building tools are available (van Koppen 1988; Waterman 1986; Harmon and King 1985; Hayes-Roth, Waterman, and Lenat 1983). Some tools, widely used for expert system development, are summarized in Figures 3.1 and 3.2.

	RULE-BASED	PROCEDURE ORIENTED	FRAME-BASED	OBJECT ORIENTED	FORWARD CHAINING	BACKWARD CHAINING	CERTAINTY HANDLING	KNOWLEDGE ACQUISITION	EXPLANATION	GRAPHICS	LANGUAGE	OTHER FEATURES
ART	●	●	●	●	●	●	●	○	●	●	Common LISP	Hypothetical worlds
HEARSAY-III	●	○	○	○	○	○	○	○	○	○	LISP	Blackboard architecture
KEE	●	●	●	●	●	●	○	○	●	●	Common LISP	External databases
Knowledge Craft	●	●	●	●	●	●	○	○	●	●	Common LISP	Language interfaces
LOOPS	●	●	○	●	○	○	○	○	○	○	Interlisp-D	
OPSS	●	○	○	○	●	○	○	○	○	○	C	Pattern matching
ROSIE	●	●	○	○	○	○	○	○	○	○	Interlisp	English-like syntax
SMALLTALK	○	○	○	●	○	○	○	○	○	●		Highly interactive

Figure 3.1. Programming Environments

Included
 Not included

EXPERT SYSTEM SHELLS (LARGE)	RULE-BASED	FRAME-BASED	OBJECT ORIENTED	FORWARD CHAINING	BACKWARD CHAINING	CERTAINTY HANDLING	KNOWLEDGE ACQUISITION	EXPLANATION FACILITY	GRAPHICS	LANGUAGE	OTHER FEATURES
Concept Modeler	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>		3-D Solids Modeler
EXPERT	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	FORTRAN	Consistency checking
EXPERT-EASE	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	PASCAL	Induction, Examples-based
G2	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	LISP	Real time Animation
GOLDWORKS II	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	Common LISP	Ext. interfaces
GURU	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	C	Nat. Lang rel DBMS
INSIGHT2+	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	PASCAL	Math functions Lang interfaces
M.1	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	C	English-like
Personal Consultant	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	LISP	LISP functions
RULE-MASTER	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	C	Rule induction Lang interfaces
S.i	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	C	Procedure oriented
Smart Model	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>		CAD based
EXPERT SYSTEM SHELLS (SMALL)											
1st Class Fusion	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	PASCAL	Rule induction Lang interfaces
Kappa PC	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>		Lang interfaces
Knowledgepro	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	LISP PASCAL	Hypertext External int.
Level5 Object	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	PASCAL	PRL dBASE
Nexpert Object	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	C	External int.
VP-EXPERT	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	C	External int. Hypertext

Figure 3.2. Expert System Shells

Programming environments give the most flexibility in developing systems, but are considerably more difficult to learn and to implement than expert system shells. An important consideration in choosing an implementation tool, is the knowledge representation and inferencing schemes selected by the knowledge engineer. Many of the newer expert system shells offer a variety of schemes, and are therefore suitable for developing a variety of expert systems applications. Derivation problems are best suited to rule-based systems, formation problems to object-oriented systems.

In general, the developer should select the highest level programming environment possible, typically a hybrid tool. A tool with interfaces to existing algorithmic code may be required for some applications. An expert system shell with graphics capabilities will enhance the development of interactive graphical applications.

DEVELOPING A PROTOTYPE

A prototype system can test the adequacy of the chosen programming tool, the representation of the expert's knowledge, and the strategy for inferences. The prototype should focus on a small set of hypotheses, combine the smallest number of findings necessary to discriminate among the solutions, and include findings that significantly improve the quality of decisions (Weiss and Kulikowski 1984, 106). The system will be developed iteratively, with increasingly sharper and deeper understanding of the expertise.

A large expert system project should be managed as any other large software project, incorporating modularity, top-down design, documentation, and accountability. An obvious observation is that object-oriented systems are more modular and therefore more conducive to top-down design.

Most expert system implementations for engineering applications integrate expert system techniques with procedural code, supported by hybrid tools. Links to appropriate software for computations, database management, spreadsheet analysis, and other existing software tools, enhance the functionality of the system, and reduce the development time.

The user interfaces should receive particular attention, and will require about half of the development time. Features that are available in some tools are windows, gauges, menus, displays, mouse sensitive screen regions, and natural language interfaces.

Different user interfaces for various levels of users, from novice to expert, will augment the usability and acceptability of the system.

An important aspect in developing an expert system is involvement of both the expert and the user. The expert needs to be willing, and the user must be involved in the development. The system must be reviewed with both the expert and the user; they also must be involved in testing and refining the system.

Maintenance of the system will be required. When an expert system stops evolving, the effectiveness of the system begins to decline, since the nature of most problems solved with expert systems changes over time.

VALIDATING THE SYSTEM

Validating an expert system typically involves running test cases and comparing the results against known results or expert opinions. The expert(s) contributing to the expert system knowledge base is a valuable resource for evaluating the tool. However, avoid validating the system against the expert, or test cases, that assisted in the development of the system since this may not identify problems or inconsistencies that were not considered during the development of the system.

Validation methods can be either qualitative or quantitative. Some qualitative methods are: predictive validation, field tests, subsystem validation, sensitivity analysis, visual interaction; quantitative methods include statistical tests and consistency measures. (O'Keefe, Balci, and Smith 1987, 85 - 88) The acceptable performance determined by either method will not be a binary value (yes or no), but will be a range of values.

SUMMARY

The key to successfully implementing an expert system is the knowledge engineer. The knowledge engineer must be able to work with the expert to formalize the knowledge and inference strategies. The knowledge engineer must also be familiar with the available tools in order to effectively develop the representation schemes for the knowledge and to implement the system. In order to implement a viable expert system, the knowledge engineer must be able to obtain the support of the expert and the potential users.

CHAPTER 4

EXPERT SYSTEMS IN ENGINEERING

Problem solving tasks in engineering mainly involve planning and design (or formation type problems). Early expert systems were applied to derivation problems, problems such as medical diagnosis and molecular structure interpretation. Many systems have been developed to monitor and control manufacturing and chemical processing, which are also derivation problems (Maus and Keyes 1991). However, few systems exist for design applications.

Two characteristics separate engineering problem solving from tasks addressed by the early systems. The first is the diversity of the knowledge, a combination of engineering models and scientific principles, information about materials and specifications, and heuristic information. A variety of specialized knowledge representations is needed to depict this diverse knowledge. The second characteristic is the complexity of engineering systems, generally physical systems with many interconnected components.

PROBLEM SOLVING

Algorithmic solutions are applied to well-structured problems. Newell (1969, 365) defines a well-structured problem as one that satisfies the criteria:

- It can be described in terms of numerical variables, scalar and vector quantities.
- The goals to be attained can be specified in terms of a well-defined objective function.
- There exist computational routines (algorithms) that permit the solution to be found and stated in actual numerical terms.

On the other hand, knowledge-based expert systems are well suited to ill-structured problems in a complex domain. Noble (1979, 27) suggests that ill-structured problems can be characterized by some or all of the following: complex, dynamic, ill-defined, political, interactive, uncontrollable, and most importantly, unpredictable.

Many engineering problems are not amenable to algorithmic solutions; they rely on judgment and experience. The problem-solving process involves skillful manipulation of large quantities of knowledge, assumptions, and hypotheses, in a trial and error manner, revising until an acceptable solution is found. These problems are amenable to expert system solutions.

DESIGN METHODOLOGY

Design is a creative process, involving multiple solutions. It is empirical, intuitive, approximate, and most importantly, requires expertise. It also involves quantitative analysis. Several steps in the design process have been identified (Hubka 1982, 62; Pahl and Beitz 1984, 38 - 40; Ullman 1992, 89 - 96) and are illustrated in Figure 4.1.

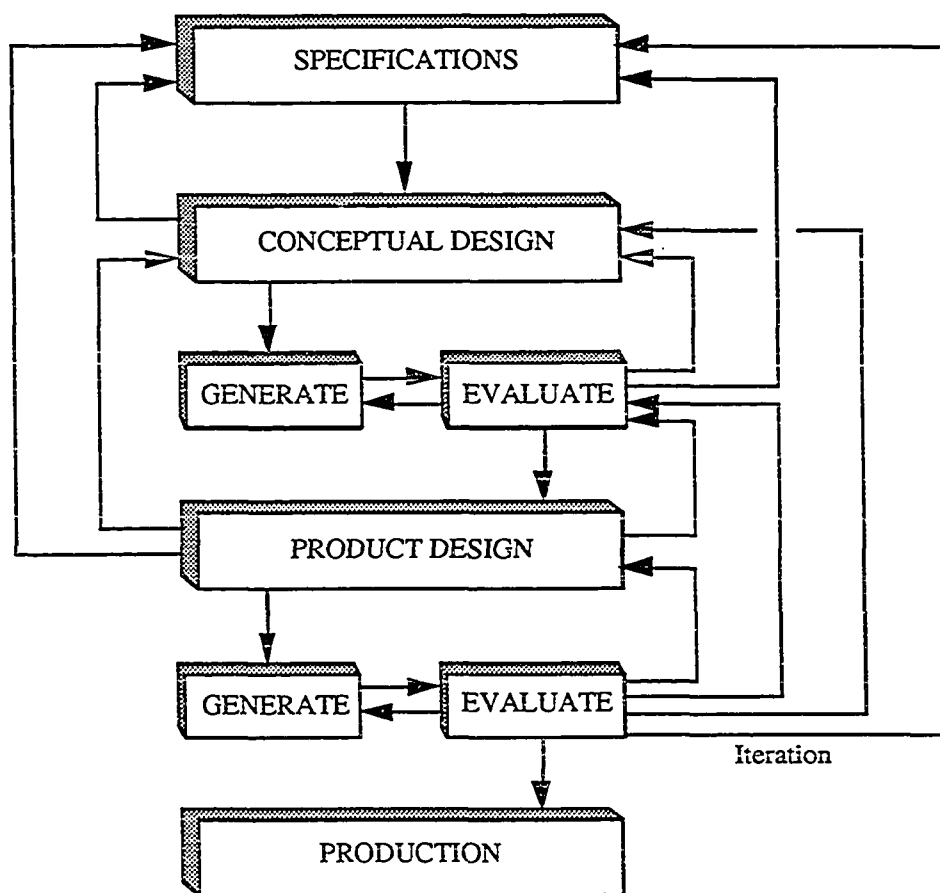


Figure 4.1. Design Process: An Iterative Model

SPECIFICATIONS. The first step in design is to transform the problem into a well-formed set of design specifications, which are the primary communication tool and control mechanisms for engineering design. The initial customer requirements are often ill-defined, imprecise, incomplete, and may even contain conflicting information. They need to be molded into a precise, and quantitative, set of design specifications for an *ideal* description. This description will be used to compare all potential, or conceptual, designs in order to discard inappropriate solutions.

CONCEPTUAL DESIGN. In the conceptual design phase, the product is viewed as a whole, from a functional approach. The individual assemblies and components are treated as *black boxes*, and are described by their functional capabilities -- or what the product does, not by their structural composition -- or how the components work. The conceptual design should identify manageable subsystems to be designed by further refinements.

Creativity is important in generating conceptual designs. Equally important, is the generation of many potential designs. Often a designer will focus on an initial solution, with the high probability that better solutions to the design problem are neglected. The designer must avoid this tendency, as well as the tendency to dismiss unlikely solutions before they have been developed to an extent that can be judged against the specification for the ideal design. The creative process is based on synthesizing personal experience or the experience of outside experts.

Several strategies which can be implemented as CAE tools are used to generate conceptual designs. One strategy involves redesigning or modifying an existing product to meet the new specifications. Another strategy uses existing components and develops new configurations to satisfy the design requirements. Parametric designs are often used by generating conceptual designs using alternate values for the design variables (or parameters).

The conceptual designs are evaluated by comparing the designs to the specifications developed during the first phase of the design process, and then judging the feasibility of each design. This is often accomplished by a rough analysis or by using empirical rules.

The designs should also be assessed for their technological readiness. Generating and evaluating conceptual designs is an iterative process, and should be encouraged at this phase. It is much less expensive to iterate a conceptual design than a product design.

PRODUCT DESIGN. The conceptual design is refined in the product design phase to a fully developed, optimal design. Each component is detailed; and choices for materials, processes and vendors are finalized. A recent trend is to design the product concurrently with the manufacturing process, by a team composed of the designers and manufacturing and materials specialists.

As the product designs are generated, they are evaluated for both performance and cost, and for manufacturability (including the ease of assembly) and maintainability. Experimental and analytical models are used to judge the performance of the design; many automated procedures exist to assist in evaluating the design. Analysis procedures produce only quantitative information about the design; they do not make judgments about what the information means or determine whether the design is good, or how to make it better.

Evaluating the product design may expose limitations in the design that can be eliminated by modifying the design. Iteratively generating and evaluating product designs will result in a better product. Sometimes it may be necessary to return to the conceptual design phase, and generate new possibilities.

The result of the product design is a set of design records: detail and assembly drawings, bill of materials, assembly information, quality control and quality assurance, and instructions for installation, operation, maintenance, and retirement. These records are used to convey the product to manufacturing and to eventually communicate with the customer.

The design process progresses from a general overview of a problem solution to increasingly detailed components, or subtasks, of the problem solution. Design is a highly iterative process of interconnected steps, iterating between synthesis of problem solutions and analysis of those solutions. Specifications may need to be relaxed to accomplish the design; conceptual design models are modified and reevaluated until an optimum design is found. New information is often incorporated after the design process has begun, or new

insights are gained that impact the design. The problem is decomposed into subproblems, then redecomposed; specified and respecified; designed and redesigned.

Hoeltzel and Chieng (1989, 48 - 50) postulate the following systematic design methodology:

- Design procedures propagate gradually from a qualitative domain to a quantitative domain, from synthesis to analysis, estimation to evaluation. Thus, design procedures are generally hierarchical.
- The design process can be separated into a generic portion and a domain-specific portion, and may be further subdivided into a creative design portion and a routine design portion, depending on the coupling of the design variables.
- An abstract design optimization process, based on a hierarchical data structure and monotonic reasoning, is guaranteed to converge during the search for the optimum solution.

Mechanical design involves additional aspects, not necessarily found in other engineering design processes: material selection, sensitivity to manufacturing concerns and processes, complex three-dimensional geometries, and non-modularity.

EXPERT SYSTEMS IN THE DESIGN DOMAIN

Knowledge-based expert systems for mechanical engineering design have been implemented using several approaches. J. R. Dixon and the Mechanical Design Automation Laboratory at the University of Massachusetts use a design-evaluate-redesign approach (Dixon, Simmons, and Cohen 1984). A second approach, used by David Brown at Worcester Polytechnic Institute and B. Chandrasekaran at The Ohio State University, involves design refinement with plan selection and redesign (1984). Another approach, transformation, is proposed by A. S. Kott of the Carnegie Group Inc. and J. H. May at the University of Pittsburgh (1989).

DESIGN - EVALUATE - REDESIGN. This architecture is applied to the design of component parts and small systems where the initial design and each subsequent redesign iteration is a complete design. An initial design is evaluated or analyzed to determine its expected performance in terms of performance parameters that may include cost, function, and manufacturability issues. A decision is made as to the design's acceptability. If the

design is acceptable, the task is complete. If not, the design is redesigned and reevaluated, iteratively. Redesign can ultimately fail; then the process returns to the initial step to relax the requirements. In redesign the analysis results and the reasons for unacceptability are used heuristically to guide the changes.

The four distinct functions in this methodology -- initial design, evaluation, acceptability determination, and redesign -- are each represented by a separate knowledge source. Two other functions -- control and the user interface -- are also represented in separate knowledge sources.

DESIGN REFINEMENT WITH PLAN SELECTION / REDESIGN. Brown and Chandrasekaran separate design into three classes of increasing difficulty and complexity:

- routine design with known *design plans*
- known components but design plans unavailable
- unknown components.

Routine design is accomplished by decomposing the known design plan. Complexity is still a factor in routine design and is related to the number of components and sub-components and the variety of combinations of the design goals. The knowledge sources are identified since the components and subcomponents are known.

The knowledge forms into clusters; it is not a large unstructured collection of rules, all having equal potential for use. The knowledge is a hierarchical organization of:

- conceptual specialists, each with different expertise and a set of plans
- plans, sequence of calls to tasks
- tasks, series of steps
- steps, which make the design decisions.

The system is divided into four stages: requirements validation; rough-design for determining the most important values (e.g., material), thus pruning the design space; design; and redesign by relaxing the requirements with user interaction. Each stage involves plan selection and design refinement.

The interaction between the subsystems is weak, but it is not negligible. Thus routine design is *almost* decomposable, but still requires communication between the

subsystems. Communication between the specialists is through messages across the hierarchical connections.

Failure handling, when a design doesn't work, is a modified form of dependency-directed backtracking controlled by suggestions and failure-handling advice. The user can be a knowledge source for failure handling, as well as for other stages of the program.

TRANSFORMATION. In the transformational process, each step starts with a design state and produces another design state of the same degree of completeness. A portion of the design structure is replaced with a different substructure. This process may operate on more than one component at a time and is used effectively when the design cannot be easily decomposed. An appropriate application is a design that has tightly coupled subcomponents.

IMPLEMENTATION ISSUES. Expert systems developed for design applications must address the design methodology. They require the integration of large amounts of intuitive knowledge, judgment, and experience, as well as quantitative tools. They involve cooperative problem-solving with multiple experts, which can be a set of logically or physically disjoint knowledge sources communicating through a blackboard. Complex design is characterized by a hierarchical model; the design proceeds from a simple, approximate model to increasing complexity, realism, and reliability. The hierarchy of abstraction is from global to detailed design. The consequences of design decisions cannot be predicted until the design has progressed considerably. Redesign is inevitable, thus scheduling of subproblems for redesign is a concern.

Spatial relationships are necessary parameters for the designer and are not easily approximated symbolically or qualitatively. Hybrid systems can effectively bridge between the symbolic and numerical domains. Other implementation considerations are associated with the user interface. The interface should differentiate between novice and expert users and provide an effective means of communication with the user. (Allen et al. 1987, 98)

EXPERT SYSTEM IMPLEMENTATIONS: DESIGN APPLICATIONS

A discussion of expert systems used in selected design tasks illustrates the current state of knowledge-based tools in engineering design applications.

STRUCTURES.

SACON is a consultant for structural engineers on the use of the finite element analysis program MARC. SACON identifies the analysis class of the problem and recommends specific features of the MARC program to activate. SACON is a backward chaining, rule-based system implemented in EMYCIN. (Bennett and Engelmores 1979)

HI-RISE addresses the preliminary structural design of buildings. HI-RISE configures and evaluates several alternative structural systems for a given three-dimensional grid. A combination of frame-based and rule-based reasoning is implemented in PSRL, a language developed at Carnegie-Mellon University. Rules in PSRL are expressed in an extension of the OPS5 language syntax; a LISP-based declarative formalism is used to represent the structured objects. HI-RISE is an early application exploring the use of expert systems for design problems. (Maher and Fenves 1984)

Composite Design Assistant coordinates access to a database manager for material properties and to analysis codes for design of sandwich panels. CDA is written in PROLOG, while the interfaces to the databases and analysis codes are written in FORTRAN. (Zumsteg, Pecora, and Pecora 1985)

BEADS, a prototype Building Envelope Analysis and Design System, assists the designer in selecting materials and constructional systems. A knowledge base containing information on performance requirements and constraints from building codes is interfaced with a database of material properties. BEADS is implemented as a framed-based system using Knowledge Craft. (Fazio, Bedard, and Gowri 1989)

FRAMEX is an integrated system for simulating the design process of rectangular multistory steel buildings, using numerical processing, symbolic processing, and database management. FRAMEX is implemented as a rule-based system, using Personal Consultant Plus, with graphical user interfaces and interfaces to analysis software written in Turbo Pascal. (Adeli and Chen 1989)

IBDE, Integrated Building Design Environment, is a prototype environment of processes and information flows for the vertical integration of architectural design, structural design and analysis, and construction planning. The processes are knowledge

based expert systems using declarative or rule-based knowledge representations: architectural planner ARCHPLAN, space planner for service core CORE, structural system configurer STRYPES, structural layout and approximate analysis system STANLAY, component designer SPEX, foundation designer FOOTER, and construction planner CONSTRUCTION PLANEX. A blackboard architecture is used to coordinate communication between the processes, and the global information is organized in an object oriented programming language. (Fenves et al. 1990)

EXPERT-SEISD is an object based rule system for the preliminary design of beam and plate components. The system consists of a design module and a knowledge acquisition module for updating and/or expansion of the knowledge base and database. EXPERT-SEISD is implemented in GCLISP, a PC version of Common LISP developed by Gold Hill Computers, Inc. (Umaratiya and Joshi 1992)

COKE, Construction Knowledge Expert, provides feedback on the constructability of the structural design of a reinforced concrete building structure. COKE reasons about the geometrical and topological model of a designed facility and provides construction input for the structure. COKE incorporates the data from AUTOCAD with Kappa PC to build a symbolic model of the project's structure. The system links the requirements of construction methods with structural design decisions to determine the constructability of a design. (Fischer 1993)

STANDARDS.

SPECON aids the structural engineer in checking structural steel elements for conformance with the AISC Steel Design Specification. The essential difference between SPECON and other expert systems is the flexibility provided to the user to alter numerical values of design parameters until the hypothesis is satisfied. An explanation module informs the user how certain deductions were made or why a particular question was asked. SPECON is a backward chaining production system, implemented in LISP and OPS5. (Sriram, Maher, and Fenves 1985, 5-6)

SICAD is a rule-based approach for checking designed components for conformance with applicable standards. SICAD integrates conformance checking with

procedural programs for structural analysis, database management, standards, and synthesis components. SICAD, a hybrid system incorporating a blackboard architecture, is implemented in POLO, a FORTRAN based language translation facility and comprehensive engineering database manager with an associated analysis package. (Lopez, Elam, and Reed 1989)

HyperLRFD is a prototype system developed to evaluate the feasibility and practicality of a unified Object-Logic model for the representation of design codes and the processing of design standards. HyperLRFD incorporates parts of the AISC Load and Resistance Factor Design (LRFD) specification and performs conformance checking and component design. The organizational aspects of the design standards are represented with an object-oriented paradigm while the reasoning mechanisms for the design are implemented in logic programming. HyperLRFD is implemented in PROLOG++ (object-oriented extension of PROLOG) and uses HyperCard (Hypertext software for Macintosh computers) to implement the user interface; HyperLRFD interfaces to an Oracle relational database system and Excel spreadsheet software. (Yabuki and Law 1993).

MECHANICAL.

VEXPERT designs standard V-belt drives. VEXPERT was implemented to demonstrate the design-evaluate-redesign architecture. A design algorithm is used to obtain an initial design from problem specifications. Utility-decision algorithms are used for analysis and acceptability. VEXPERT is written in LISP, uses OPS5 production rules, and a blackboard implementation scheme. (Dixon and Simmons 1984)

XENIF designs aluminum extruded rectangular heat fin arrays for natural convection heat transfer. XENIF was implemented to demonstrate the design-evaluate-redesign architecture, based on dependencies, or relationships, between design goals and design variables. XENIF is written in DELPHI, a General Electric proprietary expert system language, and uses rules written in LISP. It uses FORTRAN utilities for analysis. (Kulkarni et al. 1985)

AIR-CYL is an application of a general purpose design expert system, designing air cylinders for a given set of requirements. It was implemented as a demonstration of a

hierarchically structured system with plan selection for routine design. AIR-CYL was developed using the task-level DSPL language, Design Specialists and Plans Language, which is based on a LISP dialect. (Brown and Chandrasekaran 1986)

PRIDE designs paper-handling systems inside copiers and duplicators, organizing the knowledge as design plans. The plans decompose the problem into simpler parts. A problem solver executes these plans and uses dependency-directed backtracking with an advice mechanism to handle constraint failures. (Mittal, Dym, and Morjaria 1986)

DPMED selects design parameters for mechanical primitives such as gear-pairs, v-belts, bearings and shafts. DPMED incorporates rules for selecting materials and critical design criteria, and a database of standard values of design parameters. DPMED uses *Refinement + Constraint Propagation + Parameter Selection*. As each sub-module is designed, constraints are propagated to the other sub-modules to guide their design. DPMED was implemented in KEE, an object oriented environment. (Ramachandran, Shah, and Langrana 1988)

A prototype expert system for the gating design of an investment casting process incorporates a “design-with-features” approach. The prototype uses an object oriented structure, implemented in KEE, to manipulate features for geometric reasoning and interfaces to the CAEDS solid modeler. Communication between the systems is through Common LISP. (Chung et al. 1988)

XCUT is a feature language which generates process plans for the production of machines parts. XCUT couples rule-based and object-oriented programming techniques for automatic classification of machine features. (Hummel 1989)

MEFDES, Modular Element Fixture Design Expert System, interfaces a 3-D CAD system (ME30) with a feature recognizer, which analyzes the part geometry and extracts machining features, to determine fixture setups for prismatic parts. The rule / frame-based system is implemented with Nexpert Object, an expert system shell. (kumar, Nee, and Prombanpong 1992)

An integrated system combining conventional expert system methodology with operations research decision-analysis techniques has been applied to material selection in

automobile bumpers by Thurston (1993). The heuristic rules are separated into two categories: subjective rules, which embed assumptions about balancing conflicting design objectives and user preferences, and objective rules, comprised of factual information and which do not typically vary between designers. The objective rule base is used to identify a set of design alternatives that satisfy minimum performance requirements by eliminating those alternatives not falling within specified design parameters and configuration constraints. A user manipulated utility function, incorporating multiattribute utility analysis, then evaluates and ranks each alternative. The knowledge base was constructed in OPS5, and the utility function expert system module was written in Common LISP.

A system for parametric design and analysis of a family of parts with a specific focus on gas turbine nozzles has been developed with Smart Model, a knowledge-based engineering system from ICAD, and integrated with software utilities developed by General Electric. These utilities include a geometric modeling utility, TAGUS; an automatic 2-D mesh generator, QUADTREE; and a lofting type mesh generator for extruded components, EXTREME. The Smart Model knowledge-based system uses an object-oriented framework to represent the design and manufacturing information as part of the complete product definition of parts, assemblies, and systems. (Saxena and Irani 1993)

ALPRO incorporates design compatibility analysis, which ranks manufacturing processes based on feasibility for the basic geometry, material, and production requirements of components, with normalized cost analysis. The prototype addresses aluminum processes: extrusion, sheet forming, forging, die casting, permanent mold casting, sand casting; coupled with the secondary processes of bending and machining. An object oriented representation is used for the capability data; the program uses HyperCard as a front-end, PROLOG for logic-based analysis, and Excel for cost calculations. (Yu et al. 1993)

INJECTION MOLDING.

IMPARD evaluates designs of injection molded parts based on manufacturability criteria such as wall thickness, corner radii, boss and hole dimensions, melt flow length, taper angles, and draft angles. IMPARD interfaces to the GeoMod database, a solid

modeler developed by SDRC. Primitive features are input by the designer and used for visual displays and design evaluations (Vaghul et al. 1985)

A prototype knowledge-based synthesis system for injection molding is presented by Kim and Suh (1986). It combines a rule-based system with a cavity filling simulation program. Theoretical models predict the moldability of the design and the mechanical performance of the molded part. The user interacts with the design loop to synthesize designs in terms of gate location and molding conditions. The design is evaluated for moldability and strength. The prototype was implemented using EXPERT.

GERES is an expert system for selecting injection-molded resins based on pre-design application information. GERES requests nontechnical, symbolic design attributes, prioritized by the user, to guide the material search. The program selects technically feasible resins and ranks the selections by cost; the program also “relaxes” non-critical needs to find economically feasible alternatives. GERES is implemented in Delphi, a GE proprietary product, and uses rules, object-attributes-value triples, and LISP procedures. (Nielsen, Dixon, and Simmons 1986)

AMDS, Automated Mold Design System, integrates the Moldflow analysis program, features database, and iterative redesign to automate the design of injection molds. The features database represents the part and the feed system. The quality of the design is based on performance parameters. (Irani, Kim, and Dixon 1989)

IMCE, Injection Molding Cooling Expert, is a hybrid expert system for the design of the cooling system for injection molding. IMCE uses the heuristic-depth-first searching algorithm for redesign. An interactive graphics program is used to create/edit the two-dimensional geometric model, and the numerical model. The cooling process for the numerical model is analyzed. Databases contain material properties, cooling rules, and data for the analysis programs. The user can interact with the redesign stage to modify the design variables. IMCE was developed in Common LISP under the expert system shell, KEE. (Lee and Kwon 1989)

Dennis Pearce developed an expert system to estimate the cost and configuration of injection molds for plastic parts. (1989)

Ishii, Hornberger, and Liou (1989) have developed an expert system prototype to evaluate a candidate design using design compatibility analysis among user requirements, process constraints, and the part design. Suggestions for improving the design are presented both graphically and in text format. The prototype uses DAISIE (see the next section) as a platform for the compatibility analysis and is implemented in PROLOG with a HyperCard user interface.

GENERAL PURPOSE.

GEPSE is a General Engineering Problem Solving Environment. GEPSE is an object network language that simplifies the construction of object and rule bases. Other features are function libraries, user interface packages, and a facility for meta-level control. GEPSE is a forward chaining system, and is implemented in C. (Chehayeb et al. 1985)

KADBASE is a knowledge-aided database management system prototype. It is a flexible interface for multiple databases and knowledge-based systems to communicate as independent, self-descriptive components within a loosely coupled distributed system. KADBASE provides the mechanism to develop a distributed, integrated CAD system; it uses a frame representation scheme and forward and backward chaining inferencing in a blackboard model. KADBASE is implemented in Franz LISP. (Rehak and Howard 1985)

DOMINIC I performs design by iterative redesign in a domain independent environment, using a hill climbing algorithm. The class of redesign problems for DOMINIC I are those that are intellectually manageable and solvable without sub-division into smaller parts. DOMINIC I contains a knowledge acquisition module and is implemented in Common LISP. (Dixon et al. 1986)

DAISIE, Designer's Aid for Simultaneous Engineering, uses design compatibility analysis to evaluate a conceptual mechanical design for compatibility with various life-cycle issues. The knowledge bases represent issues such as functionality, esthetics, and manufacturability, important in mechanical design. The system evaluates the design while the designer makes tradeoffs and the final decisions based on suggestions from the system. DAISIE is a shell for mechanical design and is implemented in an object oriented environment using PROLOG and HyperCard for the user interface. (Adler and Ishii 1989)

IDS, Intelligent Design System, is an integrated design for manufacture environment that advises the user on the feasibility of a "design-with-features" approach. IDS is based on a CAD system and is integrated with an expert system and a database management system which uses three distinct classifications of information: object-oriented CAD data, a design catalog, and a knowledge base of rules and heuristics. IDS is implemented in the C programming language and interfaces to a CLIPS expert system shell and an Oracle database management system. The interfaces are written in the C programming language. (Miller and Colton 1992)

IES, Integrated Engineering Shell, is a framed-based expert system shell incorporating a blackboard architecture and a database management system. IES provides backward chaining, forward chaining, and hybrid chaining inferencing strategies. IES is implemented in the C programming language. (Sakthivel and Kalyanaraman 1993)

ACL, Agent Communication Language, is an agent-based framework for the development of integrated facility engineering environments. The design agents, various software programs for design and planning systems, communicate design information to facilitators in a federation architecture having no central database. Messages, based on first-order predicate logic, are used to communicate information. (Khedro, Genesereth, and Teicholz 1993)

KASE, Knowledge Assisted Software Engineering, is a set of tools for software analysts and designers at the architecture level. KASE captures the various knowledge needed for design and applies the knowledge to aid knowledge engineers in automating design activities. KASE is implemented in a blackboard architecture for a class of tracking problems, in which the task is to identify and track objects in space based on signal data. (Nii 1994)

RESEARCH AREAS

The current expert system paradigm does not suffice for real world engineering problems. The early expert system implementations for derivation problems are not directly extensible to formation problems. Several systems have been developed for design problems; however, these implementations have also exposed limitations in the current

methodologies. Some specific areas that need further research are (Lu 1986, 14 - 18):

- knowledge acquisition
- inductive reasoning
- limited, narrow problem domains
- integrating heuristic and deterministic knowledge (engineering models, physical principles, and governing equations)
- interactive user interfaces
- use of system shells for domain-specific, task-independent applications (design shell, diagnosis shell, planning shell, etc.)

This research focuses on the last three areas.

SUMMARY

Early expert systems addressed derivation problems, i.e., they look for a path that leads to a specified goal, which exists in the knowledge base. The expert systems discussed in the previous section differ from these early systems since they address engineering design problems, problems that require a diversity of knowledge bases for complex engineering systems, where the solution is not already in the knowledge base. However, most of these expert systems are either implemented in programming languages, requiring many man-months of development effort, or in programming environments requiring many months of training before a developer gains the requisite knowledge to use the tool effectively. This research is cognizant of the limitations of these implementations, particularly those most closely related, and investigates the use of expert system shells for design problems.

Expert systems for engineering design applications require an integration of heuristic and deterministic knowledge. They also involve cooperative problem solving using multiple experts. Hybrid systems have proven to be valuable tools in implementing these engineering systems.

CHAPTER 5

PROTOTYPE DEVELOPMENT

The diversity of the knowledge in engineering problem solving and the complexity of engineering systems lead to many difficulties in applying expert systems to design problems. The knowledge is a combination of heuristic information (design rules and guidelines), and deterministic knowledge (engineering models, scientific principles, governing equations, information about materials and specifications, and analysis data from existing algorithms). The knowledge is provided by multiple sources, requiring a variety of specialized knowledge representations, which need to be integrated for fully functioning systems.

Tools currently exist that are appropriate for developing expert systems for complex tasks such as engineering design. Expert system shells, in particular, offer rich development environments with interfaces to programming languages, access to databases, and graphical capabilities to assist in developing user interfaces. In order to test the capabilities of various KBES shells, a prototype system should be constructed to categorize the knowledge used in design processes and develop representations for that knowledge. The prototype should integrate the knowledge sources with existing databases and analysis software and demonstrate graphical user interfaces for explanation and knowledge acquisition facilities, as well as interactive capabilities for user participation in the design process.

Designing an injection molded plastic part is a representative engineering design problem; a subproblem, the design of a cantilever snap joint to join two components, was chosen for a detailed prototype implementation. The knowledge structures required for a snap joint are typical of the structures in a general engineering design problem; the prototype involves a materials database, design specifications, equations for analyzing the design, and heuristics or rules of thumb.

CHOOSING AN EXPERT SYSTEM BUILDING TOOL

A range of tools with varying levels of support for knowledge acquisition, explanation facilities, (interactive) graphics, uncertainty management, and other features influencing the ease of use of the expert system are available to the expert system developer. The expert systems that have been developed for design problems have generally been implemented using programming languages or programming environments. These tools are relatively difficult to use and are practically useless to the typical engineer with limited programming skills. Consequently, the use of these tools is limited to knowledge engineers having a thorough understanding of knowledge representation schemes and inference mechanisms.

Expert system shells, at the high end of the available tools, facilitate rapid development of expert systems because they incorporate specific knowledge representation schemes, inference mechanisms, and control. Since they often provide interfaces to existing databases and to procedural languages, the developer can interface the expert system with solid modeling systems and a multitude of existing analysis software (such as finite element modeling). Expert system shells typically offer a graphical interface and an explanation facility to encourage user acceptance. Another useful feature to look for in expert system shells is a knowledge acquisition facility to help ensure that the expert system will continue to evolve and will continue to be used. Most shells provide an interactive interface which allows the user to participate in the process, thus serving experts as a design aid and novices as a tutor.

Several shells currently exist that are appropriate for developing expert systems for complex tasks such as engineering design: Kappa PC, Level5 Object, Concept Modeller, G2, and Smart Model.

Kappa PC (IntelliCorp \$3500) offers object-oriented capabilities coupled with a forward and backward chaining rule system, procedural language programming, dynamic presentation graphics, graphical debugging tools, and intelligent links to other applications and databases. Kappa PC is based on KEE.

Level5 Object (Information Builders Inc. \$995) is a hybrid tool which features object-oriented capabilities and includes such functions as forward and backward chaining

inference engines, relational database models, CASE facilities, a graphical toolbox for building user interfaces, and graphical debugging tools.

Concept Modeller (Wisdom Systems, \$65,000) creates 3-D solid models using parametric design capabilities and maintains engineering and manufacturing information, such as weight, cost, list price, and material type, in object-oriented databases to provide part summaries, bill-of-material reports, and data for finite element analysis programs.

G2 (Gensym Corp., \$10,000 - \$40,000+) provides an applications environment for real time processes using a frame-based knowledge representation system with extensions for object-oriented programming. Other features of G2 include interactive windows, graphics, and animation; a structured English editor; functions and procedures; and a dynamic simulator.

Smart Model (ICAD Inc., \$35,000 - \$150,000) incorporates rules to extend the traditional CAD programs based on interactive geometric modeling systems. These rules are used to create a representation of a part that includes product structure and dependence on other parts; physical and geometric specifications; material, manufacturing and cost constraints; lead times; and manufacturing process plans. The system also includes a full-surface modeling system and features for automatically performing and displaying design iterations, relating the design knowledge base to manufacturing or processing knowledge bases, and transferring data from other CAD systems.

All five tools offer rich development environments with interfaces to programming languages, access to databases, and graphical capabilities to assist in developing user interfaces. The first two -- Kappa PC and Level5 Object -- are PC tools, while the other three are workstation tools. Kappa PC is a promising tool, principally because the relatively low cost makes the system accessible for most implementations and it offers a migration path to the workstation environment. Other important advantages are the object-oriented data representation scheme, C programming capabilities, and a large inventory of graphical tools. Both forward and backward reasoning are available, using either depth-first, breadth-first, or best-first search algorithms. IntelliCorp also offers a similar, but not compatible, workstation product -- ProKappa. The object portions of the knowledge can be ported with minimal effort, but the rules and other interface portions require conversion

from one syntax to another. Level5 Object, available at an even lower cost, was conceived and implemented as a procedure-based tool for database applications. Since the object-oriented features are add-ons, and are not an integral part of the shell, Level5 Object is not as viable as Kappa PC for object-oriented applications. Kappa PC was evaluated for its ease of use in design applications, its capabilities for interfacing to existing software and databases, and the tools for developing sophisticated user interfaces.

KAPPA PC DEVELOPERS ENVIRONMENT

Kappa PC offers the developer a rich environment of development tools for viewing and modifying elements, building customized displays, and debugging the expert system. These tools include graphical representations of the knowledge elements, editors and syntax checkers, and functions to read and write ASCII files so the developer can access the myriad of software available in DOS and WINDOWS environments.

Among the development tools are the knowledge editors used to define, examine, and modify the seven knowledge elements -- classes, instances, slots, methods, functions, rules, and goals. Kappa PC also provides graphical presentation tools, from the *ActiveImages™ package*, to facilitate user interface development and enhance the interface features. The tools (Figure 5.1) used to display static information and current information stored in single and multiple valued slots include options for text, transcripts or boxed information, line drawings, bitmap diagrams, buttons, state boxes, meters, sliders, user-



Figure 5.1. Graphical Presentation Tools

edited text, and line plots. Another tool, dialog boxes, provide menus to direct user interactions, post messages, and forms to enter information into knowledge bases.

An alternative development environment to the knowledge editors is the KAL interpreter, a mechanism for testing and executing expressions written in Kappa PC's programming language KAL. The interpreter includes a library of 240 built-in functions to create, access, and modify knowledge elements; evaluate math, logical, and string expressions; control blocks of expressions; manipulate lists, files, databases, and spreadsheets; control knowledge processing; and control the end-user graphical screen interface. User defined functions, written in C to create new functionality, control processing, and combine individual actions, can be tested in the interpreter and added to the knowledge base as functions.

Several tools provide fairly extensive debugging capabilities: an object browser, rule relations browser, an inference browser, and rule tracing. The object browser provides a graphical view of the hierarchy of classes, instances, and subclasses and allows the developer to modify objects and their relationships. Instances and subclasses can be hidden to compact the graphical representation. The rule relations browser graphically displays the linking relationships between premises and conclusions of rules. The rule tracing is a dynamic text description of the inference engine's progress; it lists the rules that the inference engine invokes and the changes to selected slots in the knowledge base due to the reasoning. Thus the developer can see how the system generates new conclusions and can trace the source of errors in the knowledge base. The inference browser graphically depicts the reasoning given by rule tracing.

APPLICATION: CANTILEVER SNAP JOINTS

Snap joints are a simple, economical, and rapid way of joining two different components. All snap joints have a protruding part of one component, e.g., hook, stud or bead, which is deflected briefly during the joining operation and catches in a depression (undercut) in the mating component (Figure 5.2). After the joining operation, the joint should return to a stress-free condition. The joint may be separable or inseparable depending on the shape of the undercut. The force required to separate the components varies greatly according to the design. Two important factors to consider in designing snap

joints are the mechanical load during the assembly operation and the force required for assembly. (Miles 1992)

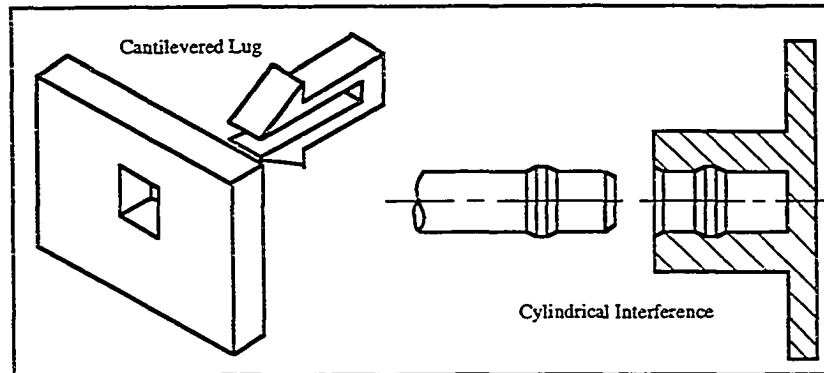


Figure 5.2. Representative Snap Joints

A typical cantilever snap joint is illustrated in Figure 5.3. Recommended design procedures are to vary the finger so either the thickness (h) or width (b) tapers from the root to the hook. Good results are obtained by reducing the thickness linearly by a factor of $1/2$ from the root to the hook, or by reducing the width to $1/4$ from the root to the hook. (Reiff 1991, 60)

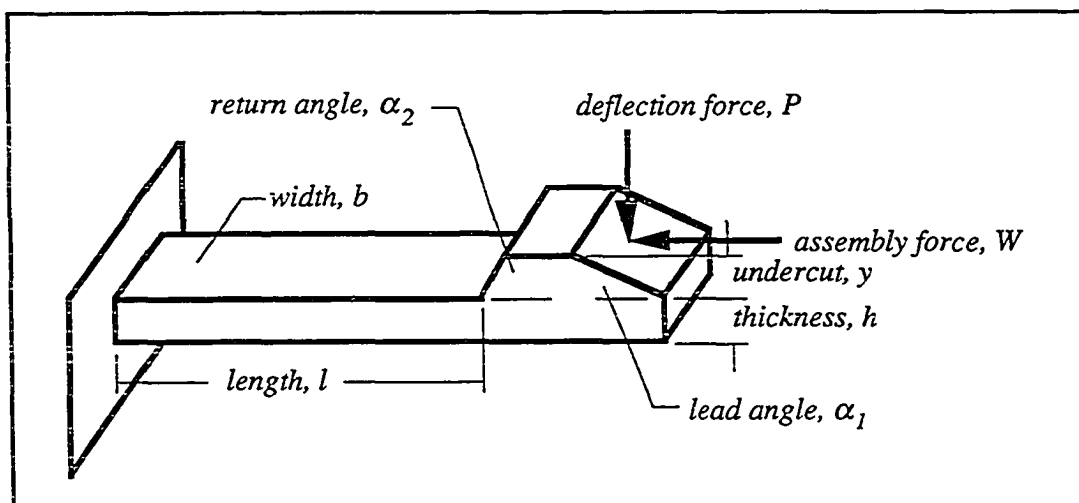


Figure 5.3. Cantilever Snap Joint Geometry

Cantilever snap joints are predominantly loaded in bending. From classical beam theory, the following equations apply to cantilevered beams with constant, rectangular cross sections:

$$\text{Stress: } \sigma = \frac{Mc}{I} \quad \text{where } M = Pl$$

$$\text{Deflection: } y = \frac{Pl^3}{3EI}$$

$$I = \frac{bh^3}{12} \quad \text{and} \quad c = \frac{h}{2}$$

For a cantilever snap joint (Figure 5.3), the following governing equations are then derived from the above equations:

$$\text{Strain: } \varepsilon = \frac{3}{2} \cdot \left(\frac{yh}{l^2} \right)$$

$$\text{Deflection Force: } P = \frac{bh^2}{6} \cdot \frac{E_s \varepsilon}{l}$$

$$\text{Assembly Force: } W = P \cdot \left(\frac{\mu + \tan \alpha}{1 - \mu \tan \alpha} \right)$$

Where:

- ε strain in outer fiber at the root
- y deflection or undercut
- l length of cantilever arm
- h thickness at root
- b width at root
- E_s secant modulus
- μ static coefficient of friction
- α angle of inclination (either lead or return)

The calculated strain is compared to the allowable strain. For amorphous materials the allowable strain is approximately 70% of the yield strain; the working value for strain should be limited to 60% of the allowable strain when the snap joint is to be separated and reassembled several times (Miles 1992, 12). For example, if the elongation at yield is 6.5%, then the allowable strain is 0.0455 for a single assembly or 0.0273 for multiple assemblies.

KNOWLEDGE REPRESENTATION

A variety of knowledge is required in designing a cantilever snap joint. The cantilever configuration and geometry must be specified, either from geometrical constraints of the part or from design specifications and rules. A material must be selected for the part, and the appropriate material properties must be available to the designer. Finally, an analysis, using governing equations and material properties, will determine the structural integrity of the snap joint. Economic factors are generally considered in a design problem but were not included in the prototype since their categorization is similar to that of design rules.

OBJECTS.

The knowledge characterizations of an injection molded plastic part can be effectively represented in an object-oriented environment. In an object-oriented program, the data is represented by objects typified by two types of information: information describing the objects (classes, subclasses, and instances and their attributes) and information specifying what the objects can do (methods). For the prototype design problem, classes are established for three different knowledge types: materials, features, and the design solution; Figure 5.4 graphically represents the object hierarchy. In Kappa PC instances are related to a class by dashed lines; thus, the material class has six instances. A solid line indicates a relationship between a class and its subclasses; the snap joint class has three subclasses -- cantilever, torsional, and annular -- providing future extensions for torsional and annular snap joints.

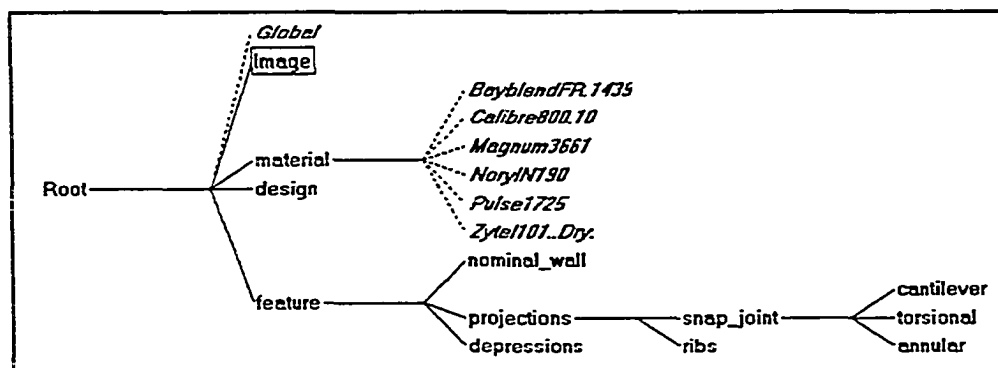


Figure 5.4. Object Hierarchy

Methods are attached to the classes or instances and associate behaviors with these objects. Generally algorithmic processes, involving few conditions in a predetermined series of steps are better suited to implementation as methods than as rules.

A representative sample of the materials and the properties that can be obtained from the material database is given in Table 5.1.

Table 5.1. Materials

Name: Type:	CALIBRE 800-10 polycarbonate resin	MAGNUM 3661 ABS resin	PULSE 1725 polycarbonate / ABS resin
Tensile Stress @ yield (psi):	8,700	5,000	8,400
Comp. Stress @ yield (psi):	14,000	6,900	11,000
Elongation @ yield (%):	6.5	2.3	4.0
Flexural modulus (psi):	360,000	340,000	400,000
Coefficient of Friction:			
Plastic to Plastic	.55	.75	.65
Plastic to Metal	.45	.65	.55

Each class, subclass, or instance is characterized by various slots; for example, each material instance has the following properties: tensile stress, allowable compressive stress, elongation at yield, flexural modulus, and coefficients of friction for plastic on plastic or metal. A listing of the classes, instances, and slot values is available in Appendix A.

METHODS.

The knowledge necessary to design a cantilever snap joint is the analysis information. In a large design problem, tools such as finite element analysis are required to predict the performance of the design. In a relatively small design problem like a cantilever snap joint, the beam theory governing equations in the previous section will adequately predict the cantilever performance. Methods attached to the design solution class use the governing equations to calculate the strain and the assembly forces (listed in Appendix A).

RULES.

Experience with injection molded parts has resulted in a host of design rules applicable to a variety of situations. These rules guide the material selection and the specification of features such as draft angles, surface textures, corner radii, and wall thicknesses, all affecting the moldability of a part, its structural stability and appearance, manufacturability, and the total production cost. The knowledge for the prototype system was acquired from multiple "experts" -- design manuals produced by plastics manufacturers: Borg-Warner, Dupont, and Miles; and plastics designers: Beall and Palsulich.

A cantilever snap joint is a projection, and therefore rules pertaining to projections are appropriate. Some representative rules for projections are:

- length should be less than three times the nominal wall, to avoid molding problems
- thickness should be within 50% to 70% of the nominal wall, to avoid sink marks
- ratio of length to thickness should be less than 10, to avoid buckling
- ratio of thickness to the width should be 1:4, slender beam theory assumption
- deflection angle should be less than 10° , slender beam theory assumption

and rules specific to cantilever snap joints are:

- ratio of length to thickness is 5.4:1, determined from a random sampling of latch geometries
- undercut should be less than one-half the length
- the lead angle should be between 10° and 35°
- the return angle should be greater than the lead angle
- for a self-locking joint, the return angle should be greater than $(90^\circ - \tan^{-1}\mu)$
- if the strain is excessive, reduce the undercut or increase the length

The first seven are implemented as methods (listed in Appendix A), attached to the cantilever class, to calculate the geometrical data and assign the values to the slots -- length, thickness, width, and undercut. (The other slot values are specified through the user interface.) The remaining logical relationships are implemented as IF-THEN rules. Pseudocode examples are given below, and the complete rules are listed in Appendix A.

If ($\epsilon >$ allowable strain)
 then reduce undercut and recalculate design values.

If ($\epsilon >$ allowable strain)
 then increase cantilever length and recalculate design values.
 If ($W >$ allowable assembly force)
 then increase cantilever length and recalculate design values.
 If ($W >$ allowable assembly force)
 then reduce angle and recalculate design values.
 If (lead angle $< 10^\circ$ or $> 35^\circ$)
 then change angle and recalculate design values.
 If (return angle $<$ lead angle)
 then change return angle and recalculate design values.
 If (joint self-locking & return angle $< 90^\circ - \tan^{-1} \mu$)
 then increase return angle.
 If (joint not self-locking & return angle $> 90^\circ - \tan^{-1} \mu$)
 then decrease return angle.

USER INTERFACE

In Kappa PC, the user interacts with the expert system application through the SESSION window (illustrated in Figures 5.5 and 5.6). The user selects an injection molding feature (Figure 5.5) and then chooses options to initialize or perform the design by using *buttons*.

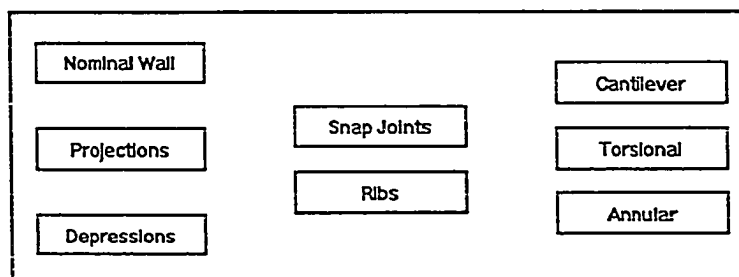


Figure 5.5 Feature Selection

In Figure 5.6 the buttons are located to the right of the diagram and allow the user to specify the initial design configuration, change the resulting geometry, select the material, and perform the design operation. A small number of functions were written to control the user interface and are listed in Appendix A.

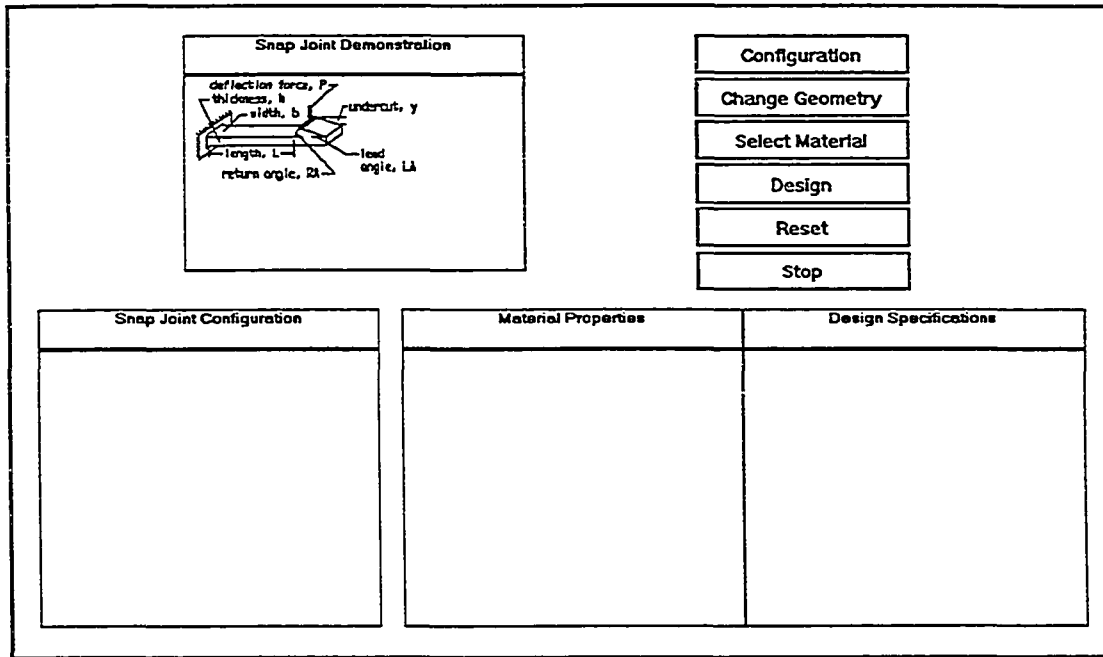


Figure 5.6 Design Interface

To initialize the design geometry, the user clicks the CONFIGURATION button which executes a method attached to the cantilever class. This method asks the user to specify the initial design configuration (Figure 5.7) and applies design heuristics to generate an initial geometry for the cantilever. For some entries, the user is provided a list of appropriate responses, which is obtained by clicking the arrow on the right hand side of the menu (e.g., “Select component types” prompt). Kappa PC also checks the values entered by the user and limits the entries to ranges specified by the developer.

Initial design configuration

Select type of geometry

Select number of assemblies

Select component types

Enter nominal wall thickness

Enter lead angle

Enter return angle

Is snap self locking?

Enter maximum length of cantilever

Enter maximum mating force

Enter maximum separating force

Figure 5.7. Initializing Cantilever Configuration

The user can change the initial geometry by clicking on the CHANGE GEOMETRY button, which executes another method attached to the cantilever class. This method provides the user an opportunity to change any, or all, of the geometry data (Figure 5.8).

Geometry data

Enter length

Enter width

Enter thickness

Enter undercut

Enter lead angle

Enter return angle

Figure 5.8. Entering Geometry Data

Finally, the user makes a material choice by clicking the **SELECT MATERIAL** button. This action provides a list of all the material instances, and asks the user to select one. Each of the actions -- **CONFIGURATION**, **CHANGE GEOMETRY**, and **SELECT MATERIAL** -- also outputs the resulting values, or properties. The user can review the output and continue to make changes, enabling an expert designer to interact with the expert system to specify a configuration that is close to meeting the design specifications. A novice can merely use the initial configuration generated by the system.

At this point, the user asks the system to perform the design operations by clicking on the **DESIGN** button. This activates the inference mechanism, to process the applicable rules for this design situation. The expert system calculates the strain and the assembly forces resulting from this configuration and compares these values to the allowable values for the given material. The system then iteratively alters the geometry until a design meeting the specifications is reached.

An example design solution is illustrated in Figure 5.9. At this point, the user may want to make a material change or change the configuration and then ask the system to

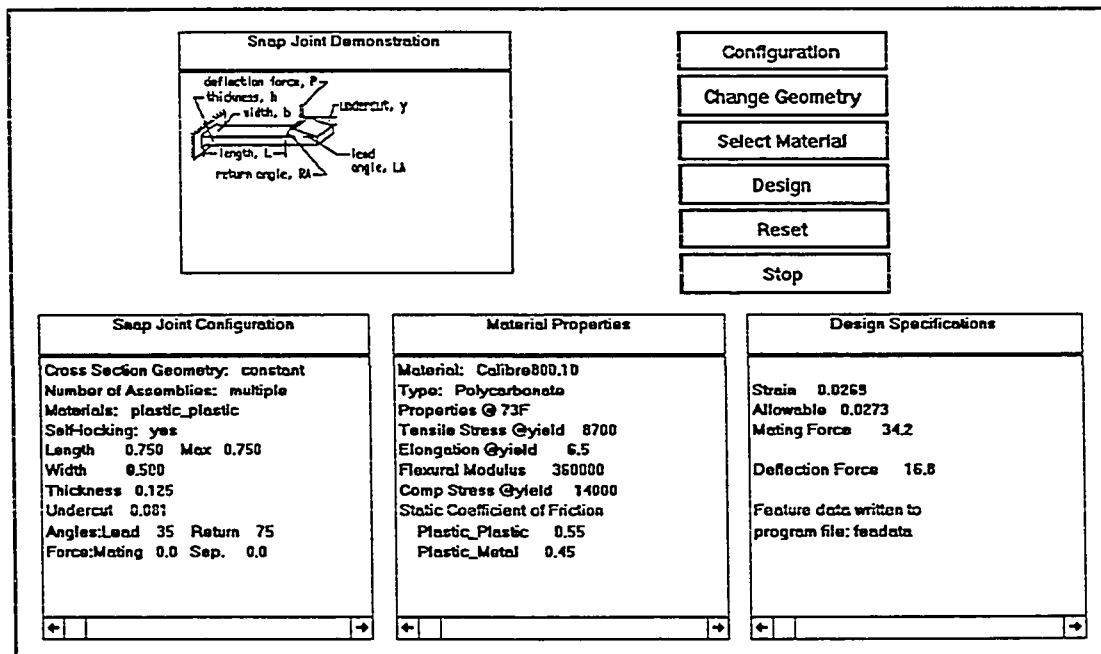


Figure 5.9. Design Results

produce a new design. This approach involves the user in the final design, providing an analysis tool to the expert designer while incorporating his experience. It also provides a design tool for the novice, producing a part which meets the design specifications.

Several examples of the design process are given in Appendix B. These examples were generated using the rule tracing feature of Kappa PC and illustrate the way the rules are applied in solving the problems with different design constraints. Different solutions, or "good" designs, are produced for the various configurations since the rules are applied in different sequences. Another tool for examining the reasoning process of the prototype system is the explanation facility, which *explains* how slot values are formulated in the design process. However, the explanation facility in Kappa PC was not activated for the prototype expert system. To include this facility in the expert system requires that explanations for each rule be entered in the comment field of the rule. Essentially the same information is available in the rule traces, but the explanation facility is more easily interpreted by a user.

INFERENCE STRATEGIES

The inference engine is responsible for searching the knowledge base and recommending a solution to the proposed problem. Specifically, the inference engine must decide where to start the inference process, which rules to fire when more than one is triggered (conflict resolution), and how to conduct the search, all in an effective and efficient manner.

Kappa PC provides a variety of methods to handle conflict resolution. Rule priorities can be assigned to control the reasoning path when more than one line of inference is possible. Rule sets can be established so only rules relevant to the task being performed are used, thus providing efficiency and modularity for the developer. And, Kappa PC provides four options for conflict resolution when more than one rule is eligible for firing: selective, breadth-first, depth-first, and best-first. The selective option is not exhaustive; only the first rule associated with the asserted facts is tested, thus only one successful path of reasoning is followed. Since the search is not exhaustive, it is more efficient. The remaining options are all exhaustive, finding all possible implications of the data that initiated the chaining process. The breadth-first option evaluates all the rules

associated with the asserted facts, before evaluating the next level of rules. The depth-first option evaluates one rule associated with the asserted facts and all its consequences, before evaluating other rules associated with the asserted facts. The best-first option combines features of the breadth-first and depth-first options, using rule priorities to select the “best” rule to fire next, i.e., it looks at all the rule possibilities and selects the one with the highest priority.

The two search strategies employed in Kappa PC are forward and backward reasoning. Forward reasoning, or data driven chaining, proceeds from premises (if part) toward conclusions (then part). It begins by declaring new facts and proceeds by matching known facts to the premises of rules. If all the premises of a rule are verified, the conclusions in the rules are asserted, generating new facts which can match the premises of more rules. Backward reasoning, or goal-driven chaining, tries to verify a fact, i.e., reach a goal, by finding rules which can prove the fact, in the conclusions, and then attempting to verify their premises. The premises in turn become new facts to be verified by other rules. The same rules can be used in both forward and backward reasoning.

A goal driven, or backward reasoning approach, is normally used in a design problem. In the cantilever snap joint design problem, *a good design*, i.e., a solution meeting the specifications, is defined by:

If ($\epsilon < \text{allowable strain}$ & $\text{assembly forces} < \text{allowable forces}$)
 then design is good.

Due to limitations in the early versions of the Kappa PC software, a backward reasoning strategy did not work. A simple solution to this obstacle was to use a forward reasoning strategy, incorporating a goal to terminate the reasoning.

The effects of the various conflict resolution options were also examined (see Appendix B). The order of rule assertion definitely affects the design solution and can result in a design which is over-corrected for the design constraints. This occurs for several reasons. A fairly large arbitrary increment was selected for the undercut, cantilever length, and angle modifications, which over-corrects the design solutions. Once a constraint is met, the application of additional design rules can result in further reductions,

or over-corrections. However, additional design rules can be added to the expert system to deal with both of these situations.

PROTOTYPE EVALUATION

Expert systems are generally validated by comparing the performance of the system with that of an expert. Formal measures, both quantitative and qualitative, have been developed to ascertain the effectiveness of an expert system. The prototype expert system was examined by an injection molding expert designer as well as injection molding software developers.

The initial validation of the prototype expert system verified that analytical results of the prototype match results from a commercial software package PD1¹. Results were compared by calculating the snap joint undercut deflection for a range of loading conditions and for the following materials:

- ABS DOW Magnum 3661
- ABS DOW Pulse 1725
- ABS GE B30-0001
- ABS Mobay Bayblend
- Modified PPO Noryl N1-190
- Polycarbonate DOW Calibre 800 -10
- Polyamide DuPont Zytel-101
- Acetal DuPont Natural

Calculations were also conducted independently to confirm the accuracy of the PD1 program results. Using the same material property data, no significant differences were noted in the results.

A second evaluation was performed independently by two injection molding software developers, Mike Craven and Gregg Nicholas². The evaluators were asked to address the following features of the prototype: correlation to known design solutions, procedures for data input, flexibility for altering configurations, design constraints, output usability and format, and ease-of-use. See Appendix E for a copy of the evaluation instrument. Comments from the evaluators were positive, with only a single suggestion to

¹ PD1, an IDES product for Injection Molding Part Design, is an on-line tutorial and interactive design tool for ribs, cross ribs and snap fits.

² Integrated Design Engineering Systems, Inc., PO Box 2131 Laramie, WY 82070.

post a busy message on the screen when the system is not expecting input from the user. This alerts the user to wait for the system to process the information.

An expert designer was then asked to review the performance of the prototype expert system. Mr. Robert Cramer³, a major contributor to the development of the PD1 program, confirmed that the performance of the system closely matched his expertise. Mr. Cramer did suggest a modification to the operation / user interface of the expert system to establish maximum dimensions for a projection which must fit in a constrained location. Changes to accommodate this modification were relatively easy to accomplish and attest to the usability of hybrid expert system shells for design problems. The changes were confined to a single class since similar functions are grouped in the object hierarchy.

The modifications have produced a more responsive expert system that more accurately reflects design concerns of a plastic part designer. The ability to easily adapt an expert system to user preferences produces a more useful design tool.

³ Associate Development Scientist, DOW Chemical Company, 433 Building, Midland MI.

CHAPTER 6

INTEGRATION OF EXTERNAL KNOWLEDGE SOURCES

In a complex design problem, an engineer typically enlists a variety of computer-aided engineering tools to assist in the design process. A solid modeling package is often used to develop a conceptual design and to provide powerful analysis tools; a commercial database of material properties can assist in selecting appropriate materials for the product. The prototype expert system resulting from this research can be added to the menu of CAE tools available to aid the product designer. The system approach to developing a plastic part, incorporating these tools interactively with a mechanical designer, is depicted in Figure 6.1.

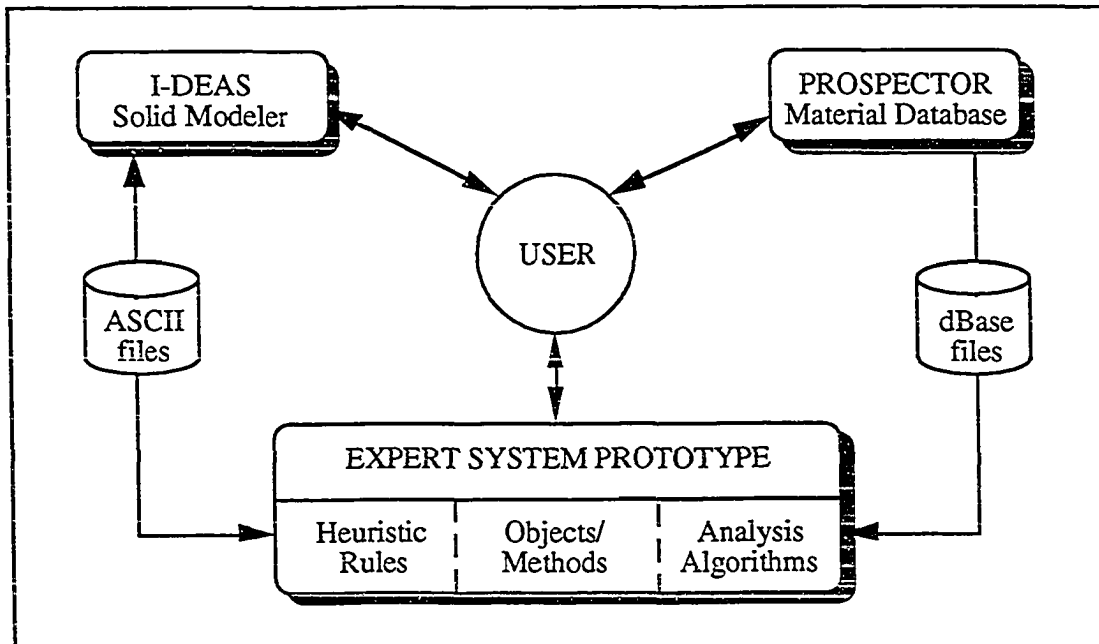


Figure 6.1. System Approach

The design scenario for this system approach involves a series of steps. The designer using a solid modeler, e.g., I-DEAS™ (Integrated Design Engineering Analysis System) available from the Structural Dynamics Research Corporation¹, inputs a conceptual design to meet a set of functional specifications for a plastic part. If a snap joint is required for the plastic part design, the designer invokes the prototype expert system to determine a set of geometric parameters meeting specifications for the required joint. At this point in the design process, a material selection for the plastic part will have typically been made; if not, the designer can rely on the expert system to incorporate knowledge contained in an external materials database, PROSPECTOR from IDES², to assist in material selection. Since the expert system is interactive, an experienced designer can influence the parameter generation, based on his/her individual experience. The expert system, using the knowledge sources interactively with the designer, determines the feasibility of the conceptual design, and modifies the design, iteratively, until acceptable design parameters are generated for the snap joint.

The expert system shell, Kappa PC, provides the capability for a developer to interface the expert system to external knowledge sources. Kappa PC interfaces to Lotus® 1-2-3® spreadsheets, dBASE® databases, and external software through built-in functions. Kappa PC also provides functions to read and write ASCII files which extends interface capabilities to most software.

Without modification to the solid modeler, real time information exchange between the prototype expert system and the modeler is not possible. However, the prototype can effectively communicate with the solid modeler through the Kappa PC functions for ASCII file exchanges, to share geometry information. In addition to the geometry database, the solid modeler can also provide analysis tools such as finite element modeling, vibration analysis, mold filling/cooling analysis and graphical numerical control machining. These capabilities have not been demonstrated for the prototype expert system, but can be utilized at any point in the design process since the geometry database resulting from the solid modeler is common to each of these options.

¹ Also available as CAEDS® (Computer Aided Engineering Design System) from IBM®

² Integrated Design Engineering Systems, Inc., PO Box 2131 Laramie, WY 82070.

The material properties for the cantilever snap joint design prototype are entered from a dBase compatible database, generated from the properties available in PROSPECTOR. This functionality is incorporated in the prototype system through the Kappa PC database functions, which map records from a database to objects in the expert system. This approach augments the expert system with material selection features in existing commercial software and takes advantage of the database capabilities for effectively, and efficiently, searching large material databases.

Since only a limited amount of code is required to perform the analysis for a snap joint, the prototype design embeds analysis capabilities in methods attached to the knowledge elements. In larger, more complex applications, the developer can incorporate external analysis programs by using Kappa PC functions to execute external programs and to pass arguments between the external programs and the expert system. Kappa PC also provides functions for reading and writing ASCII files to incorporate existing C code into methods attached to the knowledge elements.

EXTERNAL INTERFACE CAPABILITIES

The ASCII file transfer capabilities of Kappa PC provide a means for passing parameters to external programs. These capabilities include functions to open/close files, read characters or words, and write formatted text or internal Kappa PC files (classes, instances, rules, and functions). Using these functions, the prototype expert system is interfaced to a solid modeling package. External programs can also be executed from within Kappa PC, through a built-in function which passes up to three arguments to the external program.

Built-in functions interface Kappa PC to databases and spreadsheets, allowing Kappa PC to work directly with database or spreadsheet files. These functions open/close files, read/write selected data records or fields, and map Kappa PC slots to database fields. These functions also allow instances in the object-based hierarchy to be generated from the database information.

Kappa PC is available as a C library, to add intelligence capabilities to in-house programs; routines can be added to this library and called like any other Kappa PC functions. A run-time version is available to developers who want to incorporate Kappa PC into their software.

SOLID MODELING SOFTWARE

The engineer has a host of solid modeling software available to assist with engineering design tasks. I-DEAS, a package widely used by the mechanical engineering community, is an integrated package of software tools incorporating a concurrent engineering approach to mechanical design problems. I-DEAS consists of a number of "Families" of products including Solid Modeling, Engineering Analysis, System Dynamics, Test Data Analysis, Drafting, and Manufacturing. These integrated modules form a fully functional design tool for the engineer.

The Solid Modeling family includes an Object Modeling module which creates objects either from a menu of primitive solids (blocks, cylinders, cones, spheres) or from extruding or rotating a profile. These objects can be modified by various construction operations; complex objects are constructed through Boolean operations to join objects with each other or to cut them from one another. A geometry database is also maintained, which can be used for mass and inertia property calculation, interference studies, finite element modeling, manufacturing, and generating engineering drawings.

The constant cross-section snap joint was modeled in I-DEAS by generating a profile and extruding the profile to form a solid object (Figure 6.2). The snap joint was then created as a feature with the following parameters: length, width, thickness, undercut, return angle, lead angle -- parameters generated by the prototype expert system as a result of the DESIGN process. (See Appendix D for a listing of the I-DEAS commands to generate the SNAPJOINT feature.

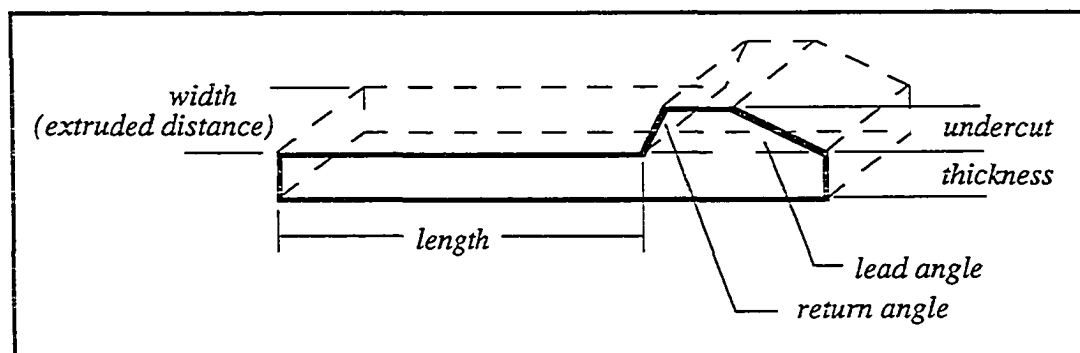


Figure 6.2. Cantilever Snap Joint Object

The prototype expert system writes these parameters to a file, in I-DEAS program file format, through Kappa PC built-in functions for ASCII file exchanges. The function `write_feadata` accomplishes this information exchange and is listed in Appendix A.

Once the parameters are determined, the designer uses the I-DEAS Model File module to generate the snap joint object at a specified location. The designer selects the Program File function, followed by the RUN command to execute the program file generated by the prototype system. The program file CONSTRUCTs an object from the SNAPJOINT feature, which is stored in the FEATURES Universal Library file.

The I-DEAS construction commands *snap* two coincident faces together and then use various positioning options to properly align the two faces. The designer is asked to specify the planar faces to be joined (one on the snap joint object and the other on the plastic part) and then to designate the exact location on the plastic part for the snap joint object. The snap joint object is thus attached to the nominal wall of the plastic part, at a user specified location.

DATABASE SOFTWARE

One problem facing a plastic designer is the best choice of plastic material for a particular application. Thousands of commercial grades of plastic materials are available on the U.S. market, making it nearly impossible for a designer to be familiar with the many blended and alloyed materials available. However, software tools exist to assist in selecting an appropriate material.

Plastic material properties are available in a commercial product, PROSPECTOR. PROSPECTOR uses the capabilities of a sophisticated data management system, FOXPRO[®], to provide query, display, report capabilities, and graphical visualization aids. The user interacts with PROSPECTOR to define a subset of materials that meet user specifications; data can be viewed in table or chart form to assist in defining the subset.

The PROSPECTOR database contains over 18,000 plastic materials, each having up to one hundred attributes representing general material characteristics, physical, mechanical, thermal, electrical, and flammability data. The information in the database is acquired directly from material manufacturers and suppliers and conforms to ASTM

specifications. A data sheet for a sample material, Calibre 800-4, is listed in Table 6.1. The user can query the database for any combination of the properties on the data sheet, which are important for a particular application. For example, a plastic part might need to be transparent and able to withstand high installation temperatures.

PROSPECTOR offers the user two major menu options -- Search and Display. Search allows the user to narrow the number of materials in the working database to only those of interest. The user selects a property to search, which can be either a text field or a numeric field. If a text field is chosen, the user selects the desired items from a list of all possible values. The numeric search shows a distribution of the material property to aid in picking a range of data values for the search.

The Display option provides the user two formats for viewing the searched material properties. The Data Sheet shows all data for a selected material, while the Data Table shows selected properties for the working database. The database can be sorted according to a particular property, or the Locate function can be used to find a material with a specific property value. The user can then use the Data Sheet to view successive materials and their properties. Within the Data Table, the user specifies the properties to display for each material and the order to display the properties.

PROSPECTOR was modified to produce an ASCII report, since PROSPECTOR's internal files are encrypted; the report is then used to generate a dBase compatible file which can be interfaced directly to the prototype expert system. A small amount of code development was necessary to generate a dBase compatible file from the PROSPECTOR output. The program also enters material property data (e.g., coefficients of friction) that are not available in the PROSPECTOR database. The edit program is listed in Appendix C.

The interface capabilities of Kappa PC were then used to import the material properties from the database into the prototype expert system. The prototype expert system creates material instances from the external database and enters the database fields into the object slots of each material. The object-based hierarchy provides the capability of automatically generating the instances from the material database and updates the user interface to reflect the current material database. The function that accomplishes this interface, loaddb, is listed in Appendix A.

Table 6.1. PROSPECTOR Data Sheet

Trade Name : Calibre 800-4
 Manufacturer: Dow Chemical U.S.A.
 Generic Name: Polycarbonate

Property	Value	Units
-General	:	:
Agency Ratings	:	:
Appearance	:	:
Features	: Ignition Resistant	:
	: Mold Release, Good	:
	: UV Resistant	:
Filler Percent By Volume	:	%
Filler Percent By Weight	:	%
Filler/Additive	:	:
Processing Methods	: Coextrusion	:
	: Blow Molding, Extrusion	:
	: Extrusion, Profile	:
	: Extrusion, Sheet	:
	: Blow Molding, Injection	:
	: Thermoforming	:
Recycled	: Yes	:
Uses	: Appliances	:
	: Business Equipment	:
	: Electrical Parts	:
	: Lawn and Garden Equipment	:
	: Communication Application	:
-Physical	:	:
Cont. Service Temperature	:	°F
Glass Transition Temp	:	°F
Linear Mold Shrink	: 6.000	mils/in
Melt Flow	: 4.00	g/10 min
Melt Flow Condition	: O - 300°C/1.2 kg	:
Melt Pt	:	°F
Specific Gravity	: 1.2095	-
Water Absorption 24 hrs.	: 0.150	%
Water Absorption @ Equil	: 0.320	%
-Mechanical	:	:
Compressive Modulus	:	psi
Compressive Strength	:	psi
Elongation @ Break	: 100.0	%
Elongation @ Yield	: 6.5	%
Flexural Modulus	: 360000	psi
Flexural Strength @ Yld	: 14000	psi
Gardner Impact	:	in-lb
Hardness Value	: Rockwell Hardness M-Scale 59	:
Notched Izod Impact	: 12.00 @ 73°F, 0.12500"	ft-lb/in
Shear Modulus	:	psi
Shear Strength	:	psi
Tensile Impact Strength	: 260.00 @ 73°F	ft-lb/in ²
Tensile Modulus	: 330000	psi
Tensile Strength @ Brk	: 8500	psi
Tensile Strength @ Yld	: 8700	psi
Unnotched Izod Impact	: No Break @ 73°F, 0.12500"	ft-lb/in
-Optical	:	:
Haze	:	%

Refractive Index	:	-
Transmittance	:	%
-Thermal	:	:
Brittle Temp	:	°F
Coef Linear Thermal Exp	: 3.80000	: in ⁻⁵ /(in-°F)
Deflection Temp @ 264 psi	: 266	: °F
Deflection Temp @ 66 psi	:	: °F
Specific Heat	:	: BTU/lb °F
Thermal Conductivity	:	: BTUin/hrft ² °F
Vicat Softening Point	: 310	: °F
-Flammability	:	:
Limiting Oxygen Index	: 40.00	: %
-Electrical	:	:
Dielectric Constant	: 3.00	:
Dielectric Strength	: 405.00	: V/10-3 in
Dissipation Factor	: 0.001000	:
Surface Resistivity	:	: ohm
Volume Resistivity	:	: ohm cm
-Underwriter Labs	:	:
Arc Resistance	:	: seconds
Comparative Tracking Ind	:	:
High Volt Arc Res to Ign	:	:
High Volt Arc Track Rate	:	:
High-Ampere Arc Ignition	:	: # of arcs
Hot Wire Ignition	:	: seconds
Rel Temp Indx Mech w/Imp	:	:
Rel Temp Indx Mech w/oImp	:	:
Relative Track Ind Elect	:	:
UL 94 Rating	: V-0	:
-Injection Molding	:	:
Back Pressure	:	: psi
Drying Temp	:	: °F
Drying Time	:	: hours
Freeze Temp	:	: °F
Front Cylinder Temp	:	: °F
Injection Pressure	:	: psi
Injection Time	:	: seconds
Middle Cylinder Temp	:	: °F
Minimum Wall Thickness	:	: in
Mold Temp	:	: °F
No Flow Temp	:	: °F
Nozzle Temp	:	: °F
Processing Temp	:	: °F
Rear Cylinder Temp	:	: °F
Screw RPM	:	: rpm
-Thermoset	:	:
Apparent Density	:	: lb/ft ³
Bulk Factor	:	:
Mix Ratio By Volume	:	:
Mix Ratio By Weight	:	:
Mixed Viscosity	:	: cps
Stoichiometry	:	: %
-Elastomer	:	:
Compression Set	:	: %
Tens Modulus, 100% Elong	:	: psi
Tens Modulus, 200% Elong	:	: psi
Tens Modulus, 300% Elong	:	: psi
Tens Modulus, 50% Elong	:	: psi

HARDWARE / SOFTWARE ENVIRONMENT

A major challenge facing computer-aided engineering software developers is interfacing a variety of tools that exist on an even wider range of hardware. For the prototype expert system, three distinct software implementations had to be considered: Kappa PC is a windows based PC product, PROSPECTOR is a DOS based PC product, and I-DEAS is a UNIX based workstation product. Specific requirements for each of these software products are listed below.

The expert system shell Kappa PC is a general purpose C-based application development and delivery environment for PCs and requires the following system components:

- 286 or higher processor
- 640 KB RAM
- Hercules™ Graphics Card, EGA®, or VGA® Monitor
- 2 MB disk space
- MS-DOS 3.0
- Microsoft Windows 3.0

The PROSPECTOR commercial database, available from IDEAS, requires the following:

- IBM or compatible PC
- 4MB RAM
- 15 MB disk space
- Microsoft Windows 3.1 (Enhanced mode)

I-DEAS™, installed on a DECstation ULTRIX configuration requires:

- ULTRIX 4.2A
- DECWindows 4.2A
- PHIGS 2.3A and PEX 5.0 Graphics Libraries
- Fortran 77 v3.1
- 16 MB Memory
- 75 MB disk space (minimum / options additionally require up to 450MB)
- 150 MB swap space

The hardware / software requirements of these software packages highly restrict the platforms that can support the integrated approach of the expert system application. Hence, the prototype expert system was implemented on an IBM DX266 (Model 77) OS/2 v2.11.

The capabilities of the IBM system provide a seamless tool for the implementation of the integrated prototype system.

SUMMARY

The integration of CAE software tools for plastic part design significantly simplifies the design process. Incorporating existing CAE applications such as computer-aided design and solid modeling, material databases, and analysis software not only extends the utility of the prototype expert system, but provides the designer with a single, easy-to-learn and easy-to-use tool for generating the design for plastic parts. The utility of the prototype is further enhanced by interactively involving the designer in the design process.

CHAPTER 7

RESULTS AND CONCLUSIONS

Expert systems have been applied to a variety of engineering problems. Early successes have been recorded in derivation problems: monitoring manufacturing processes, diagnosing and predicting failures, controlling chemical processing, and advising FEM users. These systems are currently being implemented by the *users* of the systems, often with fairly easy-to-use PC expert system shells with robust development tools. Expert systems have also been developed in the last decade for formation problems in planning and design. However, most of these implementations have used the heuristic programming languages LISP and PROLOG or complex programming environments like KEE and ART.

This research has resulted in a prototype expert system implementation for an engineering design application: the design of a feature for an injection molded plastic part. The prototype system was implemented using an expert system shell and has been evaluated by experts in both injection molding part design and software development. The prototype was modified to reflect these evaluations; the resulting expert system performs closely to an expert designer and is relatively simple for a designer to use.

The prototype expert system addresses a fairly narrow domain. To be an effective design tool, the prototype must be extended from basic feature design to the design of complex parts and their corresponding molds and to other manufacturing processes. With the object oriented rule-based representation scheme, additional features and processes can be easily incorporated. However, the value of this prototype is in establishing the guidelines, or templates, for developing expert system tools for design processes.

Previous research in expert system applications for engineering design has not addressed the use of high level development tools, i.e., expert system shells. One exception is the development of an automated fixture design system (MEFDES by kumar,

Nee, and Prombanpong 1992) which is a planning application that integrates an expert system, developed with the Nexpert Object expert system shell, with the ME30 CAD system. This research, on the other hand, addresses an engineering design application and integrates the expert system with a solid modeling system as well as external databases and interfaces to external software.

RESULTS

The goal of this research was to develop a standard approach to implementing expert systems for engineering design applications. To pursue this goal, several fundamental tasks (or objectives) for developing an expert system for an engineering design application were explored and formalized:

- investigate the use of expert systems shells for design problems
- categorize the knowledge required to solve design problems
- formulate representations for the knowledge
- integrate the expert system with external databases and solid modeling software
- develop interactive capabilities, as well as graphical interfaces.

The accomplishments for each of these tasks, along with recommendations pertinent to expert system implementations for engineering design applications, are discussed in each of the following sections.

EXPERT SYSTEM SHELLS. The prototype application has demonstrated the feasibility of using shells to develop expert systems for formation problems. The successful implementation has identified features that are essential in an expert system development tool for an engineering design application: a variety of knowledge representations and capabilities to integrate external software; to develop an interactive, graphical user interface; and for explanation.

Kappa PC has proven to be a good development tool for design problems. The cantilever snap joint prototype development has demonstrated that knowledge characteristic to a design problem can be effectively represented in an object-oriented, rule-based system. The objects, and associated methods, are useful in representing materials and specifications data, as well as the engineering models and scientific principles used to analyze the design. The heuristics, which are an integral part of any design problem, are suitably represented

by rules. Kappa PC provides capabilities to integrate the knowledge base with existing databases and software, including solid modeling systems. Kappa PC also provides an extensive development environment which facilitates rapid prototype implementation and interactive capabilities to integrate the user's expertise with the system.

The prototype development has also demonstrated the ease of using an expert system shell for design applications. A user familiar with Kappa PC can develop a simple system, complete with a viable user interface, in a matter of hours. For the user who is also the design engineer, familiar with the design heuristics of the problem, knowledge acquisition for the system is a much simpler task. The design engineer is able to provide many of the rules from his/her own experience. Since many programming tasks have been incorporated in the development tools within Kappa PC, a typical engineer with limited programming skills will be able to use Kappa PC effectively. Thus Kappa PC is a powerful tool for the design engineer.

The demonstration system has illustrated the utility of expert system shells for engineering design problems. Expert system shells deal effectively with the complexity of engineering design, and they provide a designer, familiar with the design heuristics of a problem, with an easy-to-use tool that facilitates rapid development of an expert system.

CATEGORIZE KNOWLEDGE. A variety of knowledge typically found in design problems was identified: hierarchy of configurations and components, geometric information and constraints, material properties, specifications, analysis procedures based on governing equations, and heuristic design rules. The prototype system has specifically addressed each of the knowledge categorizations identified as key elements of mechanical engineering design.

KNOWLEDGE REPRESENTATION.

A rule-based system is a good representation for the many guidelines and rules of thumb that are invoked in a design application. Generally, these rules take the form of "If-Then" procedures. However, the complexity of design does not lend itself to a procedural collection of these rules. Often, the expert designer cannot develop this set of procedures for a particular design problem but can formulate various rules, as premises leading to conclusions, that he uses to arrive at a proposed solution.

An object-oriented environment is an excellent paradigm for representing the knowledge in engineering design problems. A mechanical design is often a hierarchy of components with specific attributes, which can be modeled by a network of objects and their slot values. The relationships between the components are similar to the relationships in a network and can be modeled with methods attached to the objects. These relationships, e.g., algorithmic analysis procedures, can also be executed as external programs. The remaining knowledge -- materials and properties, geometric configurations and constraints, and specifications -- is also amenable to representation as objects.

INTEGRATION. The prototype system integrates two commercial products: the PROSPECTOR external materials database and Structural Dynamics Research Corporation's I-DEAS™ (Integrated Design Engineering Analysis System). This integration expands the capability and flexibility of the expert system. Since engineering design often uses databases, either large materials databases or geometry databases, and involves any number of simple to complex analysis software packages, the expert system design aid must have the functionality to incorporate a variety of external knowledge sources in engineering design applications. The integration can often be effectively accomplished through ASCII file exchanges.

INTERACTIVE. The prototype system exhibits an interactive user interface, which is instrumental in a user's acceptance of an expert system. A well developed graphical interface affects how easy a system is to learn and contributes to the ease of use, and thus to the system's eventual acceptance. The user interface must address the novice designer as well as the expert designer, allowing the user to participate in the design process.

The rich development environment of the expert system shell chosen for the prototype provides the developer with the means to generate a sophisticated user interface, incorporating graphical and interactive tools. The toolkit available in Kappa PC greatly simplifies the user interface development resulting in an effective, interactive interface.

CONCLUSIONS

In design problems, a variety of knowledge is available for the design solution. Integrating all these knowledge sources into the expert system enhances the problem solving capabilities of the system. In the prototype expert system, the material information

was loaded from a large materials database, demonstrating the interface capability with external databases. Realistic design problems need to incorporate an external materials database containing a wide range of materials.

The algorithmic procedures relevant to designing a cantilever snap fit were implemented in the methods attached to the objects. However, the algorithmic procedures contained in the analysis software products available for engineering design are numerous and lengthy. A more effective approach for incorporating the algorithms in an expert system is to use the features of the shell to execute external procedures.

A versatile interface accommodates a range of users -- from the novice who uses the expert system as a tutor, to the expert who uses the system as a design aid or to validate a proposed design. An expert system is most powerful when it involves an expert user in the design process. The interactive capabilities of the prototype system incorporate the user as an additional knowledge source, extending the system from a tutorial package to a truly important design aid. The rule trace feature in Kappa PC provides essentially the same information as an explanation facility and was useful in developing the system; however, an explanation facility needs to be developed to extend the use of the prototype to novice users, who need a tutorial approach to the design application.

Other expert system tools are emerging as viable development tools for implementing sophisticated expert systems. Many of the vendors offer their products in the Windows environment and have incorporated objects and message passing capabilities and graphical tools for developing and debugging applications. Some of the major tools in addition to Kappa PC / ProKappa, are Level5 Object, Nexpert Object, and TIRS and ESE (from IBM). Vendors are working to offer their products on multiple platforms providing the expert system developer more versatility in distributing expert system applications. This new generation of expert system tools provides easy-to-learn and easy-to use software for expert system implementations.

Developing the expert system application also demonstrated a need to address the same concerns that arise in a large, comprehensive software project, i.e., modularity, maintainability, scalability, and validation / verification. In developing the knowledge representations, efforts need to be made to provide modularity for the system. Modularity

makes the system development more efficient and allows future extensions to be made with minimal disruption to the system. Modularity influences the maintainability of the system, but structured programming languages have had a more pronounced impact on the maintainability of comprehensive software projects. High level tools, like expert system shells, assist the developer in structuring the system and provide an easily understood tool for maintenance tasks and for system development documentation. Object-oriented environments are inherently structured, and thus produce more maintainable systems. The value of an object-based approach was demonstrated when modifications were easily made to the prototype system. The scalability of the application needs to be addressed during the development of a prototype system since the prototype may not be applicable to larger, more complex problems. Finally the developer needs to formulate definite plans to verify / validate the system. Generally a good test for the expert system is to compare its performance to an expert, who has not been involved with the expert system development.

A hybrid expert system shell, based on an object-oriented knowledge representation coupled with production rules, provides a useful tool for the design engineer. The design engineer, familiar with separating a problem into components, can easily formulate the components as objects; he can also easily implement his design knowledge with "If-Then" rules. However, knowledge acquisition may still be a problem, even for the design expert. The expert must be able to organize his design procedures and express the procedures in some representation, typically in production rules. This is not a trivial task, and the expert is often reluctant, or unable to carry out this step.

FUTURE RESEARCH

The prototype implementation characterized the various forms of knowledge used in design processes and identified corresponding knowledge representations. This work lays the foundation for expert system implementations for more complex problems, involving many interrelated design components.

The prototype demonstrated the importance of integrating the knowledge base with existing databases and analysis software. Thousands of materials exist for manufacturing plastic parts. Database features are often employed in selecting an appropriate subset of materials for a particular application, but the format of some existing databases is not

directly compatible with the Kappa PC database interface. Standardization of database representations will promote the integration of existing databases with expert system knowledge bases. Incorporating the geometric and features databases generated by computer-aided design and solid modeling software into an expert system alleviates the user from providing this information to the system and ensures consistency of the data. In addition, the sophisticated design aspects of the CAE software can be exploited in developing the conceptual design.

More complex engineering problems require more advanced analysis tools, like finite element modeling. A widely used, integrated software system from the Structural Dynamics Research Corporation (SDRC), I-DEAS™, is used for conceptual design, analysis, detailed design and drafting, computer-aided testing and manufacturing of mechanical products. I-DEAS not only offers FEM capabilities, but also a solid modeling database, a material data system, dynamic analysis, numerical control machining and plastics analysis. Coupling an analysis tool like I-DEAS with an expert system produces an extremely valuable design aid and can be accomplished in one of three ways: embedding the expert system in the analysis tool, embedding the analysis tool in the expert system, or executing each system independently sharing information between the two applications through a blackboard architecture. For sophisticated systems like I-DEAS, the first approach provides the most flexibility to the designer; the full capabilities of the analysis tool are available for the design problem, while the expert system guides the design solution. The second approach is easier to implement and is appropriate for simpler analysis systems. Executing the two systems independently requires a great deal of communication between the systems, which may not be easily developed.

Extensive user interface features impact the acceptance and viability of a software product. Kappa PC provides a good set of graphical tools for developing an interactive, responsive user interface. More development in the user interface should focus on explanation and knowledge acquisition capabilities. Explanations of the reasoning used in the design process extend the role of an expert system to that of a computer-assisted instructor. A knowledge acquisition facility helps ensure that the expert system will continue to evolve and will continue to be used. Involving the expert user, as a knowledge

source, and adding this knowledge to the database extends the usefulness of the expert system design.

The modularity of an object-based knowledge representation readily permits extensions to the prototype for the complete hierarchy of complex plastic parts: the nominal wall, projections off the nominal wall, and depressions into the nominal wall. These three classifications can be implemented as distinct classes; the snap joint class is a subclass of projections. Other features found in injection molding applications -- annular snap-fits, ribs, bosses, holes -- are subclasses of the projection and depression classes. Complex features such as threads, springs, gears, and bearings are combinations of the basic elements. Manufacturing processes can be incorporated in the prototype through the use of rule sets. The rules used for injection molding processes can be grouped in a set; and sets can be constructed for other manufacturing processes.

An experienced designer routinely considers various factors for optimizing a design, e.g., weight, volume, and cost, and adjusts his designs accordingly. Rules can be added to the prototype expert system to incorporate optimization techniques in evaluating the design. I-DEAS includes an optimization task within the FEM module; existing software routines for optimization can also be executed using the shell interface capabilities to external programs.

A fully functioning expert system incorporating these complex features would provide a powerful design aid for the mechanical designer. A design engineer, using the SDRC solid modeling system, could develop a conceptual design; the resulting solid modeling databases would serve as knowledge sources for the expert system. Other knowledge sources would be constructed from the materials data system, heuristic rules for part design, and analysis software, such as finite element modeling. The expert system, using these knowledge sources interactively with the designer, would determine the feasibility of the conceptual design, and modify the design, iteratively, until an optimum design is formulated. The expert system would thus facilitate cooperative problem solving among multiple *experts*.

APPENDIX A

LISTINGS: CLASSES (INCLUDING METHODS), INSTANCES, RULES, FUNCTIONS

```

/*****
/**      ALL CLASSES ARE SAVED BELOW      **/
*****/

      /*****
      **** CLASS: material
      *****/
MakeClass( material, Root );

/***** METHOD: select *****/
MakeMethod( material, select, [],
{
  GetInstanceList( material, Global:matlist );
  AppendToList( Global:matlist, "NEW MATERIAL DATABASE" );
  cantilever:material_type = PostMenu( "Select a material", Global:matlist );
  If ( Global:feature:material_type #="NEW MATERIAL DATABASE" )
    Then loaddb( );
} );

/***** METHOD: output_mat *****/
MakeMethod( material, output_mat, [],
{
  ClearTranscriptImage( output_mat );
  DisplayText( output_mat, FormatValue( "Material: %s", Global:feature:material_type ));
  DisplayText( output_mat, FormatValue( "\nType: %s\nProperties @ 73F",
    Global:feature:material_type:type ) );
  DisplayText( output_mat, FormatValue(
    "\nTensile Stress @yield%8.0f\nElongation @yield%10.1f",
    Global:feature:material_type:tensile_stress, Global:feature:material_type:elongation) );
  DisplayText( output_mat, FormatValue( "
    \nFlexural Modulus%12.0f\nComp Stress @yield%10.0f",
    Global:feature:material_type:flexural_modulus,
    Global:feature:material_type:compressive_stress ) );
  DisplayText( output_mat, FormatValue( "\nStatic Coefficient of Friction\n
    Plastic_Plastic%10.2f\n Plastic_Metal%10.2f",
    Global:feature:material_type:mu_plastic_plastic,
    Global:feature:material_type:mu_plastic_metal ) );
} );
MakeSlot( material:type );
MakeSlot( material:flexural_modulus );

```

```

SetSlotOption( material:flexural_modulus, VALUE_TYPE, NUMBER );
MakeSlot( material:elongation );
SetSlotComment( material:elongation, Percentage );
SetSlotOption( material:elongation, VALUE_TYPE, NUMBER );
SetSlotOption( material:elongation, MINIMUM_VALUE, 0 );
SetSlotOption( material:elongation, MAXIMUM_VALUE, 100 );
MakeSlot( material:mu_plastic_plastic );
SetSlotOption( material:mu_plastic_plastic, VALUE_TYPE, NUMBER );
MakeSlot( material:mu_plastic_metal );
SetSlotOption( material:mu_plastic_metal, VALUE_TYPE, NUMBER );
MakeSlot( material:tensile_stress );
SetSlotOption( material:tensile_stress, VALUE_TYPE, NUMBER );
MakeSlot( material:compressive_stress );
SetSlotOption( material:compressive_stress, VALUE_TYPE, NUMBER );
MakeSlot( material:flag );
SetSlotOption( material:flag, ALLOWABLE_VALUES, yes, no );
material:flag = NULL;

```

```

/*****
**** CLASS: design
*****/

```

```

MakeClass( design, Root );
MakeSlot( design:area );
design:area = 0.062500;
MakeSlot( design:allowstrain );
design:allowstrain = 0.027300;
MakeSlot( design:deforce );
design:deforce = 14.391043;
MakeSlot( design:factor );
design:factor = 1.0;
MakeSlot( design:mateforce );
design:mateforce = 22.796429;
MakeSlot( design:sepforce );
design:sepforce = 11360179;
MakeSlot( design:strain );
design:strain = 0.021492;
MakeSlot( design:tensile_stress );
design:tensile_stress = 181770601.120000;
MakeSlot( design:compressive_stress );
design:compressive_stress = 8101.862864;
MakeSlot( design:mu );
design:mu = .55;
MakeSlot( design:criteria );
design:criteria = good;
MakeSlot( design:length );
design:length = 0.70;

```

```

/*****
**** CLASS: feature
*****/

```

```

MakeClass( feature, Root );

```

```

/*****
**** CLASS: nominal_wall
*****/
MakeClass( nominal_wall, feature );
/*****
**** CLASS: projections
*****/
MakeClass( projections, feature );

/*****
**** CLASS: snap_joint
*****/
MakeClass( snap_joint, projections );

/*****
**** CLASS: cantilever
*****/
MakeClass( cantilever, snap_joint );

/***** METHOD: change_geometry *****/
MakeMethod( cantilever, change_geometry, [],
{
  PostInputForm( "Geometry data", cantilever:length, "Enter length",
    cantilever:width,"Enter width", cantilever:thickness, "Enter thickness",
    cantilever:undercut, "Enter undercut", cantilever:lead_angle, "Enter lead angle",
    cantilever:return_angle,"Enter return angle" );
} );

/***** METHOD: init *****/
MakeMethod( cantilever, init, [],
{
  PostInputForm( "Initial design configuration",
    cantilever:geometry,"Select type of geometry",
    cantilever:flex, "Select number of assemblies",
    cantilever:material_mating, "Select component types",
    cantilever:NW, "Enter nominal wall thickness",
    cantilever:lead_angle, "Enter lead angle",
    cantilever:return_angle,"Enter return angle",
    cantilever:self_locking, "Is snap self locking?",
    cantilever:maxlength, "Enter maximum length of cantilever",
    cantilever:mateforce, "Enter maximum mating force",
    cantilever:sepforce, "Enter maximum separating force" );
  cantilever:thickness = .5 * cantilever:NW;
  cantilever:length = 5.4 * cantilever:thickness;
  If Null?( cantilever:maxlength )
  Then ( cantilever:maxlength = 3 * cantilever:NW )
  Else If ( cantilever:maxlength > 3 * cantilever:NW )
  Then {
    cantilever:maxlength = 3 * cantilever:NW;
    PostMessage( "Resetting Max Length to 3*Nominal Wall" );
  };
} );

```

```

If ( cantilever:length > cantilever:maxlength )
    Then cantilever:length = cantilever:maxlength;
cantilever:width = 4 * cantilever:thickness;
cantilever:undercut = .176 * cantilever:length;
Global:angle = 90 - Atan( design:mu ) * 180 / 3.14159;
If Null?( cantilever:sepforce )
    Then cantilever:sepforce = 0;
If Null?( cantilever:mateforce )
    Then cantilever:mateforce = 0;
});

/***** METHOD: output_config *****/
MakeMethod( cantilever, output_config, [],
{
    ClearTranscriptImage( output_config );
    DisplayText( output_config, FormatValue( "Cross Section Geometry: %s \nNumber of
Assemblies: %s\nMaterials: %s \nSelf-locking: %s ",
    cantilever:geometry, cantilever:flex,cantilever:material_mating,cantilever:self_locking));
    DisplayText( output_config, FormatValue(
    "\nLength %11.3f   Max%8.3f\nWidth %13.3f\nThickness %7.3f\nUndercut %8.3f",
    cantilever:length, cantilever:maxlength, cantilever:width, cantilever:thickness,
    cantilever:undercut ) );
    DisplayText( output_config, FormatValue( "\nAngles:Lead%6.0f   Return%6.0f",
    cantilever:lead_angle, cantilever:return_angle ) );
    DisplayText( output_config, FormatValue( "\nForce:Mating%6.1f   Sep.%8.1f",
    cantilever:mateforce, cantilever:sepforce ) );
    If ( cantilever:geometry #= constant )
        Then ( Bitmap1:FileName = snconst.bmp )
        Else If ( cantilever:geometry #= hdecreasing )
            Then ( Bitmap1:FileName = snthk.bmp )
            Else Bitmap1:FileName = snwidth.bmp;
    DrawImage( Bitmap1 );
});

/***** METHOD: calculate *****/
MakeMethod( cantilever, calculate, [],
{
    If ( cantilever:geometry #= constant )
        Then ( design:factor = .67 )
        Else If ( cantilever:geometry #= bdecreasing )
            Then ( design:factor = 1.09 )
            Else design:factor = .86;
    design:strain = cantilever:undercut * cantilever:thickness / design:factor /
    cantilever:length ^ 2;
    If ( cantilever:flex #= single )
        Then ( design:factor = .7 )
        Else design:factor = .42;
    design:allowstrain = design:factor * cantilever:material_type:elongation / 100;
    design:deforce = cantilever:width * cantilever:thickness ^ 2 / 6 *
    cantilever:material_type:flexural_modulus * design:strain / cantilever:length;
    If ( cantilever:material_mating #= plastic_plastic )
        Then ( design:mu = cantilever:material_type:mu_plastic_plastic )

```

```

Else design:mu = cantilever:material_type:mu_plastic_metal;
design:mateforce = design:deforce * ( design:mu + Tan( cantilever:lead_angle*
3.14159 / 180 ) ) / ( 1 - design:mu * Tan( cantilever:lead_angle * 3.14159 / 180 ) );
design:seforce = design:deforce * ( design:mu + Tan( cantilever:return_angle *
3.1415 / 180 ) ) / ( 1 - design:mu * Tan( cantilever:return_angle * 3.14159 / 180 ) );
If ( cantilever:geometry # = constant )
Then ( design:factor = 1.0 )
Else If ( cantilever:geometry # = bdecreasing )
Then ( design:factor = .5 )
Else design:factor = .25;
design:area = design:factor * cantilever:width * cantilever:thickness;
design:tensile_stress = design:seforce / design:area +
cantilever:material_type:flexural_modulus * design:strain;
design:compressive_stress = cantilever:material_type:flexural_modulus * design:strain
- design:mateforce / design:area;
design:length = cantilever:length;
} );

/***** METHOD: output_soln *****/
MakeMethod( cantilever, output_soln, [],
{
ClearTranscriptImage( output_soln );
DisplayText( output_soln, FormatValue( "\nStrain %10.4f\nAllowable %8.4f",
design:strain, design:allowstrain ) );
DisplayText( output_soln, FormatValue( "\n\nMating Force %10.1f\n",
design:mateforce));
If ( cantilever:self_locking # = no )
Then DisplayText( output_soln, FormatValue( "\nSeparating Force%10.1f\n",
design:seforce ) );
DisplayText( output_soln, FormatValue( "\nDeflection Force%10.1f", design:deforce ) );
DisplayText( output_soln, FormatValue( "\n\nFeature data written to \nprogram file:
feadata" ) );
} );

MakeSlot( cantilever:flex );
SetSlotOption( cantilever:flex, ALLOWABLE_VALUES, single, multiple );
cantilever:flex = multiple;
SetSlotOption( cantilever:flex, PROMPT, "Select number of assemblies" );
MakeSlot( cantilever:geometry );
SetSlotOption( cantilever:geometry, ALLOWABLE_VALUES, constant, hdecreasing,
bdecreasing );
cantilever:geometry = hdecreasing;
SetSlotOption( cantilever:geometry, PROMPT, "Select type of geometry" );
MakeSlot( cantilever:lead_angle );
SetSlotOption( cantilever:lead_angle, VALUE_TYPE, NUMBER );
cantilever:lead_angle = 35;
MakeSlot( cantilever:length );
SetSlotOption( cantilever:length, VALUE_TYPE, NUMBER );
cantilever:length = 0.675000;
MakeSlot( cantilever:material_mating );
SetSlotOption( cantilever:material_mating, ALLOWABLE_VALUES, plastic_plastic,
plastic_metal );

```



```

cantilever:material_mating = plastic_plastic;
SetSlotOption( cantilever:material_mating, PROMPT, "Select component types" );
MakeSlot( cantilever:material_type );
SetSlotOption( cantilever:material_type, VALUE_TYPE, OBJECT );
SetSlotOption( cantilever:material_type, ALLOWABLE_CLASSES, material );
cantilever:material_type = Calibre.800.10;
MakeSlot( cantilever:mateforce );
SetSlotOption( cantilever:mateforce, VALUE_TYPE, NUMBER );
cantilever:mateforce = 0;
MakeSlot( cantilever:thickness );
SetSlotOption( cantilever:thickness, VALUE_TYPE, NUMBER );
cantilever:thickness = 0.125000;
MakeSlot( cantilever:width );
SetSlotOption( cantilever:width, VALUE_TYPE, NUMBER );
cantilever:width = 0.500000;
MakeSlot( cantilever:return_angle );
SetSlotOption( cantilever:return_angle, VALUE_TYPE, NUMBER );
cantilever:return_angle = 60;
MakeSlot( cantilever:undercut );
SetSlotOption( cantilever:undercut, VALUE_TYPE, NUMBER );
cantilever:undercut = 0.056448;
MakeSlot( cantilever:NW );
SetSlotOption( cantilever:NW, VALUE_TYPE, NUMBER );
cantilever:NW = .25;
MakeSlot( cantilever:self_locking );
SetSlotOption( cantilever:self_locking, ALLOWABLE_VALUES, yes, no );
cantilever:self_locking = yes;
MakeSlot( cantilever:sepforce );
SetSlotOption( cantilever:sepforce, VALUE_TYPE, NUMBER );
cantilever:sepforce = 0;
MakeSlot( cantilever:maxlength );
cantilever:maxlength = 0.750000;

    /*****
    **** CLASS: torsional
    *****/
MakeClass( torsional, snap_joint );

    /*****
    **** CLASS: annular
    *****/
MakeClass( annular, snap_joint );

    /*****
    **** CLASS: ribs
    *****/
MakeClass( ribs, projections );

    /*****
    **** CLASS: depressions
    *****/
MakeClass( depressions, feature );

```

```

/*****
/**      ALL INSTANCES ARE SAVED BELOW      **/
*****/
MakeSlot( Global:matlist );
SetSlotOption( Global:matlist, MULTIPLE );
SetValue( Global:matlist, Bayblend.FR.1439, Calibre.800.10, Magnum.3661,
Pulse.1725, "NEW MATERIAL DATABASE" );
MakeSlot( Global:angle );
SetSlotOption( Global:angle, VALUE_TYPE, NUMBER );
Global:angle = 61.189194;
MakeSlot( Global:fieldnames );
SetSlotOption( Global:fieldnames, MULTIPLE );
SetValue( Global:fieldnames, TYPE, MODULUS, ELONGATION, MU_PP, MU_PM,
TSTRESS, CSTRESS );
MakeSlot( Global:slotnames );
SetSlotOption( Global:slotnames, MULTIPLE );
SetValue( Global:slotnames, type, flexural_modulus, elongation, mu_plastic_plastic,
mu_plastic_metal, tensile_stress, compressive_stress, flag );
MakeSlot( Global:instance );
Global:instance = Zytel.101..Dry.;
MakeSlot( Global:num );
Global:num = 6;
MakeSlot( Global:feature );
Global:feature = cantilever;
MakeSlot( Global:xscreen );
Global:xscreen = 1024;
MakeSlot( Global:yscreen );
Global:yscreen = 768;
MakeSlot( Global:RuleSet );
SetSlotOption( Global:RuleSet, MULTIPLE );
SetValue( Global:RuleSet, ckstrain, cktensile_stress, ckcompressive_stress, ckdesign,
cktensile, ckcompressive, smallest_lead_angle, largest_lead_angle, ckreturn_angle,
ckself_locking, cknotself_locking, ckmateforce, ckmateforce2, cksepforce, cksepforce2,
ckstrain2, cklength );

/*****
**** INSTANCE: geometry
*****/
MakeInstance( geometry, Button );
geometry:X = 384;
geometry:Y = 119;
geometry:Title = "Change Geometry";
geometry:Width = 192;
geometry:Height = 38;
geometry:Visible = TRUE;
geometry:Action = change_geometry;
geometry:ShowBorder = TRUE;
ResetImage ( geometry );

```

```

/*****
**** INSTANCE: process
*****/
MakeInstance( process, Button );
process:X = 384;
process:Y = 203;
process:Title = Design;
process:Width = 192;
process:Height = 38;
process:Visible = TRUE;
process:Action = process;
process:ShowBorder = TRUE;
ResetImage ( process );

/*****
**** INSTANCE: Text4
*****/
MakeInstance( Text4, Text );
Text4:X = 153;
Text4:Y = 76;
Text4:Width = 307;
Text4:Height = 38;
Text4:Visible = TRUE;
Text4:Title = "Snap Joint Demonstration";
Text4:ShowBorder = TRUE;
Text4:TextSize = 15;
Text4:Transparent = true;
ResetImage ( Text4 );

/*****
**** INSTANCE: output_soln
*****/
MakeInstance( output_soln, Transcript );
output_soln:X = 655;
output_soln:Y = 422;
output_soln:Visible = TRUE;
output_soln:Width = 307;
output_soln:Height = 268;
ResetImage ( output_soln );

/*****
**** INSTANCE: output_config
*****/
MakeInstance( output_config, Transcript );
output_config:X = 20;
output_config:Y = 422;
output_config:Visible = TRUE;
output_config:Width = 307;
output_config:Height = 268;
ResetImage ( output_config );

```

```

/*****
**** INSTANCE: output_mat
*****/
MakeInstance( output_mat, Transcript );
output_mat:X = 348;
output_mat:Y = 422;
output_mat:Visible = TRUE;
output_mat:Width = 307;
output_mat:Height = 268;
ResetImage ( output_mat );

/*****
**** INSTANCE: select_material
*****/
MakeInstance( select_material, Button );
select_material:X = 384;
select_material:Y = 161;
select_material:Title = "Select Material";
select_material:Width = 192;
select_material:Height = 38;
select_material:Visible = TRUE;
select_material:Action = select;
select_material:ShowBorder = TRUE;
ResetImage ( select_material );

/*****
**** INSTANCE: Text1
*****/
MakeInstance( Text1, Text );
Text1:X = 20;
Text1:Y = 384;
Text1:Width = 307;
Text1:Height = 38;
Text1:Visible = TRUE;
Text1:Title = "Snap Joint Configuration";
Text1:ShowBorder = TRUE;
ResetImage ( Text1 );

/*****
**** INSTANCE: Text3
*****/
MakeInstance( Text3, Text );
Text3:X = 655;
Text3:Y = 384;
Text3:Width = 307;
Text3:Height = 38;
Text3:Visible = TRUE;
Text3:Title = "Design Specifications";
Text3:ShowBorder = TRUE;
ResetImage ( Text3 );

```

```

/*****
**** INSTANCE: Text2
*****/
MakeInstance( Text2, Text );
Text2:X = 348;
Text2:Y = 384;
Text2:Width = 307;
Text2:Height = 38;
Text2:Visible = TRUE;
Text2:Title = "Material Properties";
Text2:ShowBorder = TRUE;
ResetImage ( Text2 );

/*****
**** INSTANCE: Bitmap1
*****/
MakeInstance( Bitmap1, Bitmap );
Bitmap1:X = 153;
Bitmap1:Y = 115;
Bitmap1:Visible = TRUE;
Bitmap1:FileName = snconst.bmp;
Bitmap1:FitToScreen = FALSE;
Bitmap1:Width = 307;
Bitmap1:Height = 192;
ResetImage ( Bitmap1 );

/*****
**** INSTANCE: stop
*****/
MakeInstance( stop, Button );
stop:X = 384;
stop:Y = 288;
stop:Title = Stop;
stop:Width = 192;
stop:Height = 38;
stop:Visible = TRUE;
stop:Action = stop;
stop:ShowBorder = TRUE;
ResetImage ( stop );

/*****
**** INSTANCE: configuration
*****/
MakeInstance( configuration, Button );
configuration:X = 384;
configuration:Y = 76;
configuration:Title = Configuration;
configuration:Width = 192;
configuration:Height = 38;
configuration:Visible = TRUE;
configuration:Action = config;
configuration:ShowBorder = TRUE;
ResetImage ( configuration );

```

```

/*****
**** INSTANCE: NW_button
*****/
MakeInstance( NW_button, Button );
NW_button:X = 153;
NW_button:Y = 307;
NW_button:Title = "Nominal Wall";
NW_button:Width = 153;
NW_button:Height = 38;
NW_button:Visible = FALSE;
ResetImage ( NW_button );

/*****
**** INSTANCE: proj_button
*****/
MakeInstance( proj_button, Button );
proj_button:X = 153;
proj_button:Y = 384;
proj_button:Title = Projections;
proj_button:Width = 153;
proj_button:Height = 38;
proj_button:Visible = FALSE;
proj_button:Action = projections;
ResetImage ( proj_button );

/*****
**** INSTANCE: dep_button
*****/
MakeInstance( dep_button, Button );
dep_button:X = 153;
dep_button:Y = 460;
dep_button:Title = Depressions;
dep_button:Width = 153;
dep_button:Height = 38;
dep_button:Visible = FALSE;
ResetImage ( dep_button );

/*****
**** INSTANCE: sj_button
*****/
MakeInstance( sj_button, Button );
sj_button:X = 384;
sj_button:Y = 364;
sj_button:Title = "Snap Joints";
sj_button:Width = 153;
sj_button:Height = 38;
sj_button:Visible = FALSE;
sj_button:Action = snap_joints;
ResetImage ( sj_button );

```

```

/*****
**** INSTANCE: ribs_button
*****/
MakeInstance( ribs_button, Button );
ribs_button:X = 384;
ribs_button:Y = 422;
ribs_button:Title = Ribs;
ribs_button:Width = 153;
ribs_button:Height = 38;
ribs_button:Visible = FALSE;
ResetImage ( ribs_button );

/*****
**** INSTANCE: cant_button
*****/
MakeInstance( cant_button, Button );
cant_button:X = 614;
cant_button:Y = 326;
cant_button:Title = Cantilever;
cant_button:Width = 153;
cant_button:Height = 38;
cant_button:Visible = FALSE;
cant_button:Action = start_cant;
ResetImage ( cant_button );

/*****
**** INSTANCE: tor_button
*****/
MakeInstance( tor_button, Button );
tor_button:X = 614;
tor_button:Y = 384;
tor_button:Title = Torsional;
tor_button:Width = 153;
tor_button:Height = 38;
tor_button:Visible = FALSE;
ResetImage ( tor_button );

/*****
**** INSTANCE: ann_button
*****/
MakeInstance( ann_button, Button );
ann_button:X = 614;
ann_button:Y = 441;
ann_button:Title = Annular;
ann_button:Width = 153;
ann_button:Height = 38;
ann_button:Visible = FALSE;
ResetImage ( ann_button );

```

```

/*****
**** INSTANCE: reset
*****/
MakeInstance( reset, Button );
reset:X = 384;
reset:Y = 245;
reset:Title = Reset;
reset:Width = 192;
reset:Height = 38;
reset:Visible = TRUE;
reset:Action = reset;
ResetImage ( reset );

/*****
**** INSTANCE: Bayblend.FR.1439
*****/
MakeInstance( Bayblend.FR.1439, material );
Bayblend.FR.1439:type = "ABS+PC Alloy";
Bayblend.FR.1439:flexural_modulus = 360000;
Bayblend.FR.1439:elongation = 3.5;
Bayblend.FR.1439:mu_plastic_plastic = .75;
Bayblend.FR.1439:mu_plastic_metal = .65;
Bayblend.FR.1439:tensile_stress = 7700;
Bayblend.FR.1439:compressive_stress = 12600;

/*****
**** INSTANCE: Calibre.800.10
*****/
MakeInstance( Calibre.800.10, material );
Calibre.800.10:type = PC;
Calibre.800.10:flexural_modulus = 360000;
Calibre.800.10:elongation = 6.5;
Calibre.800.10:mu_plastic_plastic = .55;
Calibre.800.10:mu_plastic_metal = .45;
Calibre.800.10:tensile_stress = 8700;
Calibre.800.10:compressive_stress = 14000;

/*****
**** INSTANCE: Magnum.3661
*****/
MakeInstance( Magnum.3661, material );
Magnum.3661:type = ABS;
Magnum.3661:flexural_modulus = 340000;
Magnum.3661:elongation = 2.3;
Magnum.3661:mu_plastic_plastic = .75;
Magnum.3661:mu_plastic_metal = .65;
Magnum.3661:tensile_stress = 5000;
Magnum.3661:compressive_stress = 8800;

```



```
/******  
**** INSTANCE: Pulse.1725  
*****/  
MakeInstance( Pulse.1725, material );  
Pulse.1725:type = "ABS+PC Alloy";  
Pulse.1725:flexural_modulus = 400000;  
Pulse.1725:elongation = 4.0;  
Pulse.1725:mu_plastic_plastic = .75;  
Pulse.1725:mu_plastic_metal = .65;  
Pulse.1725:tensile_stress = 8400;  
Pulse.1725:compressive_stress = 11000;
```

```

/*****
/**   ALL RULES ARE SAVED BELOW   **/
*****/

/*****
**** RULE: ckdesign
*****/
MakeRule( ckdesign, [],
  design:strain < design:allowstrain
  And ( design:sepforce < cantilever:sepforce Or cantilever:self_locking #= yes
        Or cantilever:sepforce == 0 )
  And ( design:mateforce < cantilever:mateforce Or cantilever:mateforce == 0 )
  And design:length <= cantilever:maxlength,
  design:criteria = good
);
SetRulePriority( ckdesign, 12 );

/*****
**** RULE: ckstrain
*****/
MakeRule( ckstrain, [],
  design:strain > design:allowstrain,
  {
  cantilever:undercut = cantilever:undercut / 1.1;
  SendMessage( cantilever, calculate );
  } );
SetRulePriority( ckstrain, 10 );

/*****
**** RULE: ckstrain2
*****/
MakeRule( ckstrain2, [],
  design:strain > design:allowstrain,
  {
  cantilever:length = cantilever:length * 1.1;
  SendMessage( cantilever, calculate );
  } );
SetRulePriority( ckstrain2, 5 );

/*****
**** RULE: ckmateforce
*****/
MakeRule( ckmateforce, [],
  cantilever:mateforce != 0 And design:mateforce > cantilever:mateforce,
  {
  cantilever:length = cantilever:length * 1.1;
  SendMessage( cantilever, calculate );
  } );
SetRulePriority( ckmateforce, 10 );

```

```

/*****
**** RULE: ckmateforce2
*****/
MakeRule( ckmateforce2, [],
cantilever:mateforce != 0 And design:mateforce > cantilever:mateforce,
{
cantilever:lead_angle = cantilever:lead_angle / 1.1;
SendMessage( cantilever, calculate );
});
SetRulePriority( ckmateforce2, 5 );

/*****
**** RULE: ckseforce
*****/
MakeRule( ckseforce, [],
cantilever:seforce != 0 And cantilever:self_locking != no
And design:seforce > cantilever:seforce,
{
cantilever:length = cantilever:length * 1.1;
SendMessage( cantilever, calculate );
});
SetRulePriority( ckseforce, 10 );

/*****
**** RULE: ckseforce2
*****/
MakeRule( ckseforce2, [],
cantilever:seforce != 0 And cantilever:self_locking != no
And design:seforce > cantilever:seforce,
{
cantilever:return_angle = cantilever:return_angle / 1.1;
SendMessage( cantilever, calculate );
});
SetRulePriority( ckseforce2, 5 );

/*****
**** RULE: cklength
*****/
MakeRule( cklength, [],
Not( Null?( cantilever:maxlength ) ) And cantilever:length
> cantilever:maxlength,
{
cantilever:length = cantilever:maxlength;
SendMessage( cantilever, calculate );
DeactivateRule( cktensile_stress );
DeactivateRule( ckcompressive_stress );
DeactivateRule( ckmateforce );
DeactivateRule( ckseforce );
DeactivateRule( ckstrain2 );
});

```

```

/*****
**** RULE: cktensile
*****/
MakeRule( cktensile, [],
  design:tensile_stress > cantilever:material_type:tensile_stress
  And cantilever:self_locking #= no,
  {
    cantilever:return_angle = cantilever:return_angle / 1.1;
    SendMessage( cantilever, calculate );
  } );
SetRulePriority( cktensile, 10 );

/*****
**** RULE: ckcompressive
*****/
MakeRule( ckcompressive, [],
  design:compressive_stress > cantilever:material_type:compressive_stress,
  {
    cantilever:lead_angle = cantilever:lead_angle / 1.1;
    SendMessage( cantilever, calculate );
  } );
SetRulePriority( ckcompressive, 5 );

/*****
**** RULE: cktensile_stress
*****/
MakeRule( cktensile_stress, [],
  design:tensile_stress > cantilever:material_type:tensile_stress
  And cantilever:self_locking #= no,
  {
    cantilever:length = cantilever:length * 1.1;
    SendMessage( cantilever, calculate );
  } );
SetRulePriority( cktensile_stress, 5 );

/*****
**** RULE: ckcompressive_stress
*****/
MakeRule( ckcompressive_stress, [],
  design:compressive_stress > cantilever:material_type:compressive_stress,
  {
    cantilever:length = cantilever:length * 1.1;
    SendMessage( cantilever, calculate );
  } );
SetRulePriority( ckcompressive_stress, 10 );

/*****
**** RULE: smallest_lead_angle
*****/
MakeRule( smallest_lead_angle, [],
  cantilever:lead_angle < 10,
  {
    cantilever:lead_angle = 10;
    PostMessage( "Lead angle must be at least 10 deg" );
  } );

```

```

SendMessage( cantilever, calculate );
});
SetRulePriority( smallest_lead_angle, 20 );
/*****
**** RULE: largest_lead_angle
*****/
MakeRule( largest_lead_angle, [],
cantilever:lead_angle > 35,
{
cantilever:lead_angle = 35;
PostMessage( "Lead angle must be less than 35 deg" );
SendMessage( cantilever, calculate );
});
SetRulePriority( largest_lead_angle, 20 );
/*****
**** RULE: ckreturn_angle
*****/
MakeRule( ckreturn_angle, [],
cantilever:return_angle < cantilever:lead_angle,
{
cantilever:return_angle = cantilever:lead_angle;
PostMessage( "Return angle must not be smaller than lead angle" );
SendMessage( cantilever, calculate );
});
SetRulePriority( ckreturn_angle, 15 );
/*****
**** RULE: ckself_locking
*****/
MakeRule( ckself_locking, [],
cantilever:self_locking != yes And cantilever:return_angle < Global:angle,
{
cantilever:return_angle = Global:angle;
PostMessage( "Return angle set to minimum allowable" );
DeactivateRule( ckself_locking );
DeactivateRule( cknotself_locking );
});
SetRulePriority( ckself_locking, 20 );
/*****
**** RULE: cknotself_locking
*****/
MakeRule( cknotself_locking, [],
cantilever:self_locking != no And cantilever:return_angle > Global:angle,
{
PostMessage( "Return angle must be smaller" );
cantilever:return_angle = 45;
SendMessage( cantilever, calculate );
DeactivateRule( ckself_locking );
DeactivateRule( cknotself_locking );
});
SetRulePriority( cknotself_locking, 20 );

```

```

/*****
/**   ALL GOALS ARE SAVED BELOW   **/
*****/

/*****
****  GOAL: gooddesign
*****/
MakeGoal( gooddesign,
         design:criteria #= good );

/*****
/**   ALL FUNCTIONS ARE SAVED BELOW   **/
*****/

/*****
****  FUNCTION: Start
*****/
MakeFunction( init, [],
{
  SetWindowBackground( SESSION, 0, 0, 100 );
  RemoveWindowMenu( SESSION );
  MaximizeWindow( SESSION );
  FreezeWindow( SESSION );
  reset();
} );

/*****
****  FUNCTION: config
*****/
MakeFunction( config, [],
{
  SendMessage( Global:feature, init );
  SendMessage( Global:feature, output_config );
  ClearTranscriptImage( output_soln );
} );

/*****
****  FUNCTION: change_geometry
*****/
MakeFunction( change_geometry, [],
{
  SendMessage( Global:feature, change_geometry );
  SendMessage( Global:feature, output_config );
  ClearTranscriptImage( output_soln );
} );

/*****
****  FUNCTION: select
*****/
MakeFunction( select, [],
{
  ClearTranscriptImage( output_mat );
  SendMessage( material, select );
  If Not( Global:feature:material_type #= "NEW MATERIAL DATABASE" )
    Then SendMessage( material, output_mat )

```

```

Else {
    SendMessage (material, select);
    SendMessage (material, output_mat);
};
});

/*****
**** FUNCTION: process
*****/
MakeFunction( process, [],
{
    PostBusy( ON );
    ClearTranscriptImage( output_config );
    ClearTranscriptImage( output_soln );
    SendMessage( Global:feature, calculate );
    design:criteria = NULL;
    Assert( cantilever:lead_angle );
    Assert( cantilever:return_angle );
    Assert( cantilever:self_locking );
    Assert( cantilever:mateforce );
    Assert( design:strain );
    SetForwardChainMode( BESTFIRST );
    ForwardChain( gooddesign, Global:RuleSet );
    SendMessage( Global:feature, output_config );
    SendMessage( Global:feature, output_soln );
    write_feadata( );
    PostBusy( OFF );
});

/*****
**** FUNCTION: reset
*****/
MakeFunction( reset, [],
{
    HideImage( Bitmap1 );
    HideImage( configuration );
    HideImage( geometry );
    HideImage( select_material );
    HideImage( process );
    HideImage( reset );
    HideImage( stop );
    HideImage( Text1 );
    HideImage( Text2 );
    HideImage( Text3 );
    HideImage( Text4 );
    HideImage( output_config );
    HideImage( output_mat );
    HideImage( output_soln );
    ShowImage( dep_button );
    ShowImage( proj_button );
    ShowImage( NW_button );
});

```

```

/*****
**** FUNCTION: stop
*****/
MakeFunction( stop, [],
Exit( ) );

/*****
**** FUNCTION: projections
*****/
MakeFunction( projections, [],
{
ShowImage( sj_button );
ShowImage( ribs_button );
} );

/*****
**** FUNCTION: snap_joints
*****/
MakeFunction( snap_joints, [],
{
ShowImage( cant_button );
ShowImage( tor_button );
ShowImage( ann_button );
} );

/*****
**** FUNCTION: start_cant
*****/
MakeFunction( start_cant, [],
{
Global:feature = cantilever;
Text4:Title = "Snap Joint Demonstration";
Bitmap1:FileName = snconst.bmp;
Text1:Title = "Snap Joint Configuration";
SetValue( Global:RuleSet, ckstrain, cktenstile_stress, ckcompressive_stress,
ckdesign, cktenstile, ckcompressive, smallest_lead_angle,
largest_lead_angle, ckreturn_angle, ckself_locking,
cknotself_locking, ckmateforce, ckmateforce2, cksepforce,
cksepforce2, ckstrain2, cklength );
HideImage( NW_button );
HideImage( proj_button );
HideImage( dep_button );
HideImage( sj_button );
HideImage( ribs_button );
HideImage( cant_button );
HideImage( tor_button );
HideImage( ann_button );
ClearTranscriptImage( output_config );
ClearTranscriptImage( output_mat );
ClearTranscriptImage( output_soln );
ShowImage( Text4 );
ShowImage( Bitmap1 );
ShowImage( configuration );

```



```

ShowImage( geometry );
ShowImage( select_material );
ShowImage( process );
ShowImage( reset );
ShowImage( stop );
ShowImage( Text1 );
ShowImage( Text2 );
ShowImage( Text3 );
ShowImage( Text4 );
ShowImage( output_config );
ShowImage( output_mat );
ShowImage( output_soln );
});

/*****
**** FUNCTION: write_feadata
*****/
MakeFunction( write_feadata, [],
{
  OpenWriteFile( feadata.txt );
  WriteLine( "K : /CO U 1 snapjoint" );
  WriteLine( "K :", cantilever:length, cantilever:width, cantilever:thickness,
    cantilever:undercut, cantilever:lead_angle, cantilever:return_angle );
  WriteLine( "K : PR E" );
  WriteLine( "E : **** END OF SESSION ****" );
  CloseWriteFile( );
  Execute( "dos2aix", "feadata.txt", "feadata.prg" );
});

/*****
**** FUNCTION: loaddb
*****/
MakeFunction( loaddb, [],
{
  Execute( "createdb.bat" );
  ForAll [ x|material ]
    DeleteInstance( x );
  DBOpenFile( material.dbf );
  DBGetFieldNames( Global:fieldnames );
  RemoveFromList( Global:fieldnames, NAME );
  GetSlotList( material, Global:slotnames );
  DBSetMapParameters( Global:slotnames, Global:fieldnames );
  Global:num = DBGetNumberOfRows( );
  For x [1 Global:num ]
    {
      Global:instance = DBReadCell( x, 1 );
      MakeInstance( Global:instance, material );
      DBMapRowToInstance( Global:instance );
      If ( Global:instance:tensile_stress == 0 Or Global:instance:elongation == 0
        Or Global:instance:flexural_modulus == 0
        Or Global:instance:compressive_stress == 0
        Or Global:instance:mu_plastic_plastic == 0
        Or Global:instance:mu_plastic_metal == 0 )

```

```

Then PostInputForm( "Enter Missing Material Properties for "
# Global:instance, Global:instance:flag, "Delete from material selection list?",
Global:instance:tensile_stress, "Tensile Stress @ yield",
Global:instance:elongation, "Elongation @ yield",
Global:instance:flexural_modulus, "Flexural Modulus",
Global:instance:compressive_stress, "Compressive Stress @ yield",
Global:instance:mu_plastic_plastic, "Coef of Friction (plas/plas)",
Global:instance:mu_plastic_metal, "Coef of Friction (plas/metal)");
If ( Not( Null?( Global:instance:flag ) ) And Global:instance:flag #= yes )
Then DeleteInstance( Global:instance );
};
DBCloseFile( material.dbf );
});

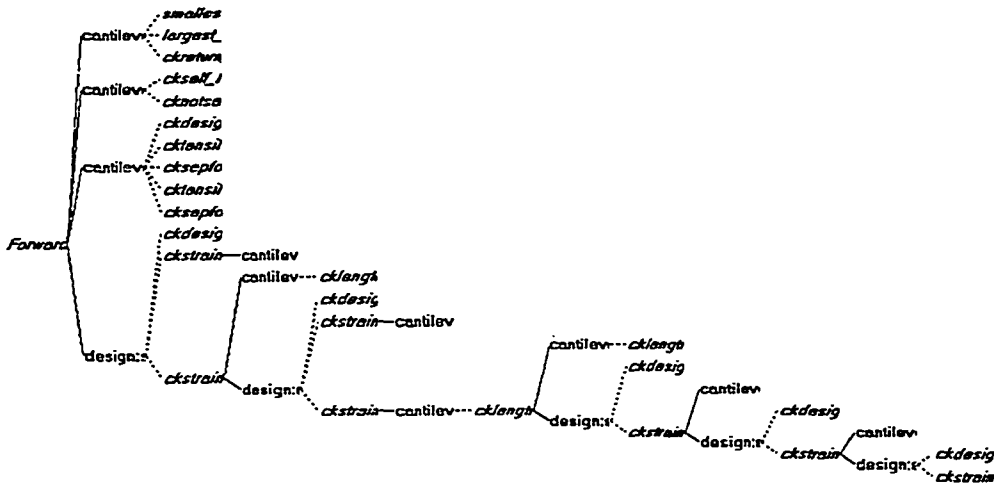
/*****
**** FUNCTION: setup_graphics
*****/
MakeFunction( setup_graphics, [],
{
Global:xscreen = GetScreenWidth( );
Global:yscreen = GetScreenHeight( );
Button:Width = .30 * Global:xscreen;
Button:Height = .05 * Global:yscreen;
Bitmap:Width = .30 * Global:xscreen;
Bitmap:Height = .25 * Global:yscreen;
Transcript:Width = .30 * Global:xscreen;
Transcript:Height = .35 * Global:yscreen;
Text:Width = .30 * Global:xscreen;
Text:Height = Button:Height;
Button:X = 2 * Bitmap:Width;
configuration:Y = Button:Height * 2;
geometry:Y = Button:Height * 3.1;
select_material:Y = Button:Height * 4.2;
process:Y = Button:Height * 5.3;
reset:Y = Button:Height * 6.4;
stop:Y = Button:Height * 7.5;
Bitmap1:X = Bitmap:Width / 2;
Bitmap1:Y = Button:Height * 3;
Text1:X = Global:xscreen / 50;
Text1:Y = Button:Height * 9;
Text2:X = Text1:X * 2 + Text:Width;
Text2:Y = Button:Height * 9;
Text3:X = Text1:X * 3 + Text:Width * 2;
Text3:Y = Button:Height * 9;
Text4:X = Bitmap:Width / 2;
Text4:Y = Button:Height * 2;
output_config:X = Text1:X;
output_config:Y = Button:Height * 10;
output_mat:X = Text2:X;
output_mat:Y = Button:Height * 10;
output_soln:X = Text3:X;
output_soln:Y = Button:Height * 10;
NW_button:Width = Button:Width / 2;

```

```
NW_button:X = Button:Width / 2;
NW_button:Y = Global:yscreen / 2 - Button:Height * 2;
proj_button:Width = Button:Width / 2;
proj_button:X = Button:Width / 2;
proj_button:Y = Global:yscreen / 2;
dep_button:Width = Button:Width / 2;
dep_button:X = Button:Width / 2;
dep_button:Y = Global:yscreen / 2 + Button:Height * 2;
sj_button:Width = Button:Width / 2;
sj_button:X = Button:Width * 1.25;
sj_button:Y = proj_button:Y - Button:Height / 2;
ribs_button:Width = Button:Width / 2;
ribs_button:X = Button:Width * 1.25;
ribs_button:Y = proj_button:Y + Button:Height;
cant_button:Width = Button:Width / 2;
cant_button:X = Button:Width * 2;
cant_button:Y = proj_button:Y - Button:Height * 1.5;
tor_button:Width = Button:Width / 2;
tor_button:X = Button:Width * 2;
tor_button:Y = proj_button:Y;
ann_button:Width = Button:Width / 2;
ann_button:X = Button:Width * 2;
ann_button:Y = proj_button:Y + Button:Height * 1.5;
} );
```

APPENDIX B
 RULE TRACE EXAMPLES

TRACE: CONFLICT RESOLUTION / BEST FIRST



Asserting: cantilever:lead_angle as 35.
 Asserting: cantilever:return_angle as 75.
 Asserting: cantilever:self_locking as yes.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.
 Mode: BESTFIRST IGNORE
 Evaluating: cantilever:lead_angle
 Relevant rules:
 smallest_lead_angle largest_lead_angle ckreturn_angle
 Testing Rule: smallest_lead_angle FALSE
 Testing Rule: largest_lead_angle FALSE
 Testing Rule: ckreturn_angle FALSE
 Evaluating: cantilever:return_angle
 Relevant rules:
 ckself_locking cknotself_locking ckreturn_angle
 Testing Rule: ckself_locking FALSE
 Testing Rule: cknotself_locking FALSE
 Testing Rule: ckreturn_angle FALSE
 Evaluating: cantilever:self_locking
 Relevant rules:
 ckself_locking cknotself_locking ckdesign cktensile cksepio cktensile_stress
 cksepio2
 Testing Rule: ckself_locking FALSE
 Testing Rule: cknotself_locking FALSE

Testing Rule: ckdesign FALSE
 Testing Rule: cktensile FALSE
 Testing Rule: cksepforce FALSE
 Testing Rule: cktensile_stress FALSE
 Testing Rule: cksepforce2 FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.108000.
 design:strain is set to 0.044223.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 62.437463.
 design:sepforce is set to -124.921366.
 design:tensile_stress is set to -1998.741856.
 design:compressive_stress is set to 998.999408.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.742500.
 design:strain is set to 0.036548.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 46.910274.
 design:sepforce is set to -93.855440.
 design:tensile_stress is set to -1501.687040.
 design:compressive_stress is set to 750.564384.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.098182.
 design:strain is set to 0.033227.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 42.647684.
 design:sepforce is set to -85.327090.
 design:tensile_stress is set to -1365.233440.
 design:compressive_stress is set to 682.362944.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.816750.
 design:strain is set to 0.027460.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 32.041453.
 design:sepforce is set to -64.106737.
 design:tensile_stress is set to -1025.707792.
 design:compressive_stress is set to 512.663248.
 Evaluating: cantilever:length
 Relevant rules:

cklength
 Testing Rule: cklength TRUE
 cantilever:length is set to .75.
 design:strain is set to 0.032565.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 41.380015.
 design:sepforce is set to -82.790807.
 design:tensile_stress is set to -1324.652912.
 design:compressive_stress is set to 662.080240.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.
 Deactivating Rule: ckmateforce.
 Deactivating Rule: cksepforce.
 Deactivating Rule: ckstrain2.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.029604.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 37.617501.
 design:sepforce is set to -75.262982.
 design:tensile_stress is set to -1204.207712.
 design:compressive_stress is set to 601.880016.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.026914.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 34.199344.
 design:sepforce is set to -68.424126.
 design:tensile_stress is set to -1094.786016.
 design:compressive_stress is set to 547.189504.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign TRUE
 design:criteria is set to good.
 Done forwardChaining.

TRACE: CONFLICT RESOLUTION / BREADTH FIRST



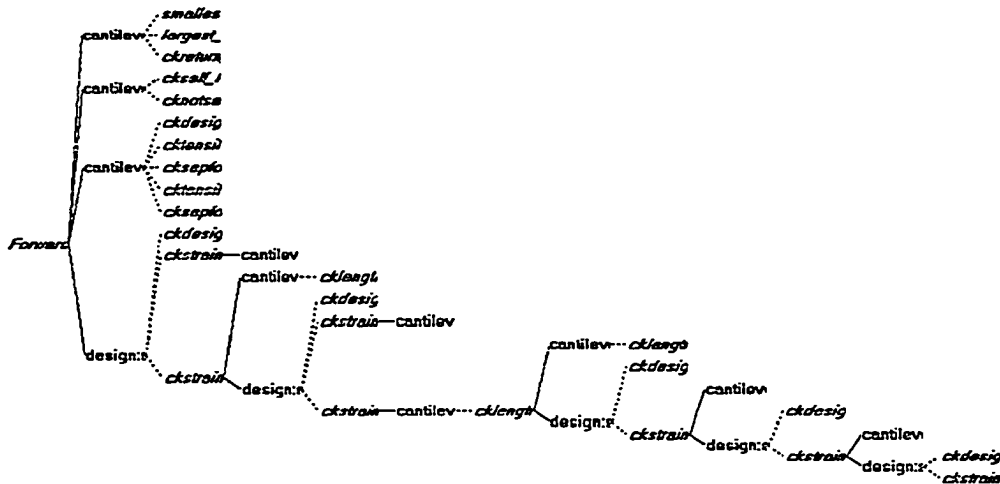
Asserting: cantilever:lead_angle as 35.
Asserting: cantilever:return_angle as 75.
Asserting: cantilever:self_locking as yes.
Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.
Mode: BREADTHFIRST IGNORE
Evaluating: cantilever:lead_angle
Relevant rules:
 smallest_lead_angle largest_lead_angle ckreturn_angle
Testing Rule: smallest_lead_angle FALSE
Testing Rule: largest_lead_angle FALSE
Testing Rule: ckreturn_angle FALSE
Evaluating: cantilever:return_angle
Relevant rules:
 ckself_locking cknotself_locking ckreturn_angle
Testing Rule: ckself_locking FALSE
Testing Rule: cknotself_locking FALSE
Testing Rule: ckreturn_angle FALSE
Evaluating: cantilever:self_locking
Relevant rules:
 ckself_locking cknotself_locking ckdesign cktensile cksepforce cktensile_stress
cksepforce2
Testing Rule: ckself_locking FALSE
Testing Rule: cknotself_locking FALSE
Testing Rule: ckdesign FALSE
Testing Rule: cktensile FALSE
Testing Rule: cksepforce FALSE
Testing Rule: cktensile_stress FALSE
Testing Rule: cksepforce2 FALSE
Evaluating: design:strain
Relevant rules:
 ckdesign ckstrain ckstrain2
Testing Rule: ckdesign FALSE
Testing Rule: ckstrain TRUE
cantilever:undercut is set to 0.108000.

design:strain is set to 0.044223.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 62.437463.
 design:sepforce is set to -124.921366.
 design:tensile_stress is set to -1998.741856.
 design:compressive_stress is set to 998.999408.
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.742500.
 design:strain is set to 0.036548.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 46.910274.
 design:sepforce is set to -93.855440.
 design:tensile_stress is set to -1501.687040.
 design:compressive_stress is set to 750.564384.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.098182.
 design:strain is set to 0.033227.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 42.647684.
 design:sepforce is set to -85.327090.
 design:tensile_stress is set to -1365.233440.
 design:compressive_stress is set to 682.362944.
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.816750.
 design:strain is set to 0.027460.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 32.041453.
 design:sepforce is set to -64.106737.
 design:tensile_stress is set to -1025.707792.
 design:compressive_stress is set to 512.663248.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength TRUE
 cantilever:length is set to .75.
 design:strain is set to 0.032565.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 41.380015.
 design:sepforce is set to -82.790807.
 design:tensile_stress is set to -1324.652912.
 design:compressive_stress is set to 662.080240.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.

Deactivating Rule: ckmateforce.
 Deactivating Rule: cksepforce.
 Deactivating Rule: ckstrain2.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.029604.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 37.617501.
 design:sepforce is set to -75.262982.
 design:tensile_stress is set to -1204.207712.
 design:compressive_stress is set to 601.880016.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.026914.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 34.199344.
 design:sepforce is set to -68.424126.
 design:tensile_stress is set to -1094.786016.
 design:compressive_stress is set to 547.189504.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign TRUE
 design:criteria is set to good.
 Testing Rule: ckstrain FALSE
 Done forwardChaining.

TRACE: CONFLICT RESOLUTION / DEPTH FIRST



Asserting: cantilever:lead_angle as 35.
 Asserting: cantilever:return_angle as 75.
 Asserting: cantilever:self_locking as yes.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.

Mode: DEPTHFIRST IGNORE

Evaluating: cantilever:lead_angle

Relevant rules:

smallest_lead_angle largest_lead_angle ckreturn_angle

Testing Rule: smallest_lead_angle FALSE

Testing Rule: largest_lead_angle FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:return_angle

Relevant rules:

ckself_locking cknotself_locking ckreturn_angle

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:self_locking

Relevant rules:

ckself_locking cknotself_locking ckdesign cktensile cksepf cktensile_stress
 cksepf2

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckdesign FALSE

Testing Rule: cktensile FALSE

Testing Rule: cksepf FALSE

Testing Rule: cktensile_stress FALSE

Testing Rule: cksepf2 FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2

Testing Rule: ckdesign FALSE

Testing Rule: ckstrain TRUE

cantilever:undercut is set to 0.108000.

design:strain is set to 0.044223.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 62.437463.
 design:sepforce is set to -124.921366.
 design:tensile_stress is set to -1998.741856.
 design:compressive_stress is set to 998.999408.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Testing Rule: ckstrain2 TRUE

cantilever:length is set to 0.742500.
 design:strain is set to 0.036548.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 46.910274.
 design:sepforce is set to -93.855440.
 design:tensile_stress is set to -1501.687040.
 design:compressive_stress is set to 750.564384.

Evaluating: cantilever:length

Relevant rules:

cklength

Testing Rule: cklength FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2

Testing Rule: ckdesign FALSE

Testing Rule: ckstrain TRUE

cantilever:undercut is set to 0.098182.
 design:strain is set to 0.033227.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 42.647684.
 design:sepforce is set to -85.327090.
 design:tensile_stress is set to -1365.233440.
 design:compressive_stress is set to 682.362944.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Testing Rule: ckstrain2 TRUE

cantilever:length is set to 0.816750.
 design:strain is set to 0.027400.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 32.041453.
 design:sepforce is set to -64.106737.
 design:tensile_stress is set to -1025.707792.
 design:compressive_stress is set to 512.663248.

Evaluating: cantilever:length

Relevant rules:

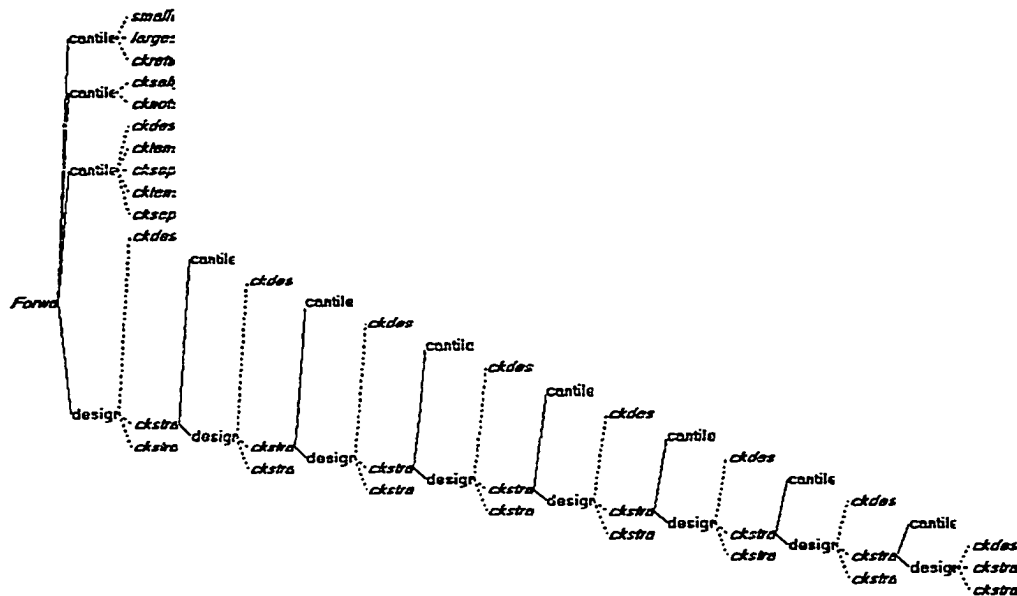
cklength

Testing Rule: cklength TRUE

cantilever:length is set to .75.
 design:strain is set to 0.032565.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 41.380015.
 design:sepforce is set to -82.790807.
 design:tensile_stress is set to -1324.652912.
 design:compressive_stress is set to 662.080240.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.

Deactivating Rule: ckmateforce.
 Deactivating Rule: cksepforce.
 Deactivating Rule: ckstrain2.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.029604.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 37.617501.
 design:sepforce is set to -75.262982.
 design:tensile_stress is set to -1204.207712.
 design:compressive_stress is set to 601.880016.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.026914.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 34.199344.
 design:sepforce is set to -68.424126.
 design:tensile_stress is set to -1094.786016.
 design:compressive_stress is set to 547.189504.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign TRUE
 design:criteria is set to good.
 Done forwardChaining.

TRACE: CONFLICT RESOLUTION / SELECTIVE



Asserting: cantilever:lead_angle as 35.
 Asserting: cantilever:return_angle as 75.
 Asserting: cantilever:self_locking as yes.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.

Mode: SELECTIVE IGNORE

Evaluating: cantilever:lead_angle

Relevant rules:

smallest_lead_angle largest_lead_angle ckreturn_angle

Testing Rule: smallest_lead_angle FALSE

Testing Rule: largest_lead_angle FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:return_angle

Relevant rules:

ckself_locking cknotself_locking ckreturn_angle

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:self_locking

Relevant rules:

ckself_locking cknotself_locking ckdesign cktensile ckseforce cktensile_stress

ckseforce2

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckdesign FALSE

Testing Rule: cktensile FALSE

Testing Rule: ckseforce FALSE

Testing Rule: cktensile_stress FALSE

Testing Rule: ckseforce2 FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.108000.
 design:strain is set to 0.044223.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 62.437463.
 design:sepf force is set to -124.921366.
 design:tensile_stress is set to -1998.741856.
 design:compressive_stress is set to 998.999408.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.098182.
 design:strain is set to 0.040204.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 56.763125.
 design:sepf force is set to -113.568472.
 design:tensile_stress is set to -1817.095552.
 design:compressive_stress is set to 908.210000.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.036548.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 51.601302.
 design:sepf force is set to -103.240985.
 design:tensile_stress is set to -1651.855760.
 design:compressive_stress is set to 825.620832.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Evaluating: design:strain

Relevant rules:

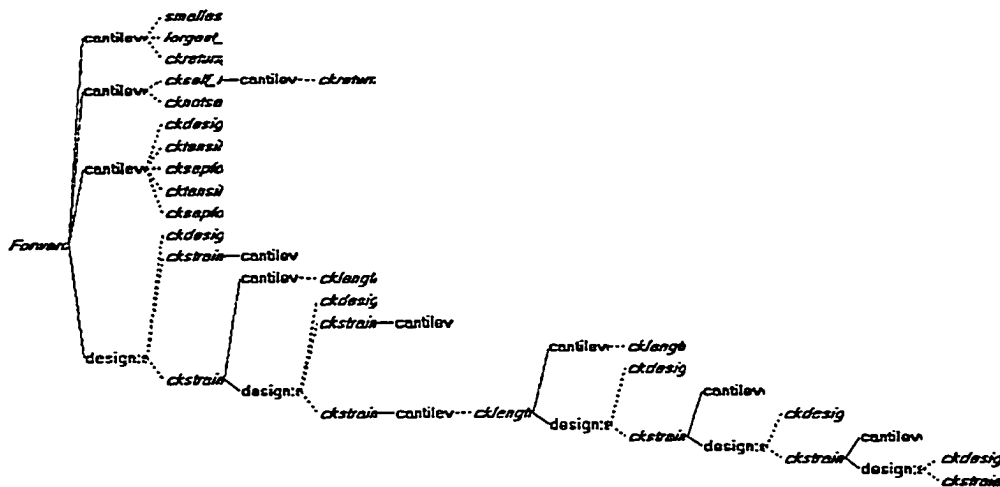
ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.033227.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 46.912454.
 design:sepf force is set to -93.859800.
 design:tensile_stress is set to -1501.756800.
 design:compressive_stress is set to 750.599264.

Evaluating: cantilever:undercut

Relevant rules:

NONE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain ckstrain2
Testing Rule: ckdesign FALSE
Testing Rule: ckstrain TRUE
cantilever:undercut is set to 0.073765.
design:strain is set to 0.030207.
design:allowstrain is set to 0.027300.
design:mateforce is set to 42.648585.
design:sepforce is set to -85.328893.
design:tensile_stress is set to -1365.262288.
design:compressive_stress is set to 682.377360.
Evaluating: cantilever:undercut
Relevant rules:
NONE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain ckstrain2
Testing Rule: ckdesign FALSE
Testing Rule: ckstrain TRUE
cantilever:undercut is set to 0.067059.
design:strain is set to 0.027457.
design:allowstrain is set to 0.027300.
design:mateforce is set to 38.765922.
design:sepforce is set to -77.560678.
design:tensile_stress is set to -1240.970848.
design:compressive_stress is set to 620.254752.
Evaluating: cantilever:undercut
Relevant rules:
NONE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain ckstrain2
Testing Rule: ckdesign FALSE
Testing Rule: ckstrain TRUE
cantilever:undercut is set to 0.060963.
design:strain is set to 0.024961.
design:allowstrain is set to 0.027300.
design:mateforce is set to 35.241877.
design:sepforce is set to -70.509966.
design:tensile_stress is set to -1128.159456.
design:compressive_stress is set to 563.870032.
Evaluating: cantilever:undercut
Relevant rules:
NONE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain ckstrain2
Testing Rule: ckdesign TRUE
design:criteria is set to good.
Done forwardChaining.

TRACE: CHECK RETURN ANGLE



Asserting: cantilever:lead_angle as 35.
 Asserting: cantilever:return_angle as 50.
 Asserting: cantilever:self_locking as yes.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.

Mode: BESTFIRST IGNORE

Evaluating: cantilever:lead_angle

Relevant rules:

smallest_lead_angle largest_lead_angle ckreturn_angle

Testing Rule: smallest_lead_angle FALSE

Testing Rule: largest_lead_angle FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:return_angle

Relevant rules:

ckself_locking cknotself_locking ckreturn_angle

Testing Rule: ckself_locking TRUE

cantilever:return_angle is set to 61.189194.

Deactivating Rule: ckself_locking.

Deactivating Rule: cknotself_locking.

Evaluating: cantilever:self_locking

Relevant rules:

ckdesign ckensile cksepforce ckensile_stress cksepforce2

Testing Rule: ckreturn_angle FALSE

Testing Rule: ckdesign FALSE

Testing Rule: ckensile FALSE

Testing Rule: cksepforce FALSE

Testing Rule: ckensile_stress FALSE

Testing Rule: cksepforce2 FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2

Testing Rule: ckdesign FALSE

Testing Rule: ckstrain TRUE

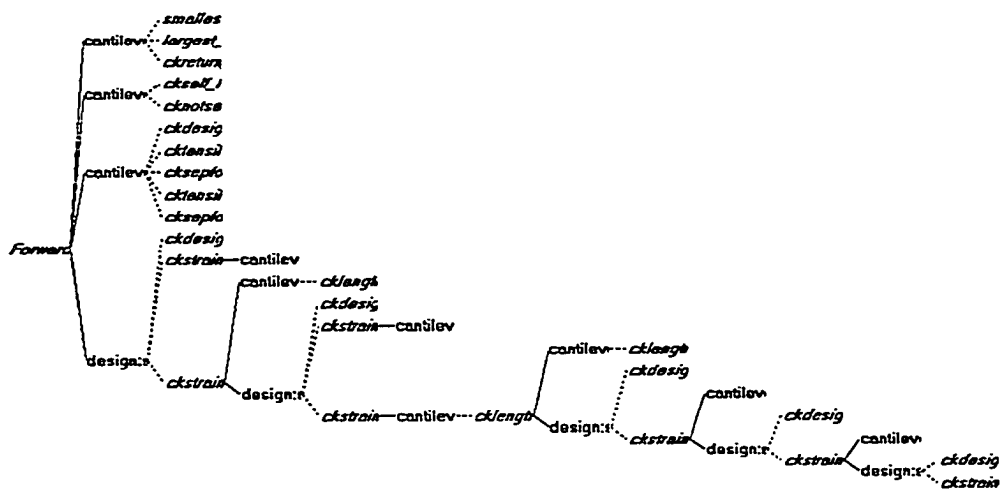
cantilever:undercut is set to 0.108000.

design:strain is set to 0.044223.

design:allowstrain is set to 0.027300.
 design:mateforce is set to 62.437463.
 design:sepforce is set to 24241016.666667.
 design:tensile_stress is set to 387856266.666672.
 design:compressive_stress is set to 998.999408.
 Evaluating: cantilever:return_angle
 Relevant rules:
 ckreturn_angle
 Testing Rule: ckreturn_angle FALSE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.742500.
 design:strain is set to 0.036548.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 46.910274.
 design:sepforce is set to 18212667.333333.
 design:tensile_stress is set to 291402677.333328.
 design:compressive_stress is set to 750.564384.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.098182.
 design:strain is set to 0.033227.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 42.647684.
 design:sepforce is set to 16557739.333333.
 design:tensile_stress is set to 264923829.333328.
 design:compressive_stress is set to 682.362944.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.816750.
 design:strain is set to 0.027460.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 32.041453.
 design:sepforce is set to 12439925.666667.
 design:tensile_stress is set to 199038810.666672.
 design:compressive_stress is set to 512.663248.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength TRUE
 cantilever:length is set to .75.
 design:strain is set to 0.032565.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 41.380015.
 design:sepforce is set to 16065573.

design:tensile_stress is set to 257049168.
 design:compressive_stress is set to 662.080240.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.
 Deactivating Rule: ckmateforce.
 Deactivating Rule: cksepforce.
 Deactivating Rule: ckstrain2.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.029604.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 37.617501.
 design:sepforce is set to 14604797.
 design:tensile_stress is set to 233676752.
 design:compressive_stress is set to 601.880016.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.026914.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 34.199344.
 design:sepforce is set to 13277715.666667.
 design:tensile_stress is set to 212443450.666672.
 design:compressive_stress is set to 547.189504.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign TRUE
 design:criteria is set to good.
 Done forwardChaining.

TRACE: NOT SELF-LOCKING



Asserting: cantilever:lead_angle as 35.
 Asserting: cantilever:return_angle as 50.
 Asserting: cantilever:self_locking as no.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.

Mode: BESTFIRST IGNORE

Evaluating: cantilever:lead_angle

Relevant rules:

smallest_lead_angle largest_lead_angle ckreturn_angle

Testing Rule: smallest_lead_angle FALSE

Testing Rule: largest_lead_angle FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:return_angle

Relevant rules:

ckself_locking cknotself_locking ckreturn_angle

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:self_locking

Relevant rules:

ckself_locking cknotself_locking ckdesign cktensile cksepf cktensile_stress

cksepf2

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckdesign FALSE

Testing Rule: cktensile FALSE

Testing Rule: cksepf FALSE

Testing Rule: cktensile_stress FALSE

Testing Rule: cksepf2 FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2

Testing Rule: ckdesign FALSE

Testing Rule: ckstrain TRUE

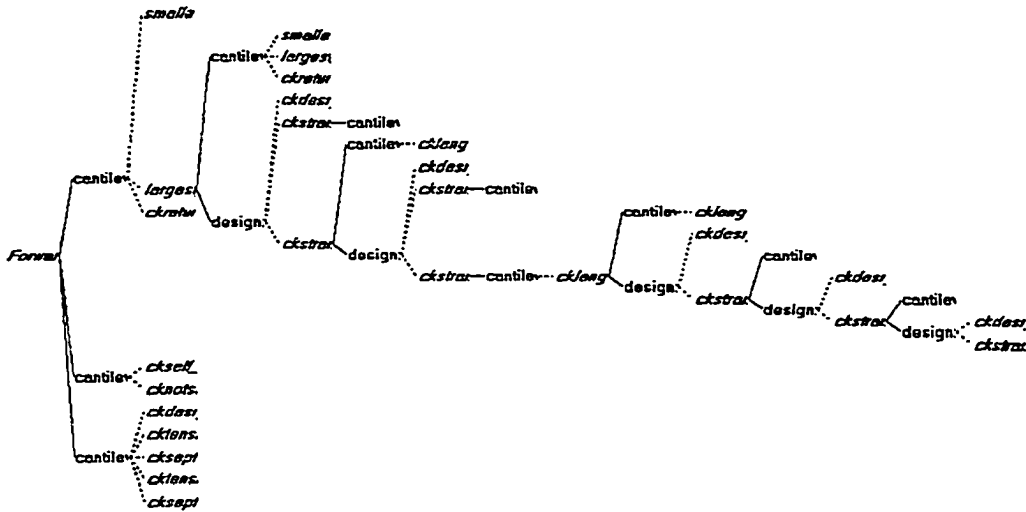
cantilever:undercut is set to 0.108000.

design:strain is set to 0.044223.

design:allowstrain is set to 0.027300.
 design:mateforce is set to 62.437463.
 design:sepforce is set to 155.242146.
 design:tensile_stress is set to 2483.874336.
 design:compressive_stress is set to 998.999408.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.742500.
 design:strain is set to 0.036548.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 46.910274.
 design:sepforce is set to 116.635931.
 design:tensile_stress is set to 1866.174896.
 design:compressive_stress is set to 750.564384.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.098182.
 design:strain is set to 0.033227.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 42.647684.
 design:sepforce is set to 106.037590.
 design:tensile_stress is set to 1696.601440.
 design:compressive_stress is set to 682.362944.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.816750.
 design:strain is set to 0.027460.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 32.041453.
 design:sepforce is set to 79.666656.
 design:tensile_stress is set to 1274.666496.
 design:compressive_stress is set to 512.663248.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength TRUE
 cantilever:length is set to .75.
 design:strain is set to 0.032565.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 41.380015.
 design:sepforce is set to 102.885701.
 design:tensile_stress is set to 1646.171216.
 design:compressive_stress is set to 662.080240.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.
 Deactivating Rule: ckmateforce.

Deactivating Rule: cksepforce.
 Deactivating Rule: ckstrain2.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.029604.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 37.617501.
 design:sepforce is set to 93.530731.
 design:tensile_stress is set to 1496.491696.
 design:compressive_stress is set to 601.880016.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.026914.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 34.199344.
 design:sepforce is set to 85.031956.
 design:tensile_stress is set to 1360.511296.
 design:compressive_stress is set to 547.189504.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign TRUE
 design:criteria is set to good.
 Done forwardChaining.

TRACE: CHECK LEAD ANGLE



Asserting: cantilever:lead_angle as 50.
 Asserting: cantilever:return_angle as 50.
 Asserting: cantilever:self_locking as no.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.
 Mode: BESTFIRST IGNORE
 Evaluating: cantilever:lead_angle
 Relevant rules:
 smallest_lead_angle largest_lead_angle ckreturn_angle
 Testing Rule: smallest_lead_angle FALSE
 Testing Rule: largest_lead_angle TRUE
 cantilever:lead_angle is set to 35.
 design:strain is set to 0.048645.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 68.680781.
 design:sepf force is set to 170.765296.
 design:tensile_stress is set to 2732.244736.
 design:compressive_stress is set to 1098.892496.

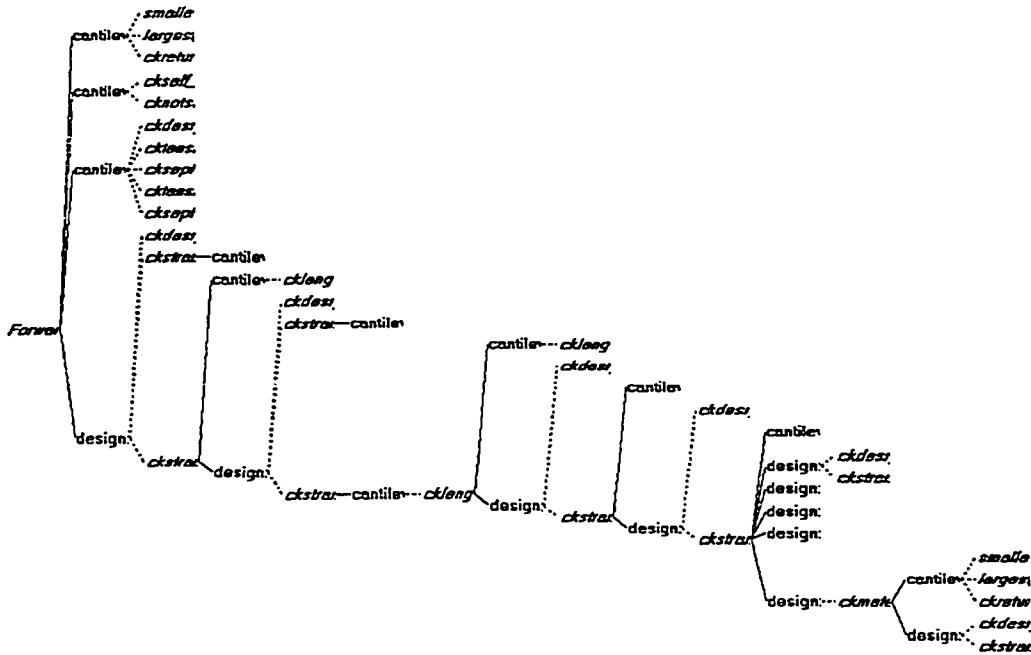
Evaluating: cantilever:return_angle
 Relevant rules:
 ckself_locking cknotself_locking ckreturn_angle
 Testing Rule: ckself_locking FALSE
 Testing Rule: cknotself_locking FALSE
 Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:self_locking
 Relevant rules:
 ckself_locking cknotself_locking ckdesign cktensile cksepf force cktensile_stress
 cksepf force2
 Testing Rule: ckself_locking FALSE
 Testing Rule: cknotself_locking FALSE
 Testing Rule: ckdesign FALSE
 Testing Rule: cktensile FALSE
 Testing Rule: cksepf force FALSE
 Testing Rule: cktensile_stress FALSE
 Testing Rule: cksepf force2 FALSE

Evaluating: cantilever:lead_angle
 Relevant rules:
 smallest_lead_angle largest_lead_angle ckreturn_angle
 Testing Rule: smallest_lead_angle FALSE
 Testing Rule: largest_lead_angle FALSE
 Testing Rule: ckreturn_angle FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.108000.
 design:strain is set to 0.044223.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 62.437463.
 design:sepforce is set to 155.242146.
 design:tensile_stress is set to 2483.874336.
 design:compressive_stress is set to 998.999408.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.742500.
 design:strain is set to 0.036548.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 46.910274.
 design:sepforce is set to 116.635931.
 design:tensile_stress is set to 1866.174896.
 design:compressive_stress is set to 750.564384.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.098182.
 design:strain is set to 0.033227.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 42.647684.
 design:sepforce is set to 106.037590.
 design:tensile_stress is set to 1696.601440.
 design:compressive_stress is set to 682.362944.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.816750.
 design:strain is set to 0.027460.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 32.041453.
 design:sepforce is set to 79.666656.
 design:tensile_stress is set to 1274.666496.
 design:compressive_stress is set to 512.663248.
 Evaluating: cantilever:length

Relevant rules:
 cklength
 Testing Rule: cklength TRUE
 cantilever:length is set to .75.
 design:strain is set to 0.032565.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 41.380015.
 design:sepforce is set to 102.885701.
 design:tensile_stress is set to 1646.171216.
 design:compressive_stress is set to 662.080240.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.
 Deactivating Rule: ckmateforce.
 Deactivating Rule: cksepforce.
 Deactivating Rule: ckstrain2.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.029604.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 37.617501.
 design:sepforce is set to 93.530731.
 design:tensile_stress is set to 1496.491696.
 design:compressive_stress is set to 601.880016.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.026914.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 34.199344.
 design:sepforce is set to 85.031956.
 design:tensile_stress is set to 1360.511296.
 design:compressive_stress is set to 547.189504.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign TRUE
 design:criteria is set to good.
 Done forwardChaining.

TRACE: MAXIMUM MATEFORCE



Asserting: cantilever:lead_angle as 35.
 Asserting: cantilever:return_angle as 75.
 Asserting: cantilever:self_locking as yes.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.
 Mode: BESTFIRST IGNORE
 Evaluating: cantilever:lead_angle
 Relevant rules:
 smallest_lead_angle largest_lead_angle ckreturn_angle
 Testing Rule: smallest_lead_angle FALSE
 Testing Rule: largest_lead_angle FALSE
 Testing Rule: ckreturn_angle FALSE
 Evaluating: cantilever:return_angle
 Relevant rules:
 ckself_locking cknotself_locking ckreturn_angle
 Testing Rule: ckself_locking FALSE
 Testing Rule: cknotself_locking FALSE
 Testing Rule: ckreturn_angle FALSE
 Evaluating: cantilever:self_locking
 Relevant rules:
 ckself_locking cknotself_locking ckdesign cktensile cksepfence cktensile_stress
 cksepfence2
 Testing Rule: ckself_locking FALSE
 Testing Rule: cknotself_locking FALSE
 Testing Rule: ckdesign FALSE
 Testing Rule: cktensile FALSE
 Testing Rule: cksepfence FALSE
 Testing Rule: cktensile_stress FALSE
 Testing Rule: cksepfence2 FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2

Testing Rule: ckdesign FALSE

Testing Rule: ckstrain TRUE

cantilever:undercut is set to 0.108000.

design:strain is set to 0.044223.

design:allowstrain is set to 0.027300.

design:mateforce is set to 62.437463.

design:sepforce is set to -124.921366.

design:tensile_stress is set to -1998.741856.

design:compressive_stress is set to 998.999408.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Testing Rule: ckstrain2 TRUE

cantilever:length is set to 0.742500.

design:strain is set to 0.036548.

design:allowstrain is set to 0.027300.

design:mateforce is set to 46.910274.

design:sepforce is set to -93.855440.

design:tensile_stress is set to -1501.687040.

design:compressive_stress is set to 750.564384.

Evaluating: cantilever:length

Relevant rules:

cklength

Testing Rule: cklength FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2

Testing Rule: ckdesign FALSE

Testing Rule: ckstrain TRUE

cantilever:undercut is set to 0.098182.

design:strain is set to 0.033227.

design:allowstrain is set to 0.027300.

design:mateforce is set to 42.647684.

design:sepforce is set to -85.327090.

design:tensile_stress is set to -1365.233440.

design:compressive_stress is set to 682.362944.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Testing Rule: ckstrain2 TRUE

cantilever:length is set to 0.816750.

design:strain is set to 0.027460.

design:allowstrain is set to 0.027300.

design:mateforce is set to 32.041453.

design:sepforce is set to -64.106737.

design:tensile_stress is set to -1025.707792.

design:compressive_stress is set to 512.663248.

Evaluating: cantilever:length

Relevant rules:

cklength

Testing Rule: cklength TRUE

cantilever:length is set to .75.

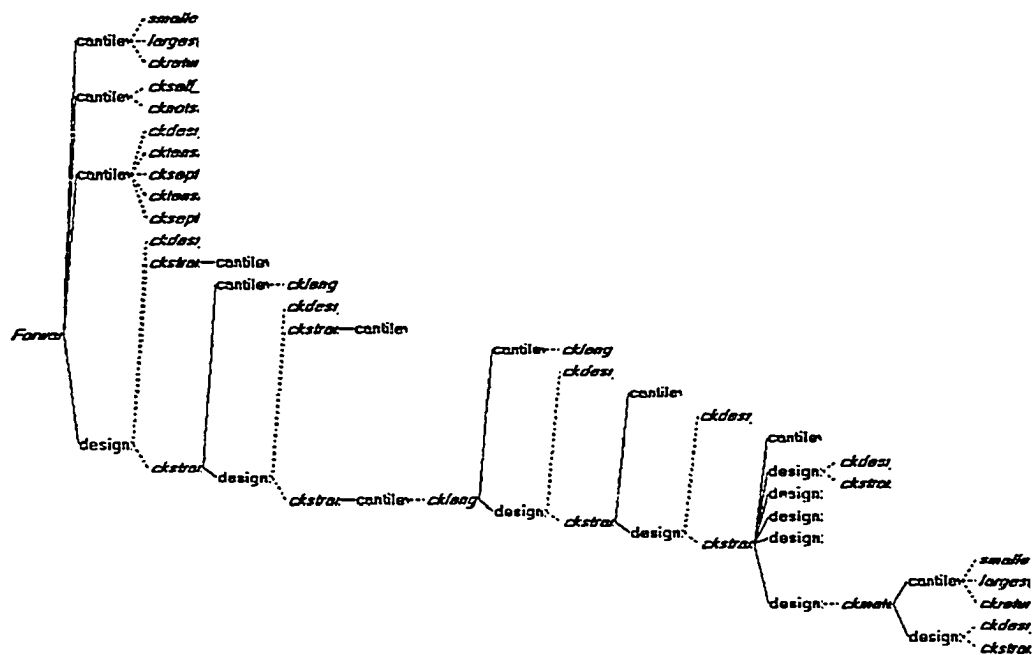
design:strain is set to 0.032565.

design:allowstrain is set to 0.027300.

design:mateforce is set to 41.380015.
 design:sepforce is set to -82.790807.
 design:tensile_stress is set to -1324.652912.
 design:compressive_stress is set to 662.080240.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.
 Deactivating Rule: ckmateforce.
 Deactivating Rule: cksepforce.
 Deactivating Rule: ckstrain2.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.029604.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 37.617501.
 design:sepforce is set to -75.262982.
 design:tensile_stress is set to -1204.207712.
 design:compressive_stress is set to 601.880016.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.026914.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 34.199344.
 design:sepforce is set to -68.424126.
 design:tensile_stress is set to -1094.786016.
 design:compressive_stress is set to 547.189504.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain FALSE
 Evaluating: design:allowstrain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain FALSE
 Evaluating: design:mateforce
 Relevant rules:
 ckdesign ckmateforce2
 Testing Rule: ckdesign FALSE

Testing Rule: ckmateforce2 TRUE
cantilever:lead_angle is set to 31.818182.
design:strain is set to 0.026914.
design:allowstrain is set to 0.027300.
design:mateforce is set to 29.886331.
design:sepforce is set to -68.424126.
design:tensile_stress is set to -1094.786016.
design:compressive_stress is set to 478.181296.
Evaluating: cantilever:lead_angle
Relevant rules:
smallest_lead_angle largest_lead_angle ckreturn_angle
Testing Rule: smallest_lead_angle FALSE
Testing Rule: largest_lead_angle FALSE
Testing Rule: ckreturn_angle FALSE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain
Testing Rule: ckdesign TRUE
design:criteria is set to good.
Done forwardChaining.

TRACE: MAXIMUM MATEFORCE / NOT SELF-LOCKING



Asserting: cantilever:lead_angle as 35.
 Asserting: cantilever:return_angle as 50.
 Asserting: cantilever:self_locking as no.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.

Mode: BESTFIRST IGNORE

Evaluating: cantilever:lead_angle

Relevant rules:

smallest_lead_angle largest_lead_angle ckreturn_angle

Testing Rule: smallest_lead_angle FALSE

Testing Rule: largest_lead_angle FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:return_angle

Relevant rules:

ckself_locking cknotself_locking ckreturn_angle

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:self_locking

Relevant rules:

ckself_locking cknotself_locking ckdesign cktensile cksepf cktensile_stress
 cksepf2

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckdesign FALSE

Testing Rule: cktensile FALSE

Testing Rule: cksepf FALSE

Testing Rule: cktensile_stress FALSE

Testing Rule: cksepf2 FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2

Testing Rule: ckdesign FALSE

Testing Rule: ckstrain TRUE

cantilever:undercut is set to 0.108000.

design:strain is set to 0.044223.

design:allowstrain is set to 0.027300.

design:mateforce is set to 62.437463.

design:sepforce is set to 155.242146.

design:tensile_stress is set to 2483.874336.

design:compressive_stress is set to 998.999408.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Testing Rule: ckstrain2 TRUE

cantilever:length is set to 0.742500.

design:strain is set to 0.036548.

design:allowstrain is set to 0.027300.

design:mateforce is set to 46.910274.

design:sepforce is set to 116.635931.

design:tensile_stress is set to 1866.174896.

design:compressive_stress is set to 750.564384.

Evaluating: cantilever:length

Relevant rules:

cklength

Testing Rule: cklength FALSE

Evaluating: design:strain

Relevant rules:

ckdesign ckstrain ckstrain2

Testing Rule: ckdesign FALSE

Testing Rule: ckstrain TRUE

cantilever:undercut is set to 0.098182.

design:strain is set to 0.033227.

design:allowstrain is set to 0.027300.

design:mateforce is set to 42.647684.

design:sepforce is set to 106.037590.

design:tensile_stress is set to 1696.601440.

design:compressive_stress is set to 682.362944.

Evaluating: cantilever:undercut

Relevant rules:

NONE

Testing Rule: ckstrain2 TRUE

cantilever:length is set to 0.816750.

design:strain is set to 0.027460.

design:allowstrain is set to 0.027300.

design:mateforce is set to 32.041453.

design:sepforce is set to 79.666656.

design:tensile_stress is set to 1274.666496.

design:compressive_stress is set to 512.663248.

Evaluating: cantilever:length

Relevant rules:

cklength

Testing Rule: cklength TRUE

cantilever:length is set to .75.

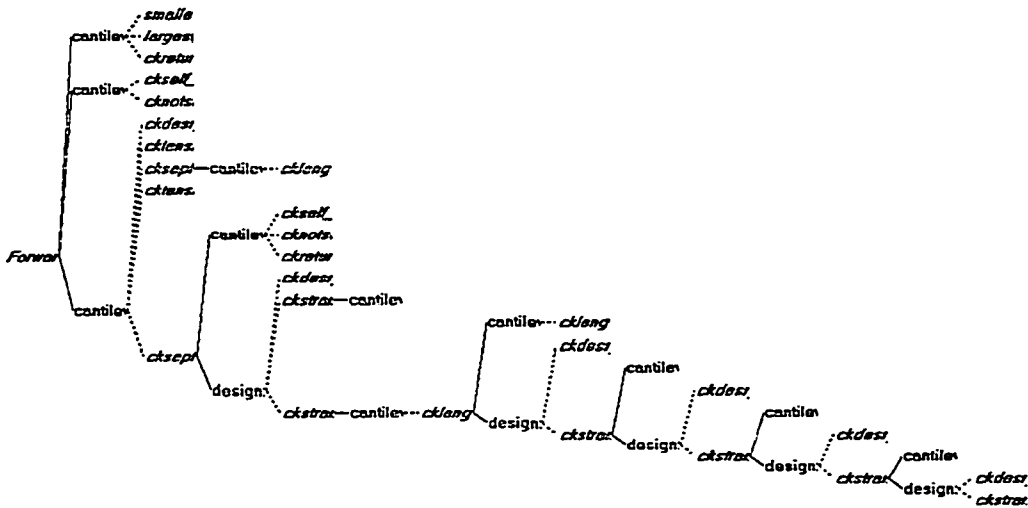
design:strain is set to 0.032565.

design:allowstrain is set to 0.027300.

design:mateforce is set to 41.380015.
 design:sepforce is set to 102.885701.
 design:tensile_stress is set to 1646.171216.
 design:compressive_stress is set to 662.080240.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.
 Deactivating Rule: ckmateforce.
 Deactivating Rule: cksepforce.
 Deactivating Rule: ckstrain2.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.089256.
 design:strain is set to 0.029604.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 37.617501.
 design:sepforce is set to 93.530731.
 design:tensile_stress is set to 1496.491696.
 design:compressive_stress is set to 601.880016.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.081142.
 design:strain is set to 0.026914.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 34.199344.
 design:sepforce is set to 85.031956.
 design:tensile_stress is set to 1360.511296.
 design:compressive_stress is set to 547.189504.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain FALSE
 Evaluating: design:allowstrain
 Relevant rules:
 ckdesign ckstrain
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain FALSE
 Evaluating: design:mateforce
 Relevant rules:
 ckdesign ckmateforce2
 Testing Rule: ckdesign FALSE

Testing Rule: ckmateforce2 TRUE
cantilever:lead_angle is set to 31.818182.
design:strain is set to 0.026914.
design:allowstrain is set to 0.027300.
design:mateforce is set to 29.886331.
design:sepforce is set to 85.031956.
design:tensile_stress is set to 1360.511296.
design:compressive_stress is set to 478.181296.
Evaluating: cantilever:lead_angle
Relevant rules:
smallest_lead_angle largest_lead_angle ckreturn_angle
Testing Rule: smallest_lead_angle FALSE
Testing Rule: largest_lead_angle FALSE
Testing Rule: ckreturn_angle FALSE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain
Testing Rule: ckdesign TRUE
design:criteria is set to good.
Done forwardChaining.

TRACE: MAXIMUM SEPARATING FORCE



Asserting: cantilever:lead_angle as 35.
 Asserting: cantilever:return_angle as 50.
 Asserting: cantilever:self_locking as no.
 Asserting: design:strain as 0.048645.

Forward Chaining: 17 Rules.

Mode: BESTFIRST IGNORE

Evaluating: cantilever:lead_angle

Relevant rules:

smallest_lead_angle largest_lead_angle ckreturn_angle

Testing Rule: smallest_lead_angle FALSE

Testing Rule: largest_lead_angle FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:return_angle

Relevant rules:

ckself_locking cknotself_locking ckreturn_angle

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckreturn_angle FALSE

Evaluating: cantilever:self_locking

Relevant rules:

ckself_locking cknotself_locking ckdesign cktensile cksepforce cktensile_stress

cksepforce2

Testing Rule: ckself_locking FALSE

Testing Rule: cknotself_locking FALSE

Testing Rule: ckdesign FALSE

Testing Rule: cktensile FALSE

Testing Rule: cksepforce TRUE

cantilever:length is set to 0.742500.

design:strain is set to 0.040203.

design:allowstrain is set to 0.027300.

design:mateforce is set to 51.601557.

design:sepforce is set to 128.300157.

design:tensile_stress is set to 2052.802512.

design:compressive_stress is set to 825.624912.

Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cktensile_stress FALSE
 Testing Rule: cksepforce2 TRUE
 cantilever:return_angle is set to 45.454545.
 design:strain is set to 0.040203.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 51.601557.
 design:sepforce is set to 90.079478.
 design:tensile_stress is set to 1441.271648.
 design:compressive_stress is set to 825.624912.
 Evaluating: cantilever:return_angle
 Relevant rules:
 ckself_locking cknotself_locking ckreturn_angle
 Testing Rule: ckself_locking FALSE
 Testing Rule: cknotself_locking FALSE
 Testing Rule: ckreturn_angle FALSE
 Testing Rule: cklength FALSE
 Evaluating: design:strain
 Relevant rules:
 ckdesign ckstrain ckstrain2
 Testing Rule: ckdesign FALSE
 Testing Rule: ckstrain TRUE
 cantilever:undercut is set to 0.108000.
 design:strain is set to 0.036548.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 46.910274.
 design:sepforce is set to 81.890028.
 design:tensile_stress is set to 1310.240448.
 design:compressive_stress is set to 750.564384.
 Evaluating: cantilever:undercut
 Relevant rules:
 NONE
 Testing Rule: ckstrain2 TRUE
 cantilever:length is set to 0.816750.
 design:strain is set to 0.030205.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 35.244432.
 design:sepforce is set to 61.525276.
 design:tensile_stress is set to 984.404416.
 design:compressive_stress is set to 563.910912.
 Evaluating: cantilever:length
 Relevant rules:
 cklength
 Testing Rule: cklength TRUE
 cantilever:length is set to .75.
 design:strain is set to 0.035820.
 design:allowstrain is set to 0.027300.
 design:mateforce is set to 45.516109.
 design:sepforce is set to 79.456270.
 design:tensile_stress is set to 1271.300320.
 design:compressive_stress is set to 728.257744.
 Deactivating Rule: cktensile_stress.
 Deactivating Rule: ckcompressive_stress.
 Deactivating Rule: ckmateforce.
 Deactivating Rule: cksepforce.

Deactivating Rule: ckstrain2.
Evaluating: cantilever:length
Relevant rules:
cklength
Testing Rule: cklength FALSE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain
Testing Rule: ckdesign FALSE
Testing Rule: ckstrain TRUE
cantilever:undercut is set to 0.098182.
design:strain is set to 0.032565.
design:allowstrain is set to 0.027300.
design:mateforce is set to 41.380015.
design:sepforce is set to 72.236000.
design:tensile_stress is set to 1155.776000.
design:compressive_stress is set to 662.080240.
Evaluating: cantilever:undercut
Relevant rules:
NONE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain
Testing Rule: ckdesign FALSE
Testing Rule: ckstrain TRUE
cantilever:undercut is set to 0.089256.
design:strain is set to 0.029604.
design:allowstrain is set to 0.027300.
design:mateforce is set to 37.617501.
design:sepforce is set to 65.667879.
design:tensile_stress is set to 1050.686064.
design:compressive_stress is set to 601.880016.
Evaluating: cantilever:undercut
Relevant rules:
NONE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain
Testing Rule: ckdesign FALSE
Testing Rule: ckstrain TRUE
cantilever:undercut is set to 0.081142.
design:strain is set to 0.026914.
design:allowstrain is set to 0.027300.
design:mateforce is set to 34.199344.
design:sepforce is set to 59.700893.
design:tensile_stress is set to 955.214288.
design:compressive_stress is set to 547.189504.
Evaluating: cantilever:undercut
Relevant rules:
NONE
Evaluating: design:strain
Relevant rules:
ckdesign ckstrain
Testing Rule: ckdesign TRUE
design:criteria is set to good.
Done forwardChaining.

APPENDIX C
EDIT PROGRAM

```
DECLARE SUB STRMOD (NSTRG$)
OPEN "G:\KAPPA\SNAP\DATABASES\REPORT.TXT" FOR INPUT AS #1
OPEN "G:\KAPPA\SNAP\DATABASES\MATERIAL.TXT" FOR OUTPUT AS #2
FOR I = 1 TO 7
  INPUT #1, LINE$
NEXT I

DO WHILE NOT EOF(1)
  INPUT #1, LINE$
  NAME1$ = RTRIM$(MID$(LINE$, 1, 16))
  NAME2$ = RTRIM$(MID$(LINE$, 17, 16))
  NTYPE$ = RTRIM$(MID$(LINE$, 33, 51))
  NMOD = VAL(MID$(LINE$, 84, 11))
  ELCNG = VAL(MID$(LINE$, 95, 11))
  TSTRESS = VAL(MID$(LINE$, 106, 11))
  CSTRESS = VAL(MID$(LINE$, 117, 11))
  CALL STRMOD(NAME1$)
  CALL STRMOD(NAME2$)
  NAMES$ = NAME1$ + NAME2$
  IF NTYPE$ = "Polypropylene" THEN
    MUPP = .4
    MUPM = .25
  ELSEIF NTYPE$ = "Polystyrene" THEN
    MUPP = .5
    MUPM = .4
  ELSEIF NTYPE$ = "Styrene Acrylonitrile" THEN
    MUPP = .55
    MUPM = .45
  ELSEIF NTYPE$ = "Polycarbonate" THEN
    MUPP = .55
    MUPM = .45
  ELSEIF NTYPE$ = "Acrylonitrile Butadiene Styrene" THEN
    MUPP = .75
    MUPM = .65
  ELSEIF NTYPE$ = "Polyvinyl Chloride" THEN
    MUPP = .6
    MUPM = .55
  ELSEIF NTYPE$ = "Acrylonitrile Butadiene Styrene + PC Alloy" THEN
    MUPP = .65
    MUPM = .55
```

```
ELSE
  MUPP = 0
  MUPM = 0
END IF
WRITE #2, NAMES$, NTYPE$, NMOD, ELONG, MUPP, MUPM, TSTRESS,
      CSTRESS
LOOP
CLOSE #1
CLOSE #2
END

SUB STRMOD (NSTRG$)

FOR NCHAR = 1 TO LEN(NSTRG$)
  CH$ = MID$(NSTRG$, NCHAR, 1)
  IF INSTR(" -()/:", CH$) THEN
    MID$(NSTRG$, NCHAR, 1) = "."
  END IF
NEXT NCHAR

END SUB
```

APPENDIX D
IDEAS SNAP FEATURE

Parameter Name/Number: 1 - LENGTH

Type of Parameter : Prompted

Type of Limit : Min and Max

Type of Units : Length

Prompt: Enter length of snap

Default Value: 10.000

Minimum Value: 0.0010000 Maximum Value: 1.0000E+13

The parameter controls the following entities:

Leaf 1, Extrusion Linear_Dimension_1

Parameter Name/Number: 2 - WIDTH

Type of Parameter : Prompted

Type of Limit : None

Type of Units : Length

Prompt: Enter width of snap

Default Value: -4.0000

The parameter controls the following entities:

Leaf 1, Extrusion Distance in Z

Parameter Name/Number: 3 - THICKNESS

Type of Parameter : Prompted

Type of Limit : Min and Max

Type of Units : Length

Prompt: Enter height of snap

Default Value: 5.0000

Minimum Value: 0.0010000 Maximum Value: 1.0000E+13

The parameter controls the following entities:

Leaf 1, Extrusion Linear_Dimension_3

Parameter Name/Number: 4 - UNDERCUT

Type of Parameter : Prompted

Type of Limit : Min and Max

Type of Units : Length

Prompt: Enter height of undercut

Default Value: 4.0000

Minimum Value: 0.0010000 Maximum Value: 1.0000E+13

The parameter controls the following entities:

Leaf 1, Extrusion Linear_Dimension_4

Parameter Name/Number: 5 - LEADANGLE

Type of Parameter : Prompted

Type of Limit : None

Type of Units : Length

Prompt: Enter lead angle

Default Value: 45.000

The parameter is referenced by the following parameters:

7 - LEADROT

Parameter Name/Number: 6 - RETANGLE

Type of Parameter : Prompted

Type of Limit : None

Type of Units : Length

Prompt: Enter return angle

Default Value: 45.000

The parameter is referenced by the following parameters:

8 - RETROT

Parameter Name/Number: 7 - LEADROT

Type of Parameter : Equational

Type of Limit : Min and Max

Type of Units : None

Equation: 90+LEADANGLE

Last evaluated value: 135.00

Minimum Value: 1.0000E-06 Maximum Value: 180.00

The parameter controls the following entities:

Leaf 1, Extrusion Angular_Dimension_12

Parameter Name/Number: 8 - RETROT

Type of Parameter : Equational

Type of Limit : Min and Max

Type of Units : None

Equation: 180-RETANGLE

Last evaluated value: 135.00

Minimum Value: 1.0000E-06 Maximum Value: 180.00

The parameter controls the following entities:

Leaf 1, Extrusion Angular_Dimension_15

APPENDIX E
USER EVALUATION FORM

NAME _____

1. Does the expert system prototype match known design solutions? Give examples.

2. Discuss the data input. Is it self-explanatory? Is the format easy to use?
Do you have any suggestions for changes to the data input?

3. Does the prototype provide enough flexibility in altering the configuration geometry?
Should any other variables be modifiable?

4. Are the appropriate design constraints (strain and maximum mating and separating forces)considered in the prototype? Would you add any additional constraints?

5. Discuss the output? Is it descriptive? Is the format appropriate?

6. Was the level of instruction adequate?

7. For a non-programmer, do you think the system would be easy to use?

For the extended evaluator

8. Discuss the ease of adding knowledge to the prototype. How easy is it to add new materials or rules to the system? How easy is it to modify the user interface?

9. Compare the expert system software development tools to other packages that you are familiar with.

REFERENCES

H. Adeli, ed., *Expert Systems in Construction and Structural Engineering*, New York, NY: Chapman & Hall, 1988.

H. Adeli and Y. S. Chen, "Structuring Knowledge and Data Bases in Expert Systems for Integrated Structural Design," *Microcomputers in Civil Engineering*, vol. 4, no. 3, Sept. 1989, 175 - 199.

Hojjat Adeli, *Knowledge Engineering*, 2 vols, New York: McGraw-Hill, 1990.

R. E. Adler and K. Ishii, "DAISIE: Designer's Aid for Simultaneous Engineering," *ASME Computers in Engineering Conference*, vol. 1, 1989, 19 - 26.

Janice Aikins, John Kunz, and Edward H. Shortliffe, "PUFF: An Expert System for Interpretation of Pulmonary Function Data," *Computers and Biomedical Research*, vol. 16, 1983, 199 - 208.

R. H. Allen, "Design Guidelines for Expert Systems," *Proceedings of Conference on AI*, Springer-Verlag / Computational Mechanics Publications, 1986, 651 - 658.

R.H. Allen, M. G. Boarnet, C. J. Culbert, and R. T. Savely, "Using Hybrid Expert System Approaches for Engineering Applications," *Engineering with Computers*, vol. 2, 1987, 95-110.

Morris Asimow, *Introduction to Design*, Englewood Cliffs, NJ: Prentice Hall, 1962.

S. D. Bacon and D. C. Brown, "Reasoning about Mechanical Devices: A Top-Down Approach to Deriving Behavior from Structure," *ASME Computers in Engineering Conference*, vol. 1, 1988, 467 - 472.

David Barnett, Charles Jackson, and James A. Wentworth, "Developing Expert Systems," U. S. Department of Transportation Technical Report, 1988.

Avron Barr, Paul Cohen, and Edward Feigenbaum, *The Handbook of Artificial Intelligence*, Stanford, CA: HeurisTech Press, 1982.

V. Baya, N. A. Langrana, and Y. Jularia, "Design of a Die in an Extrusion Manufacturing Process," *ASME Computers in Engineering Conference*, vol. 1, 1989, 141 - 149.

Glen L. Beall, "Plastic Part Design for Economical Injection Molding," Society of the Plastics Industry Seminar Notes, 1990.

Claude Bedard and Krishnan Gowri, "Automating Building Design Process with KBES," *Computing in Civil Engineering*, vol. 4, no. 2, April 1990, 69 - 83.

- James S. Bennett and Robert S. Englemore, "SACON: A Knowledge-Based Consultant for Structural Analysis", *IJCAI*, 1979, 47 - 49.
- P. P. Bonissone and R. M. Tong, "Editorial: Reasoning with Uncertainty in Expert Systems", *International Journal of Man-Machine Studies*, vol. 22, no. 3, 1985, 241 - 250.
- Borg-Warner Chemicals. *Plastics Design Manual*. Parkersburg, WV, 1986 - 1988.
- Borg-Warner Chemicals. *Techniques*. Parkersburg, WV, 1986 - 1988.
- Ronald J. Brachman, "On the Epistemological Status of Semantic Networks," N. Findler, Ed., *Associative Networks: Representation and Use of Knowledge by Computers*, New York: Academic Press, 1979.
- M.A. Bramer, "A Survey and Critical Review of Expert Systems Research," *Introductory Readings in Expert Systems*, Donald Mitchie, ed., 1982, 3 - 29.
- M.A. Bramer, "Expert Systems: the Vision and the Reality," *Fourth Technical Conference of British Computer Society Specialist Group on Expert Systems / Research and Development in Expert Systems*, M.A. Bramer ed., 1984, 1 - 12.
- Alan Brody, "The Experts," *Infoworld*, June 19, 1989, 59 - 75.
- D. C. Brown and B. Chandrasekaran, "An Approach to Expert Systems for Mechanical Design," *Proceedings Trends and Applications*, 1983, 173 - 180.
- D. C. Brown and B. Chandrasekaran, "An Expert System for Mechanical Design: A Progress Report," *ASME Computers in Engineering Conference*, vol. 1, 1984, 343-344.
- D. C. Brown and B. Chandrasekaran, "Expert Systems for a Class of Mechanical Design Activity," *Knowledge Engineering in Computer-Aided Design IFIP Proceedings*, 1984, 259 - 277.
- David C. Brown and B. Chandrasekaran, "Knowledge and Control for a Mechanical Design Expert System," *Computer*, July 1986, 92 -100.
- D.C. Brown, "Capturing Mechanical Design Knowledge," *ASME Computers in Engineering Conference*, vol. 2, 1985, 121 - 129.
- David C. Brown, "Failure Handling in a Design Expert System," *Computer Aided Design*, vol. 17, no. 9, Nov. 1985, 436 - 441.
- D.C. Brown and W.N. Sloan, "Compilation of Design Knowledge for Routine Design Expert Systems: An Initial View," *ASME Computers in Engineering Conference*, vol. 1, 1987, 131 - 136.
- J.P. Brown, J.H. Clinton, and G.E. Nevill, "Managing Subproblem Interactions in Preliminary Mechanical Design," *ASME Computers in Engineering Conference*, vol. 1, 1989, 265 - 272.
- Bruce G. Buchanan and Edward A. Feigenbaum, "DENDRAL and Meta-DENDRAL: Their Applications Dimension," *Artificial Intelligence*, vol. 11, August 1978, 5-24.

- B. G. Buchanan, et al., "Constructing an Expert System," *Building Expert Systems*, F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, eds., 1983, 127- 168.
- Bruce G. Buchanan and Edward H. Shortliffe, *Rule Based Expert Systems*, Reading, MA: Addison-Wesley, 1984.
- Bruce G. Buchanan, "What Do Expert Systems Offer the Science of AI?" *Applications of Expert Systems*, vol. 2, J. Ross Quinlan, ed., 1989, 11 - 35.
- B. Chandrasekaran, "Towards a Taxonomy of Problem Solving Types," *AI Magazine*, Winter/Spring 1983, 9 - 17.
- B. Chandrasekaran, "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE Expert*, Fall 1986, 23 - 30.
- F.S. Chehayeb, J. J. Connor, and J. H. Slater, "An Environment for Building Engineering Knowledge Based Systems," *Applications of Knowledge Based Systems to Engineering Analysis and Design / Winter Annual Meeting ASME*, 1985, 9 - 28.
- J. C. H. Chung, R. L. Cook, D. Patel, and M. K. Simmons, "Feature-Based Geometry Construction for Geometric Reasoning," *ASME Computers in Engineering Conference*, vol. 1, 1988, 497 - 504.
- CIME Staff Report, "AI Pays Off in Flexible Design System," *Mechanical Engineering*, April 1989, 68 - 72.
- Jonathan S. Colton and John L. Dascanio, II, "An Integrated, Intelligent Design Environment," *Engineering with Computers*, vol. 7, 1991, 11 - 22.
- Michael J. Coombs, *Developments in Expert Systems*, Orlando, FL:Academic Press, 1984.
- Robert Cramer, Notes on Snap-Fit Module, 1987.
- J. J. Cunningham and J. R. Dixon, "Designing with Features: The Origin of Features," *ASME Computers in Engineering Conference*, vol. 1, 1989, 237 - 243.
- Randall Davis and Jonathon King, "An Overview of Production Systems," *Machine Intelligence*, vol. 8, 1977, 300 - 332.
- R. Davis, "Where Are We and Where Do We Go from Here?," *AI Magazine*, Spring 1982, 3 - 22.
- Michelle Dibble, "How to Get the Most from Plastics Technical Centers," *Machine Design*, Sept. 24, 1992, 58 - 64.
- J. R. Dixon and M. K. Simmons, "Expert Systems for Engineering Design: Standard V-Belt Design as an Example of the Design-Evaluate-Redesign Architecture," *ASME Computers in Engineering Conference*, vol. 1, 1984, 332 - 337.
- J. R. Dixon, M. K. Simmons, and P. R. Cohen, "An Architecture for Application of Artificial Intelligence to Design," *ACM / IEEE 21st Design Automation Conference*, 1984, 634 - 640.

J. R. Dixon and M. K. Simmons, "Expert Systems for Mechanical Design: A Program of Research," *ASME Design Automation Conference/Design Engineering Division*, 1985, 1-9.

J. R. Dixon, A. Howe, P.R. Cohen, and M. K. Simmons, "Dominic I: Progress Towards Domain Independence in Design by Iterative Redesign," *ASME Computers in Engineering Conference*, vol.1, 1986, 199-206.

John R. Dixon, "Artificial Intelligence and Design: A Mechanical Engineering View," *Fifth National Conference on Artificial Intelligence / AAAI Proceedings*, 1986, 872-877.

John R. Dixon, Eugen C. Libardi, Jr. , Steven C. Luby, Mohan Vaghul, and Melvin K. Simmons, "Expert Systems for Mechanical Design: Examples of Symbolic Representations of Design Geometries," *Engineering with Computers*, vol. 2, 1987, 1-10.

J. R. Dixon, M. R. Duffey, R. K. Irani, K. L. Meunier, and M. F. Orelup, "A Proposed Taxonomy of Mechanical Design Problems," *ASME Computers in Engineering Conference*, vol. 1, 1988, 41 - 55.

Richard Duda, John Gaschnig, and Peter Hart, "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," *Expert Systems in the Micro-electronic Age*, 1979, 153 - 167.

Richard Duda and John Gaschnig, "Knowledge-Based Expert Systems Come of Age," *BYTE*, vol. 6, no. 9, Sept. 1981, 238 - 281.

M. R. Duffey and J. R. Dixon, "Automating the Design of Extrusions: A Case Study in Geometric and Topological Reasoning for Mechanical Design," *ASME Computers in Engineering Conference*, vol. 1, 1988, 505 - 511.

E. I. duPont de Nemours & Co. (Inc.), Polymer Products Department. *Engineering Polymers Design Handbook*. Wilmington, DE.

Paul Dvorak, "Keeping Talent with Knowledge Systems," *Machine Design*, Aug. 22, 1991, 37 - 42.

Clive L. Dym, "EXPERT SYSTEMS: New Approaches to Computer-aided Engineering," *Engineering with Computers*, vol. 1, 1985, 9-25.

Clive L. Dym and Raymond E. Levitt, *Knowledge-Based Systems in Engineering*, New York, McGraw-Hill, 1991.

Clive L. Dym and Raymond E. Levitt, "Toward the Integration of Knowledge for Engineering Modeling and Computation," *Engineering with Computers*, vol. 7, 1991, 209 - 224.

Steven L. Elam and L. A. Lopez, *Knowledge Based Approach to Checking Designs for Conformance with Standards*, Ph.D. diss., U of Illinois at Urbana-Champaign, 1988.

L. D. Erman, F. Hayes-Roth, V. Lesser, and D. Reddy, "The HEARSAY-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, vol. 12, no. 2, June 1980, 213 - 253.

- A. Esterline, D. Rosen, K. Otto, L. Nelson, T. Hessburg, D.R. Riley, and A.G. Erdman, "A Methodology for Capturing Mechanical Design Expertise," *ASME Computers in Engineering Conference*, vol. 1, 1988, 47 - 55.
- P. Fazio, C. Bedard, and K. Gowri, "Knowledge-Based System Approach to Building Envelope Design," *Computer-Aided Design*, vol. 21, no. 8, October 1989, 519 - 527.
- Edward A. Feigenbaum, "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," *IJCAI 5*, 1977, 1014 - 1029.
- Edward A. Feigenbaum, Pamela McCorduck, and H. Penny Nii, *The Rise of the Expert Company*, New York: Times Books, 1988.
- E. A. Feigenbaum, "Expert Systems in the 1980s," *Machine Intelligence*, (Infotech State of the Art Report Series 9, No. 3), A. Bond, ed., 1981, 219 - 229.
- E. A. Feigenbaum and P. McCorduck, *The Fifth Generation*, Addison-Wesley, 1983.
- E. A. Feigenbaum, "Knowledge Processing: From File Servers to Knowledge Servers," *Applications of Expert Systems*, vol. 2, J. Ross Quinlan ed., 1989, 3 - 10.
- S. J. Fenves, "A Framework for Knowledge Based Finite Element Analysis Assistant," *Applications of Knowledge Based Systems to Engineering Analysis and Design / Winter Annual Meeting ASME*, 1985, 1-8.
- S. J. Fenves, "What is an Expert System?" *Expert Systems in Civil Engineering / Proceedings ASCE, Technical Council on Computer Practices*, 1986, 1-6.
- S. J. Fenves, U. Flemming, C. Hendrickson, M.L. Maher, and G. Schmitt, "Integrated Software Environment for Building Design and Construction," *Computer-Aided Design*, vol. 22, no. 1, Jan 1990, 27 - 36.
- Gavin A. Finn and Kenneth F. Reinschmidt, "Expert Systems in an Engineering-Construction Firm," *Expert Systems in Civil Engineering / Proceedings ASCE Technical Council on Computer Practices*, 1986, 40 - 54.
- Martin Fischer, "Linking CAD and Expert Systems for Constructability Reasoning," *Proceedings of 5th International Conference on Computing in Civil and Building Engineering*, 1993, 1563 - 1570.
- Bruce W.R. Forde, Alan D. Russell, and Siegfried F. Stierner, "Object-Oriented Knowledge Frameworks," *Engineering with Computers*, vol. 5, 1989, 79 - 89.
- Mark S. Fox, "AI and Expert System Myths, Legends, and Facts," *IEEE Expert*, Feb 1990, 8 -20.
- J. Gaschnig, "Prospector: An Expert System for Mineral Exploration," *Introductory Readings in Expert Systems*, Donald Mitchie, ed., 1982, 47 - 63.
- J. Gaschnig, R. Reboh, and J. Reiter, "Development of a Knowledge-Based Expert System for Water Resources Problems," SRI Project 1619, SRI International, August, 1981.

- Michael R. Genesereth, "The Role of Plans in Automated Consultation," *IJCAI*, 1979, 311 - 319.
- J. S. Gero, ed., *Applications of Artificial Intelligence in Engineering V*, vol. 1 Design, (Proceedings of the Fifth International Conference) Boston: Springer-Verlag, 1990.
- William B. Gevarter, "An Overview of Expert Systems," NBSIR 82-2505, May 1982.
- William B. Gevarter, "Expert Systems: Limited but Powerful," *IEEE Spectrum*, August, 1983, 39 - 45.
- William B. Gevarter, *Intelligent Machines: An Introductory Perspective of Artificial Intelligence and Robotics*, New Jersey: Prentice Hall, 1985.
- William B. Gevarter, "The Nature and Evaluation of Commercial Expert System Building Tools," *Computer*, vol. 20, no. 5, May 1987, 24 - 41.
- M. Maher Hakim and J. H. Garrett, "A Description Logic Approach for Representing Engineering Design Standards," *Engineering with Computers*, vol. 9, 1993, 108 - 124.
- Paul Harmon and David King, *Expert Systems: Artificial Intelligence in Business*, New York, NY: John Wiley & Sons, Inc, 1985.
- Anna Hart, "Knowledge Elicitation: Issues and Methods," *Computer Aided Design*, vol. 17, no. 9, Nov. 1985, 455 - 462.
- Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat, eds., *Building Expert Systems*, Reading, MA: Addison-Wesley Publishing Company, Inc, 1983.
- Hoechst Celanese Corp, Engineering Plastics Division. *Designing with Plastic*. Chatham, NJ, 1989.
- David A. Hoeltzel and Wei-Hua Chieng, "Factors that Affect Planning in a Knowledge-Based System for Mechanical Engineering Design Optimization with Application to the Design of Mechanical Power Transmissions," *Engineering with Computers*, vol. 5, 1989, 47-62.
- David Horn, "Expert Systems Emerge from Their Shells," *Mechanical Engineering*, April 1989, 64 - 67.
- H. Craig Howard and Daniel R. Rehak, "KADBASE: Interfacing Expert Systems with Databases," *IEEE Expert*, Fall 1989, 65 - 76.
- Guo Huang, Derek Sheldon, and Roger Perks, "Concurrent Engineering by Cooperating Expert Systems," *ASME Design for Manufacturability*, vol. 52, 1993, 51 - 56.
- Vladimir Hubka, *Principles of Engineering Design*, London, England: Butterworth Scientific, 1982.
- K. E. Hummel, "Coupling Rule-Based and Object-Oriented Programming for the Classification of Machined Features," *ASME Computers in Engineering Conference*, vol. 1, 1989, 409 - 418.

- V. Daniel Hunt, *Artificial Intelligence & Expert Systems Sourcebook*, New York, NY: Chapman & Hall, 1986.
- James P. Ignizio, *Introduction to Expert Systems*, New York: McGraw-Hill, 1991.
- Intellicorp, Inc. *Kappa PC Reference Manual*. 1990.
- Intellicorp, Inc. *Kappa PC User's Guide*. 1990.
- K. Ishii and P. Barkan, "Design Compatibility Analysis -- A Framework for Expert Systems in Mechanical System Design," *ASME Computers in Engineering Conference*, vol. 1, 1987, 95 - 102.
- K. Ishii, L. Hornberger, and M. Liou, "Compatibility-Based Design for Injection Molding," *Concurrent Product and Process Design / Winter Annual Meeting ASME*, 1989, 153 - 160.
- R. K. Irani, B. H. Kim, and J. R. Dixon, "Integrating CAE, Features, and Iterative Redesign to Automate the Design of Injection Molds," *ASME Computers in Engineering Conference*, vol. 1, 1989, 27 - 33.
- Peter Jackson, *Introduction to Expert Systems*, Workingham, England: Addison-Wesley Publishing Co., 1986.
- Taesik Jeong, Thomas Kicher, and Ronald Zab, "A Mechanical Design Framework Based on Object-Oriented Approach," *ASME Computers in Engineering Conference*, 1993, 315 - 324.
- Seiji Kameoka, Nobuhiro Haramoto, and Tadamoto Sakai, "Development of an Expert System for Injection Molding Operations," *Advances in Polymer Technology*, vol. 12, no. 4, 1993, 403 - 418.
- Taha Khedro, M. Genersereth, and P. Teicholz, "Agent-Based Framework for Integrated Facility Engineering," *Engineering with Computers*, vol. 9, 1993, 94 - 107.
- S.B. Kim and N.P. Suh, "Expert Design System for Injection Molding," *KBES for Manufacturing / Winter Annual Meeting ASME*, 1986, 311 - 325.
- F. Kinoglu, D. Riley, and M. Donath, "Knowledge-Based System Model of the Design Process," *ASME Computers in Engineering Conference*, vol. 1, 1986, 181 - 191.
- Michael L. Kmetz, *CAD/CAE of Piece Parts for a Specific Manufacturing Process*, Ph.D. diss., University of Wyoming, 1986.
- A. S. Kott and J. H. May, "Decomposition vs. Transformation: Case Studies of Two Models of the Design Process," *ASME Computers in Engineering Conference*, vol.1, 1989, 1 - 8.
- V.M. Kulkarni, J.R. Dixon, J.E. Sunderland, and M.K. Simmons, "Expert Systems for Design: The Design of Heat Fins as an Example of Conflicting Subgoals and the Use of Dependencies," *ASME Computers in Engineering Conference*, vol. 2, 1985, 145 - 150.

A. Senthil kumar, A.Y.C. Nee, and S. Prombanpong, "Expert Fixture-Design System for an Automated Manufacturing Environment," *Computer-Aided Design*, vol. 24, no. 6, June 1992, 316 - 326.

T. H. Kwon and P. A. Weeks, "Expert System Aid for Intelligent Molding Cooling System Design," *ASME Computers in Engineering Conference*, vol.1, 1988, 281 -286.

Alice LaPlante, "Bring in the Expert," *Infoworld*, Oct. 1, 1990, 55 - 64.

Jean-Claude Latombe, "Failure Processing in a System for Designing Complex Assemblies," *IJCAI*, 1979, 508 - 515.

H. Lee and T.H. Kwon, "Heuristic Redesign with Numerical Analysis Aids," *ASME Computers in Engineering Conference*, vol. 1, 1989, 131 - 140.

H. H. Lee and J. S. Arora, "Object-Oriented Programming for Engineering Applications," *Engineering with Computers*, vol. 7, 1991, 225 - 235.

K. S. Leung and M. H. Wong, "An Expert-System Shell Using Structured Knowledge," *Computer*, March 1990, 38 - 47.

L. A. Lopez, S. Elam, and K. Reed, "Software Concept for Checking Engineering Designs for Conformance with Codes and Standards," *Engineering with Computers*, vol. 5, 1989, 63 - 79.

S. C-Y. Lu, "Knowledge-Based Expert Systems: A New Horizon of Manufacturing Automation," *KBES for Manufacturing / Winter Annual Meeting ASME*, 1986, 11-23.

K.J. MacCallum and A. Duffy, "An Expert System for Preliminary Numerical Design Modelling," *Advances in Engineering Software*, vol. 8, no. 4, 1986, 217 - 222.

J. Mackerle, "A Review of Expert systems Development Tools," *Engineering Computations*, vol. 6, March 1989, 2 - 17.

M. L. Maher and S. J. Fenves, "HI-RISE: An Expert System for the Preliminary Structural Design of High Rise Buildings," *Knowledge Engineering in Computer-Aided Design / IFIP Proceedings*, 1984, 125 - 134.

Mary Lou Maher, "HI-RISE and Beyond: Directions for Expert Systems in Design," *Computer Aided Design*, Nov. 1985, 420-427.

Mary Lou Maher, "Problem Solving Using Expert System Techniques," *Expert Systems in Civil Engineering / Proceedings ASCE Technical Council on Computer Practices*, 1986, 7-17.

Mary Lou Maher, "Expert System Components," *Expert Systems for Civil Engineers: Technology and Application*, ASCE, Mary Lou Maher, ed., 1987, 3-14.

M. L. Maher, D. Sriram, and S. J. Fenves, "Tools and Techniques for Knowledge Based Expert Systems for Engineering Design," *Advances in Engineering Software*, vol. 6, no. 4, 1984, 178 - 188.

Rex Maus and Jessica Keyes, *Handbook of Expert Systems in Manufacturing*, New York: McGraw-Hill, 1991.

John McDermott, "R1: A Rule-based Configurer of Computer Systems," *Artificial Intelligence*, 19, 1982, 39 - 88.

K. L. Meunier and J. R. Dixon, "Iterative Respecification: a Computational Model for Hierarchical Mechanical System Design," *ASME Computers in Engineering Conference*, vol. 1, 1988, 125 - 32.

Donald Michie, "Expert Systems," *The Computer Journal*, vol. 23, no. 4, Nov. 1980, 369 - 376.

Donald Michie, ed., *Introductory Readings in Expert Systems*, Gordon and Breach Science Publishers, 1982.

Miles, *Plastic Snap-Fit Joints*, Pittsburgh: Miles Inc., Polymers Division, 1992.

Garth Miller and J. S. Colton, "The Complementary Roles of Expert Systems and Database Management Systems in a Design for Manufacture Environment," *Engineering with Computers*, vol. 8, 1992, 139 - 149.

M. Minsky, "A Framework for Representing Knowledge," *The Psychology of Computer Vision*, P. Winston, ed., McGraw Hill Book Company, 1975.

Sanjay Mittal and Agustin Araya, "A Knowledge-Based Framework for Design," *National Conference on Artificial Intelligence AAAI Proceedings*, 1986, 856 - 865.

S. Mittal, C. L. Dym, and M. Morjaria, "PRIDE: An Expert System for the Design of Paper Handling Systems," *Computer*, July 1986, 102 - 114.

M. Morjaria, S. Mittal, and C.L. Dym, "Interactive Graphics in Expert Systems for Engineering Applications," *ASME Computers in Engineering Conference*, vol. 2, 1985, 235 - 239.

J. Mostow, "Toward Better Models of the Design Process," *AI Magazine*, vol. 6, no. 1, Spring 1985, 44 - 57.

A. Newell, "Heuristic Programming: Ill-Structured Problems", *Progress in Operations Research, Vol. III*, Aronofsky, ed., 1969, 360 - 414.

Allen Newell and Herbert A. Simon, *Human Problem Solving*, New Jersey: Prentice Hall, 1972.

E.H. Nielsen, J.R. Dixon, and M.K. Simmons, "GERES: A Knowledge Based Material Selection Program for Injection Molded Resins," *ASME Computers in Engineering Conference*, 1986, 255 - 261.

E.H. Nielsen, J.R. Dixon, and G.E. Zinsmeister, "Capturing and Using Designer Intent in a Design-With-Features System," *ASME Design Theory and Methodology Conference*, 1991, 95 - 102.

H. Penny Nii, "The Blackboard Model of Problem-Solving," *AI Magazine*, Summer 1986, 38 - 53.

H. Penny Nii, "Blackboard Systems at the Architecture Level," *Expert Systems with Applications*, vol. 7, no. 1, Jan-Mar 1994, 43 - 54.

Nils J. Nilsson, *Principles of Artificial Intelligence*, Palo Alto, CA: Tioga Publishing Co., 1980.

Carl E. Noble, "Solving Ill-Structured Management Problems," *Business*, Jan-Feb 1979, 26 - 33.

Robert M. O'Keefe, Osman Balci, and Eric P. Smith, "Validating Expert System Performance," *IEEE Expert*, Winter 1987, 81 - 89.

G. Pahl and W. Beitz, *Engineering Design*, K. Wallace, ed., London, England: The Design Council, Springer-Verlag, 1984.

Dennis Pearce, "A Statistical/Heuristic Approach to Estimating Model Costs." *ANTEC* 1989, 364 - 366.

H. E. Pople, J. D. Myers, and R. A. Miller, "DIALOG: A Model of Diagnostic Logic for Internal Medicine," *IJCAI*, 1975, 848 - 855.

Harry E. Pople, "The Formation of Composite Hypotheses in Diagnostic Problem Solving / An Exercise in Synthetic Reasoning," *IJCAI*, 1977, 1030 - 1037.

M. Ross Quillian, "Semantic Memory," *Semantic Information Processing*, M. Minsky, ed., 1968, 216 - 270.

J.R. Quinlan, "Fundamentals of the Knowledge Engineering Problem," *Introductory Readings in Expert Systems*, Donald Michie, ed., 1982, 33 - 46.

J. R. Quinlan, "Inductive Knowledge Acquisition: A Case Study," *Applications of Expert Systems*, vol.1, J. Ross Quinlan, ed., 1987, 157 - 173.

N. Ramchandran, A. Shah, and N.A. Langrana, "Expert System Approach in Design of Mechanical Components," *ASME Computers in Engineering Conference*, vol. 1, 1988, 1-10.

Martin Ramsey, "Gaining Proficiency in Expert Systems," *Mechanical Engineering*, April 1989, 73 - 78.

R. H. Rand, *Computer Algebra in Applied Mathematics: An Introduction to MACSYMA. Research Notes in Mathematics, 94*, Boston: Pitman Publishing, 1984.

W. J. Rasdorf, "Perspectives on Knowledge in Engineering Design," *ASME Computers in Engineering Conference*, vol. 2, 1985, 249 - 253.

B. Ravi and M.N. Srinivasan, "Decision Criteria for Computer-Aided Parting Surface Design," *Computer Aided Design*, vol. 22, no. 1, Jan./Feb. 1990, 11 - 18.

- Daniel R. Rehak and H. Craig Howard, "Interfacing Expert Systems with Design Databases in Integrated CAD Systems," *Computer Aided Design*, Nov. 1985, 443 - 454.
- Dave Reiff, "Integral Fastener Design," *Plastics Design Forum*, Sept/Oct. 1991, 59 - 63.
- D. Rosen, A. Erdman, and D. Riley, "A General Design Knowledge-Based System Shell, with Application to Dwell Mechanism Design," *ASME Computers in Engineering Conference*, vol. 1, 1987, 29 - 36.
- David Rosen, John R. Dixon, Corrado Poli, and Xin Dong, "Features and Algorithms for Tooling Cost Evaluation in Injection Moulding and Die Casting," *ASME Computers in Engineering Conference*, vol. 1, 1992, 45 - 52.
- Michael Rosenman and John Gero, "Design Codes as Expert Systems," *Computer-Aided Design*, vol. 17, no. 9, Nov 1985, 399 - 409.
- M. D. Rychener, "Expert Systems for Engineering Design," *Proceedings Trends and Applications*, 1983, 21 - 27.
- G. Rzevski, ed., *Applications of Artificial Intelligence in Engineering V*, vol. 2 Manufacturing and Planning, (Proceedings of the Fifth International Conference) Boston: Springer-Verlag, 1990.
- T.S. Sakhivel and V. Kalyanaraman, "A KBES for Integrated Engineering," *Engineering with Computers*, vol. 9, 1993, 1 - 16.
- Mukul Saxena and Rohinton Irani, "Knowledge-Based Parametric Modeling for Nozzles," *ASME Computers in Engineering Conference*, 1993, 385 - 395.
- Mukul Saxena and Rohinton Irani, "An Integrated NMT-Based CAE Environment -- Part II: Applications to Automated Gating Plan Synthesis for Injection Molding," *Engineering with Computers*, vol. 9, 1993, 220 - 230.
- Peter M. Schoonmaker, "The Best Laid Plans: Troubleshooting an Expert System," *Mechanical Engineering*, Dec. 1989, 56 - 58.
- J. J. Shah, "Development of a Knowledge Base for an Expert System for Design of Structural Parts," *ASME Computers in Engineering Conference*, vol. 2, 1985, 131 - 136.
- Aroon Shenoy, "Expert Systems in Plastics Processing," *Materials Engineering*, Nov. 1988, 33 - 36.
- Edward H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, New York: American Elsevier/North Holland, 1976.
- D. Sriram, "Computer-Aided Engineering: The Knowledge Frontier," Course Notes, MIT, 1988.
- D. Sriram, M. L. Maher, S. J. Fenves, "Knowledge-Based Expert Systems in Structural Design," *Computers and Structures*, vol. 20, no. 1-3, 1985, 1-9.

D. Sriram et al., "Knowledge-Based System Applications in Engineering Design: Research at MIT," *AI Magazine*, Fall 1989, 79 - 96.

Sally Steadman, "An Integrated Expert System for Engineering Design," *6th International Conference on Artificial Intelligence and Expert Systems* (In press).

Sally Steadman and Kynric M. Pell, "Expert Systems in Engineering Design: An Application for Injection Molding of Plastic Parts," *The Journal of Intelligent Manufacturing* (In press).

Luc Steels, "Components of Expertise," *AI Magazine*, Summer 1990, 28 - 49.

M. Stefik, J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti, "The Organization of Expert Systems: A Tutorial," *Artificial Intelligence*, 18(2), March 1982, 135 - 173.

Mark Stefik and Danial G. Bobrow, "Object-Oriented Programming: Themes and Variations," *AI Magazine*, Winter 1985, 40 - 62.

J. Stutz and R.L. Kashyap, "Improving Variant Design of Mechanical Systems through Functional Relationships," *ASME Computers in Engineering Conference*, vol. 1, 1989, 151 - 159.

Anthony Stylianou, Gregory Madey, and Robert Smith, "Selection Criteria for Expert System Shells: A Socio-Technical Framework," *Communications of the ACM*, vol. 35, no. 10, Oct 1992, 30 - 48.

R.P. Ten Dyke and J. C. Kunz, "Object-oriented Programming," *IBM Systems Journal* vol. 28, no. 3, 1989, 465 - 478.

John V. Thomson, "A Water Penetration Expert System using PROLOG with Graphics," *Applications of Expert Systems*, vol. 1, J. Ross Quinlan, ed., 1987, 48 - 65.

Deborah L. Thurston, "Concurrent Engineering in an Expert System," *IEEE Transactions on Engineering Management*, vol. 40, no. 2, May 1993, 124 - 135.

David G. Ullman, *The Mechanical Design Process*, New York: McGraw-Hil, Inc., 1992.

David G. Ullman and Thomas A. Dietterich, "Mechanical Design Methodology: Implications on Future Developments of Computer-Aided Design and Knowledge-Based Systems," *Engineering with Computers*, vol. 2, 1987, 21-29.

J.R. Umaretiya and S.P. Joshi, "An Insight into the Expert-Seed: A Knowledge Based System for Structural Design," *Engineering with Computers*, vol. 8, 1992, 151 - 161.

M. Vaghul, J. R. Dixon, G.E. Zinsmeister, and M. K. Simmons, "Expert Systems in a CAD Environment: Injection Molding Part Design as an Example," *ASME Computers in Engineering Conference*, vol. 2, 1985, 77 - 82.

J. van Koppen, "A Survey of Expert System Development Tools," *Expert Systems in Engineering*, D.T. Pham, ed., Springer-Verlag, 1988, 43 - 57.

- R.J. Verrilli, K. L. Meunier, J.R. Dixon, and M.K. Simmons, "Iterative Respecificaiton Management: A Model for Problem-Solving Networks in Mechanical Design," *ASME Computers in Engineering Conference*, vol. 1, 1987, 103 - 112.
- Donald A. Waterman, *A Guide to Expert Systems*, Reading, MA: Addison-Wesley, 1986.
- Shalom M. Weiss and Casimir A. Kulikowski, *A Practical Guide to Designing Expert Systems*, Totowa, NJ: Rowman & Allanheld, 1984.
- Patrick H. Winston, *Artificial Intelligence*, Reading, MA: Addison-Wesley, 1984.
- Patrick H. Winston, "The Commercial Debut of Artificial Intelligence," *Applications of Expert Systems*, vol.1, J. Ross Quinlan, ed., 1987, 3-20.
- Nobuyoshi Yabuki and K. Law, "An Object-Logic Model for the Representation and Processing of Design Standards," *Engineering with Computers*, vol. 9, 1993, 133 - 159.
- Jyh-Cheng Yu, Sherveen Lotfi, Kos Isii, and Andrew Trageser, "Process Selection for the Design of Aluminum Components," *ASME Computers in Engineering Conference*, 1993, 181 - 188.
- Zhentao Zhang and S. L. Rice, "An Expert System for Conceptual Mechanical Design," *ASME Computers in Engineering Conference*, vol. 1, 1989, 281 - 285.
- Zhentao Zhang and Stephen L. Rice, "Conceptual Design: Perceiving the Pattern," *Mechanical Engineering*, July 1989, 58 - 60.
- J.R. Zumsteg, D. Pecora, and V.J. Pecora, "A Prototype Expert System for the Design and Analysis of Composite Material Structures," *ASME Computers in Engineering Conference*, vol. 2, 1985, 137 - 143.
- J. R. Zumsteg and D. L. Flagg, "Knowledge-Based Analysis and Design Systems for Aerospace Structures," *Applications of Knowledge Based Systems to Engineering Analysis and Design / Winter Annual Meeting ASME*, 1985, 67-79.