UTILIZATION OF DATABASES AND EXPERT SYSTEMS IN THE DESIGN PROCESS

by

G. BENTON GIBBS

B.S., Kansas State University, 1985

A MASTERS REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTERS OF SCIENCE

College of Agriculture

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1987

Approved by: Stanley J. Clark
Stanley J. Clark, Major Professor

LD
2668
.4
GE
1981
.S52
c. 2

A11207 303117

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank the members of my graduate committee, Dr. Gene Grosh, Dr. Dennis Kuhlman, Dr. Mark Schrock, and especially my major professor, Dr. Stanley Clark, for the encouragement, support, and the freedom that they gave me in the pursuit of my masters degree.

Most of all, for their support, both financially and morally, I wish to thank my family and my wife, Rebecca for encouraging me to get the "thing" done.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

As the computer industry moves towards the next century, the advancements continue along unabated. Not many years ago, all the designer had in his toolkit was pencils, paper, ruler and a slide rule. Now, regardless of the discipline, the computer has become the principal tool for completing the design task.

The computer has relieved the designer of many of the mundane, tedious, error-prone jobs, therefore reducing the amount of time required to complete the design process. Engineers can now focus their concentration on the more creative aspects of design. The use of the computer also allows the engineer to evaluate further concepts quickly and efficiently.

The building and testing of prototypes is considered by many to be the most time consuming aspect of engineering. This function is extremely susceptible to improvements in productivity through the use of computer automation. In mechanical design, prototypes can be tested by generating a 3-Dimensional model of the design. Some of the modeling techniques available to the engineer include solids modeling and finite element analysis. In solids modeling, the engineer receives a more realistic view of the design. In finite element analysis, the model is created, whether it is a crankshaft, piston, or the entire engine, and the simula-tion is begun to evaluate its performance. An application, for example, lets the simulation calculate the breaking points and stress areas.

Research and testing has been carried out in the area of Artificial Intelligence with respect to using Knowledge Based Systems as a "intelligent front-end" to large design databases. Some of this work dates back to the late 1960's and early 1970's, but as it turned out, practical advances were severely limited by a lack of hardware powerful enough to test the theories with large-scale databases. Due to ever increasing advancements in computer technology, it is now possible to access and implement large databases of design criterion efficiently. With Knowledge Based Systems we can interface with the databases to provide knowledge-assisted design schemes. The implementation of Knowledge Based Systems and Database Management Systems is referred to as Knowledge Based Management Systems.

The development and testing of Knowledge Based Management Systems has been done more extensively in Australia and Great Britain. In Holland, Knowledge Base Systems have been incorporated into a design database that aids the engineer in the preliminary design of high rise buildings. In the United States, Digital Equipment Corporation has begun to use Knowledge Based Management Systems in the configuration of it's VAX line of computers. Promising results have been obtained from the use of Knowledge Based Systems and Database Management Systems, but development of these products are still in the infancy stage.

The purpose of this study was to establish and identify developers of Knowledge Based Management Systems. This work is to determine how Knowledge Based Systems are utilized with design databases in obtaining Knowledge Based Management Systems. In particular, the study is to address three major inquiries:

1. Type of inference engine used in the Knowledge Based Management System.
2. Type of search method used in the Knowledge Based Management System.
3. The knowledge representation method used in the Knowledge Based Management System.

LITERATURE REVIEW

Background

The evolution of Knowledge Based Systems has brought about a means of automating the answer to problems that have resisted being formalized as algorithms. Maher (1985) stated that Knowledge Based Systems have developed into practical problem-solving tools that can reach a level of performance comparable to that of a human expert in some specific way.

Recently there has been an increasing amount of research on design or design related problem-solving systems. This work includes Birmingham and Siewiorek (1984), Grinberg (1980), Dixon et al (1984), Bowen (1985), Kowalski and Thomas (1983), Latombe (1979), McDermott (1982), Mitchell (1983) and Sussman (1977). Much of the work in this area has been concentrated in electronics, but efforts are now turning to research on design in other areas.

The design activity in general has many components; such as problem definition, planning, the use of prestored plans, and the refinement of descriptions and the use of large amounts of knowledge. The work that is the focus of this study is concerned with the last activity, which requires that at every stage of the design the designer knows both what sequences of design steps are appropriate and also what knowledge is required.

The complete design process using a knowledge based system proceeds by first obtaining and checking the requirements. A rough design is rendered to establish whether a full design is worth pursuing. If the rough design proves successful, then the full design is attempted by requesting a design from the topmost *specialist*. A *specialist* is considered to be a concept about some subproblem of the design. Each specialist may select from its own set of plans.

Selection depends on the current state of the design. Each plan is a sequence of design actions. An action may be a request to attempt portions of the design using another specialist lower in the hierarchy.

This theory has been used to construct an expert problem-solver for the design of a type of air cylinder. The system, called AIR-CYL, has been reported in Brown (1985) and Brown and Chandrasekaran (1985). In this system, it only takes a few minutes to design an air cylinder that involves about 120 design decisions.

Matthews and Swift (1983) reported on the work preformed at Hull University in England, in 1981. Work had begun there on a computer-based consultation system in design for economic manufacture. The researchers realized the problem that is facing the designer, if the engineer is to design for economic manufacture and optimum functionality, the engineer needs to assimilate information of considerable breadth and complexity and have the necessary experience and judgmental skills to make the correct design decisions from a range of possibilities. A further difficulty is that, in general, the assessments are so complex and diffuse that they defy the mathematical treatments that have been developed in the domains of engineering science.

The idea considered by the Hull University team is that the task of "design for economic manufacture" (DEM) might be amenable to solution by using computers operating on artificial intelligence principles. The questions that researchers raised were: can knowledge based expert systems be used to solve *open-ended* design problems and can such systems be successfully applied in industry? It was found that these systems offer the potential to explain the line of reasoning behind their deductions. Knowledge based systems can also offer useful advice in open-ended and inexact domains. These systems can acquire knowledge, and be made readily usable by a design or production engineer who is not familiar with computing.

Rehak and Howard (1985) reported the importance of knowledge based systems on future integrated computer-aided design (CAD) systems. Flexibility and adaptability are of prime importance in incorporating such components into a CAD system. They placed emphasis on a flexible interface in which multiple knowledge based systems and multiple design databases communicate as independent, self-descriptive components within an integrated CAD system composed of heterogeneous components operating in a distributed computing environment.

Database Systems

A Database Management System (DBMS) is a software tool for developing applications that require access to shared information. A Database Management System is needed and used when multiple users or application programs require access to the same collection of data or knowledge. A separate software facility is required between the users and the database to protect the shared information. This type of facility provides security control, consistency control, recovery control, and concurrency control.

A Database Management System provides these functions via a high-level transaction specification language tailored to fit that particular system. Users generate transactions in this language and the Database Management System assumes the responsibility for managing physical data access, interleaving of overlapped operations, recovery from system failures, and managing access rights. Smith (1984) stated that to insulate users from the details of the physical data representations, the Database Management System should provide a special knowledge representation language called a *data model*.

As more and more data is stored in machine-readable formats, engineers will increasingly access computer databases for the technical information they currently find in books and reports. This is encouraged by the availability of powerful and inexpensive computers (e.g., personal computers); individuals, trade associations, professional societies, government agencies, and private concerns are beginning to offer various machine-readable databases to the engineering community. The databases that are being developed generally fall into two categories: databases that consist primarily of text (bibliographic), and databases that consist of numbers (numeric).

The American Society for Metals began computerizing bibliographic information on metals and other related engineering materials in 1966 when *Metadex*, the machine-readable version of the published *Metals Abstract*, was developed. This database was expanded in 1974 when material from *Alloy Index* was added. The Metadex database is a computerized bibliographic index of the worlds literature on metals and related metallurgical processes. The Metadex database consists of more than 600,000 abstracts; approximately 3,500 new abstracts are added monthly. Table A.1 in the appendices contains a list of currently available databases used in engineering applications.

Expert Systems/Knowledge Based Systems

Knowledge based systems cover the area of human ability concerned with the problem solving and application of expertise. They are the computer systems that blend knowledge and inference or reasoning procedures to solve problems that are normally handled by experts. Expert knowledge is a collection of policies, procedures, and methods used by human experts in solving a certain domain of problems. A knowledge based system (KBS) is an application system built around a direct representation of such expert knowledge.

There are generally three main components involved with knowledge based systems: the user interface, the inference mechanism or engine, and the knowledge base. The user interface allows the user to interact with the system to present the problem and see the conclusions. The inference engine, or reasoning mechanism, is similar to the control structure in a conventional program; it operates deductively and selects the relevant knowledge to reach a conclusion. This allows the system to answer users' queries even when the answer is not explicitly stored in the knowledge base. The knowledge base is perhaps the most important component as it contains the experts' knowledge and expertise.

Knowledge based systems represent expert knowledge generally in diminutive pieces; as logic rules, production rules, frames, and scripts. Pure logic rules have very simple semantics and are effective for representing knowledge that does not involve *side-effects* (influence from outside data). The following example shows an example of a pure logic rule system:

Given the statement: *FeS is a sulfide, it is a dark-gray compound and it is brittle.*

Produces in logic rules:

sulfide(FeS) && compound(FeS) && darkgray(FeS) && brittle(FeS)

where the characters && in logic represents a conjunctive *and*.

Production rules are similar to logic rules in form and are necessary to handle side effects. For example, the following example depicts the use of production rules:

Given the statement: *If X works in department Y, and Z is the manager of Y, then Z is the boss of X.*

Produces a production rule:

work_in(X,Y) && manager(z,y) => boss_of(z,x)

This form separates itself from a pure logic rule by the fact that production rules need additional information from the outside world. Frames are well suited to clustering rules based on their applicability at different stages of the problem solving process. Researchers in the field of Artificial Intelligence have different ideas concerning what a frame actually is, but basically, a frame is a structure of data that includes declarative or procedural information in predefined internal relationship. Frames often are used to describe a collection of attributes that a given object, such as a hydraulic cylinder, normally possesses. In a frame-like language such as Knowledge Representation Language (KRL) an equipment-frame might look like this:

```
(cylinder (a-kind-of (value hydraulic))
        (length     (value 24))
        (cyl_bore   (value 3.00))
        (rod_dia    (value 1.25))
        (sys_psi    (default 3000))
        (cycle_time (value 4.59))
        (hyd_hp     (value 13.99)))
```

Scripts like frames provide this same type of clustering of data and knowledge. Scripts are used to describe common sequences of events, such as what happens when one goes into a restaurant. The following script depicts this event:

```
EAT_AT_RESTAURANT Script
    Props:             (Restaurant, Money, Food, Menu, Tables, Chairs)
    Roles:             (Hungry-Persons, Wait-Persons, Chef-Person)
    Point-of-View:     (Hungry-Persons)
    Time-of-Occurrence: (Times-of-Operation of Restaurant)
    Place-of-Occurrence: (Location of Restaurant)
```

```
Event-Sequence:
       first:  Enter-Restaurant Script
       then:   if (Wait-To-Be-Seated-Sign or Reservations)
                  then Get-Maitre-d's-Attention Script
       then:   Please-Be-Seated Script
       then:   Order-Food-Script
       then:   Eat-Food-Script unless (Long-Wait)
                  then Exit-Restaurant-Angry Script
       then:   if (Food-Quality was better than Palatable)
                  then Compliments-To-The-Chef Script
       then:   Pay-For-It-Script
     finally:  Leave-Restaurant Script
```

This is an example extracted from Barr and Feigenbaum (1981) and is a rough rendition in English of the type of Restaurant script described by Schank and Abelson (1977). The script specifies a normal or default sequence of events as well as exceptions and possible error situations. The script also requires the use of a few static descriptions such as Props and Roles that refer to other frames.

In the field of Artificial Intelligence, frames are the most widely used primitives for Knowledge Representation (Minsky 1975). Although frames were widely conceived independently of the object-oriented paradigm they were in fact consistent with it, and provide an excellent demonstration of its power and flexibility. Frames are capable of representing both specific and general knowledge, and of accommodating both descriptive and prescriptive computations. In a frame system, the properties of both specific objects and generic objects (classes) are described by their *slots*, which may contain references to other frames (defining their relationships), actual values or procedural attachments to compute them.

One major difference between knowledge based systems and conventional programs is the separation of the expert knowledge (the rules) from the general reasoning mechanism. Knowledge based systems cover the area of human ability concerned with problem solving and applying expertise. Knowledge based systems are computer systems that use knowledge and inference which determines how the system interacts with the users.

Knowledge Based Management Systems

Knowledge Based Management Systems (KBMS) can be defined as a tool for developing custom applications requiring both a Database Management System and one or more Expert Systems/Knowledge Based Systems. A Knowledge Based Management System can be defined as "a system for developing applications requiring knowledge-assisted processing of information in a specific domain." Brodie (1986) defined knowledge base management systems as "A system providing highly efficient management of large, shared knowledge bases for knowledge-directed systems".

The necessity of integrating Artificial Intelligence and database technologies has been widely recognized, particularly in projects that build tools for developing knowledge-directed applications. Table A.2 in the appendices contains a list of such tools and projects. Although all projects, claim some form of database, most of them use simple file system concepts.

There are three different components that exist between the user and the information that is stored in the database: Knowledge Based Systems for the knowledge-assisted processing, unique processors for the handling of formatted data, and a Database Management System. There needs to be the distinction made between database management systems and knowledge based management systems. A database consists of mathematical structures together with a computational theory that states how the structures can be implemented efficiently. A knowledge base contains symbolic structures with a notion of interpretation of a specific subject matter. The different emphasis on computation versus interpretation has always distinguished database and AI research.

Approaches to integrating AI and Database technology must accommodate distinctions between them. Many distinctions have been observed: semantic versus computational theories of information, intentional versus extensional statements, complex versus simple statements, general versus specific statements, complex versus simple update semantics, propositional versus quantified logic, and proof theory versus model theory. The distinctions are useful if they suggest how to integrate AI and Database technologies or how to design knowledge base management systems. A knowledge based management system is intended to be a high-level tool that provides knowledge management for knowledge-directed applications, just as a database management system provides data management for data intensive applications.

The key research issues concern how to integrate the selected AI and database management systems functions. The most challenging issues fall according to Brodie (1986) into two levels: the knowledge level and the computational level. The knowledge level requires an integration of AI knowledge representation concepts in order to meet requirements for expressive power, reasoning, and truth management, with concepts from database data models if it is to meet requirements for data/knowledge management. This research should take advantage of the common interest in what the database area calls *semantic integrity constraints.*

The central issues of research in the conceptual level are concerned with performance. Existing processing techniques must be applied and new techniques must be developed to efficiently support the function of knowledge based management systems. Currently optimal database search techniques are being extended to include recursion. This leads to the fact that the database management system must perform all the theoretic deduction in the knowledge based management system. Brodie (1986) further stated that it is an open issue whether *proof theoretic deduction* can be handled by the knowledge-directed systems being supported, by the knowledge based management system, or through some cooperation between the two. Clearly, database research can bring a lot to research at the computational level.

Present Applications using Knowledge Based Management Systems

DACON Expert System

The design for assembly consultation system (DACON 1981) is aimed at tackling the problems of assembly rationalization, component optimization for handling and assembly processes, selection of a target assembly system and the optimum level of automation. The costs that are likely to accrue from a variety of systems are reported to illustrate DACON's decisions. DACON uses a depth-first search method of determining it's decision process. This means that as the dialogue progresses, if evidence obtained rules out one hypothesis the system continues by trying to backtrack to prove another goal. When DACON has drawn a conclusion the system will display a rule chain it has used to reach the deduction.

Having used it's knowledge base to diagnose assembly problems and make decisions on design modifications, the system can be used to compute assembly costs. As mathematical models can easily be applied to cost estimating, this element of DACON is a conventional program. The knowledge based elements are used where the decision making defies scientific treatment. Having evaluated costs, DACON displays handling and assembly process costs for all the components in the product. The DACON knowledge base has been built up from published material and case studies and, most importantly, by discussions with domain experts; it currently embodies 120 rules. However it is only by applying the system on industrial problems that its value can be fully judged. Work is also being carried out to determine how well DACON performs, compared to a human expert, when used by a designer of lesser expertise. Applications of DACON to assembly redesign studies, (Ellison and Boothroyd 1980) have shown that it can produce at least as good a result as the teams involved in these studies. The following is a sample rule used in the DACON system.

IF: 1. Part will tangle with other parts when in bulk supply
2. A force and a manipulation is required to separate the parts.
THEN: 1. There is strong evidence (.96) that the part requires manual feeding.
2. Consider redesign options.

AIR-CYL Expert System

Brown and Chandrasekaran (1985) reported on a knowledge based system that they developed for a company that manufactures air cylinders. The air cylinders had to be redesigned for each customer. This was done to account for the particular space it had to fit in or the intended operating temperatures and pressures. The AIR-CYL design problem-solving system was developed with the task-level Design Specialists and Plans Language (DSPL), that was developed by Brown and Chandrasekaran using the Rutgers Elisp language on a DECsystem-20;

which currently involves approximately 120 decision rules. DSPL is a language specifically tailored for writing design problem-solvers of this type.

AIR-CYL is unique in the fact that it incorporates the use of failure handling of the system (Brown 1985). Each design phase or specialist in the system should be able to detect their own failure, and be able to determine what went wrong and then attempt to correct the problem. As mentioned previously, a specialist is a concept or module that is used in the system. In the AIR-CYL system, the top-most specialist is responsible for the whole design. Specialists lower down in the hierarchy make the more detailed decisions. Every specialist has some local design knowledge, some of which is expressed as constraints. The constraints capture the major things that must be true of a specialist's design before it can be considered successfully completed. Other constraints, embedded in the specialist's plans, check the correctness of intermediate design decisions and check the compatibility of sub-problem solutions.

HI-RISE Expert System

HI-RISE is an knowledge based system that configures and evaluates several alternative structural systems for a given three dimensional grid. The expertise in HI-RISE is derived primarily from a recent book on preliminary structural design containing approximate analysis techniques and applicable design heuristics (Maher 1985).

HI-RISE addresses the preliminary structural design phase which involves the selection of a feasible structural configuration satisfying a few key constraints. HI-RISE divides the preliminary design process into two major components; each component addresses the design of a functional system. The functional systems are designed in a fixed order; first the lateral load resisting system is designed, followed by the design of the gravity load resisting system. After the previous functional system design has successfully completed, only then will the design of a new functional system be started. In the above mentioned component order, results from the design of the gravity system, namely, the depth, type, and weight of the floor system, are needed for the design of the lateral system.

Each of the two major components are broken down into a set of similar sub-components. The sub-components have the same goals for each functional system; however, the details of reaching these goals differ. For example, the first sub-component uses a depth-first search through the knowledge base to select a set of alternatives for the functional system under consideration. The purpose of the analysis sub-component or specialist is to evaluate the feasibility of an alternative and to define its component groups.

On completion, HI-RISE presents the user with all the structurally feasible systems that were developed and indicates which system was determined to be the "best", selected as the system with the minimum value assigned by the *system evaluation function.* The user has the option of accepting the recommended design or selecting one of the other structurally feasible systems that HI-RISE produced.

CARTER Expert System

Reynier and Fouet (1984) reported that CARTER is an expert system that designs crankcases without any intervention from the user (other than setting the problem by describing the mechanism to be encased). It uses several algorithmic specialists or modules such as hamiltonian path in a graph, plate theory, and finite elements.

As inputs to the system, the user describes the shafts, giving the length and diameters of each cylindrical or conical subsection. The user then describes the gears and bearings supported by these shafts, the torques and power. Many additional indications can be supplied, such as clamping points, lubrication, boundary values of strains, precisions, security ratios, manufacturing method and heat sources.

To model the huge amounts of knowledge required, Reynier and Fouet (1984) felt that conventional programming languages - however sophisticated-were completely inaccurate. The knowledge needed was sometimes quite fuzzy, and had to be constantly updated. Production rules were formed and a pattern-directed inference system was built (Watterman and Hayes-Roth 1978). This type of system that uses production rules, is a system that needs to be told what to do, and not how to do it. For example, production rules follow the following syntax:

$$If\ P_1\ and\ P_2\ and\ ...and\ P_n\ Then\ P'_1\ ;\ P'_2\ ;\ ...;\ P'_m$$

where the P_j are predicates and the P'_i are either predicates or calls to routines provided by the user. An example of a rule used in the Carter System is as follows:

IF SEARCH IS INSIDE THE ZONE, AND THERE ARE TWO BEARINGS IN THE ZONE, AND ONE MAY WORK ON THIS SIDE, AND IF THERE IS ONLY ONE SHAFT IN THE ZONE, AND IF OVERCROWDING ALLOW AXIAL CONNECTING OF THE HOUSINGS OF THE ZONE, AND IF MOUNTING OF THE SHAFT IS 'O-LIKE', THEN BUILD A SHARED

HOUSING IN THE ZONE.

In the previous example, BUILD... is not a call to a routine, but a predicate (BUILD-SHARED-HOUSING) of the variable ZONE that comes true, so that rules testing it will now *fire*. The fewer the calls to routines there are, the better, since the inference engine cannot organize its work efficiently if there are too many specialists or modules. There are currently approximately 300 such rules in the Carter System, dealing mainly with:

1) technological options
2) shapes and thicknesses
3) tools and costs
4) Setting the hypotheses for mechanical computings
5) piloting these computings

The system that runs this knowledge base is called GOSSEYN, was developed in the same manner as EMYCIN (Van Melle 1980) with some enhancements which have been discussed at length by Fouet (1982). The system has two primary components; a rule compiler for reading rules, checking for syntactical errors, build up the network of pointers from variables to the predicates containing them, and then pointers from the predicates to the rules. The second component is the inference engine; this is the pattern matcher that explores the short term memory to find facts that will satisfy all the conditions of the rule.

It was found that development of design tools incorporating knowledge based systems and databases is still in the infancy stage. A majority of the work in this area has dealt with electronics, mainly in the Very Large Scale Integration (VLSI) area. Subrahmanyam (1986) reported on SYNAPSE, which is an expert system developed at Bell Laboratories for VLSI design. The distinguishing features of the SYNAPSE architecture include a cohesive formal algebraic framework that supports input specifications at a very high abstraction level, the use of conventional knowledge based system tools and special-purpose theorem provers, and accommodation of machine learning techniques, including rote learning. SYNAPSE is a means to explore language and expert systems issues in VLSI design, and is part of a long-term research project. Future work will be directed at enriching the knowledge base of the system and augmenting its learning capabilities.

Although work on knowledge based systems incorporating databases towards design processes has been limited in the field of electronics, work in the mechanical design area is beginning to take form. This can be credited to the fact that more and more engineers are making use of present computer technology.

INVESTIGATION

Objectives

The objectives of this investigation were:

1. To define and critique knowledge based systems
2. To establish and identify developers and users of three Knowledge Based Management Systems.

During the critique of the three knowledge based systems, the following topic areas will be covered:

* Heuristic Search Method
* Computer language(s) used for development of the system
* Number of rules presently in system
* How external databases are interfaced to system
* What stage of development is system currently in

Background

Today many of the human mental activities such as solving mathematical problems, writing computer programs, design, understanding natural language are said to require some sort of *intelligence*. Over the past few decades, several researchers have developed computer systems that can perform tasks such as these. These systems have the capability of diagnosing diseases, discover mineral deposits, understand limited amounts of human speech and natural language text, and solve complex equations in symbolic form. We might say that such systems possess some degree of *artificial intelligence*.

Most of the work on building these kinds of systems has taken place in the field called Artificial Intelligence (AI), a sub-branch of the Computer Science field. This work has had largely an empirical and engineering orientation. Drawing from a loosely structured, but growing body of computational techniques. AI systems are developed, undergo experimentation, and are improved. This process has produced and refined several general AI principles of wide applicability.

The emergence of Knowledge Based Expert Systems provides a means for the engineer to use the computer as an aid in the solution of ill-structured problems. Knowledge based expert systems are interactive computer programs that incorporate the knowledge and judgement of experts in appropriate domains. The development of a knowledge based system presently involves the cooperative effort between one or more experts who possess the *domain-dependent* knowledge and a knowledge engineer. A knowledge engineer elicits the knowledge and uses either an expert system building tool or a general purpose language to represent and manipulate it. The representation of knowledge in a knowledge based system is dependent on the selection of the tool or language to be used. The knowledge engineer must make a choice among several available tools before embarking into a major developmental task; the ease of building a knowledge based system depends in part on the choice of the tool. This report will cover the structures and techniques used in knowledge based system design and will discuss their applicability to engineering design. The nature of engineering design will be briefly presented in the next section, which will be followed by an introduction to knowledge based system structures and techniques. In the latter sections three domain independent tools used in design will be described.

Overview of Engineering Design

Preiss (1980) defined design as the process in which an idea is developed, refined and elaborated into the detailed instructions for manufacturing. The design process may be divided into phases, where each phase may be handled by different individuals from different disciplines. The design process starts by a recognition of a need. This may come from different sources such as customer reports, competition, patents, or government agencies. After a need has been recognized, specifications and requirements of the design are developed. These specifications should be as specific as possible and care must be taken to include safety, health, and legal requirements. In addition they should meet with government, commercial and industrial standards.

After the specifications and requirements are determined, then a feasibility study is developed to verify the possible success or failure of a proposal both from a technical and economic standpoint. Considerations include, are there any natural laws being violated, are specifications realistic in terms of current technology, is there a dependency on scarce materials and will the end product cost be too high. It is this phase of design that can be well suited for applications of knowledge based systems. For instance, some of the tools required for a feasibility study include a knowledge of the engineering sciences, have a good grasp of material usage, production methods and sales department requirements. It is also in this phase that modifications to the specifications be performed to improve the success of the design. Incorporating this knowledge into a knowledge based system can serve as a tool to aid the engineer in this phase by relieving the engineer of the mundane, tedious, and error-prone tasks associated with developing mathematical models, analysis, and therefore reducing the time required to complete the design process. This allows the engineer to evaluate further concepts quickly and efficiently.

By using the computer to perform those tasks mentioned above, this allows the engineer to spend more time in the next phase of the design called *creative design synthesis*. This is perhaps the most challenging and interesting part of the design process. Here the engineer can act as engineer, inventor, and artist. This is where the creativity occurs. By using computerized tools such as computer aid design, prototypes can be developed and tested at the engineer's workstation instead of the research and development shop. Tools of this nature allow the engineer to perform what-if situations, therefore increasing the engineer's creativity potential. The next phase of design is *preliminary design and development*. Usually there may be one or more designs that will meet the specifications. The engineer will select one design using decision tools such as matrix tables, and probability theory. Lay out drawings are prepared showing the overall configuration and functional relations between parts of the machine. Further development work may be needed to prove and idea, determine critical material properties, or to evaluate a device. The use of knowledge based systems in this phase can further aid the engineer and result in reduced lead time towards final production.

The detailed design phase is concerned with part sizing and dimensioning, special processes are specified and materials are selected. With the aid of a draftsman, complete drawings are made. Using a knowledge based system that interfaces with currently available design databases such as Metadex allows the engineer to *pull-up* the necessary information quickly and efficiently. After detailed drawings, sub-assembly and assembly drawings including materials and parts lists are complete, the design is ready to enter the *prototype building and testing phase*. During this phase the prototype is constructed and tested, and any modifications necessary are added. Modifications are made as required during the prototype testing.

Any changes that are compatible with the best methods of production are made in the design for production phase. This phase is sometimes referred to as *value analysis*. Value analysis is becoming an ever growing and more important consideration in design. Knowledge based systems developed for this stage of development would include decision structures for manufacturing methods such as stamping, casting, forging, and welded assembly. One of the systems that will be studied later covers this area of design. During this phase product release production prototypes are built and tested. This includes solving production problems and changes are made to the design as required.

The design process as outlined above, starts with the visualization of the product at the highest abstract level, and as design progresses, this abstraction is refined into smaller subsystems. Such an approach is referred to as hierarchical planning. Depending on the complexity of the product, the design process may become very complex, requiring different problem solving strategies at different levels of the design. For example, the overall approach to machinery design requires working from the abstract to the detailed, but some aspects of the building design may require proposing details and working toward more general abstractions. Also, the design process rarely follows the indicated order of tasks without backtracking. It is necessary to make assumptions during design that may lead to inconsistencies or contradictions as the design progresses. In such cases it is necessary to backtrack to a previous task and revise the assumption.

In addition to the description of engineering design in terms of the solution process, it is important to describe the design problem in terms of the constraints on the solution. Engineering design is constraint oriented: much of the design process involves the recognition of applicable constraints and the satisfaction of these constraints. There are many sources of constraints and the satisfaction of these constraints, ranging from subjective constraints imposed by individuals to constraints imposed by the fundamental laws of nature. The efficient and knowledgeable handling of the potentially large number of constraints can expedite the design process.

Overview of Knowledge Based Systems

The range of tasks performed by experts consist of a spectrum bounded by derivation and formation tasks (Amarel 1978). In derivation tasks, the problem conditions are described as parts of a solution description; this description is completed by using the rules so that the given facts are well integrated into the solution. In formation tasks, the problem conditions are given in the form of properties that the solution as a whole must satisfy; the possible candidate solutions are generated and tested against the given conditions or constraints. In real life, most tasks fall between these two extreme categories. Tasks normally encountered at the derivation end of the spectrum are: interpretation diagnosis, monitoring, control and repair. Planning and design are typical of tasks at the formation end (Maher et al. 1984).

Knowledge based systems are interactive computer programs which are designed to emulate the reasoning of an expert in a given domain. This provides the user of the system with the expertise and advice for a wide range of problems such as those described above. The technology of knowledge based systems is presently characterized by the management of complex objects (the

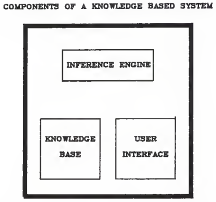


Figure I-1: Components of Knowledge Based System

knowledge) organized into relative simple structures (Barr 1981). Knowledge based systems basically have three main components outlined in Figure I-1. These components are typically the *knowledge base, the user interface, and the inference engine (rule interpreter).*

The knowledge base is the area where the system stores and retrieves it's knowledge. This can either be declarative knowledge such as general facts or heuristics, or procedural knowledge (the course of action to take) which can be in the form of scripts, frames, production rules or logic. The inference engine controls the processing of the information that is stored in the knowledge base. The inference engine uses *heuristic* search techniques to determine how the rules in the knowledge base are to be applied to the task at hand. It is the inference engine that governs the knowledge based system by accessing the appropriate rules, then executing the rules and then determining the acceptable solution. The user interface is the component of a knowledge based system that communicates with the user. The user must have the capability of communicating with the system in order to define the problem and to receive the results.

A knowledge base for a knowledge based system is constructed through a process of iterative development. After the initial design is implemented, the system begins to grow incrementaly both in breadth and depth. While other large software systems are sometimes built by accretion, the style of construction is inescapable for knowledge based systems because the requisite knowledge is impossible to define as one complete block.

One of the key principles in constructing a knowledge based system is transparency- making the system understandable despite the complexity of the task. Buchanan and Shortliffe (1984) suggested a knowledge based system must be understandable for the following reasons:

1. The system matures through incremental improvements, which require thorough understanding of previous versions and of the reasons for good and poor performance on test cases.
2. The system improves through criticism from persons who are not (or need not be) familiar with the implementation details.
3. The system uses heuristic methods and symbolic reasoning because mathematical algorithms do not exist (or are inefficient) for the problems it solves.

Hayes-Roth et al., (1983) divided applications using knowledge based systems into ten separate categories listed on the following page.

Category	Problem Addressed	Types of Systems
Interpretation	Infers situation description from sensor data	Speech understanding, image analysis, surveillance
Prediction	Infers likely consequences of given situations	Weather forecasting, Crop estimation
Diagnosis	Infers system malfunctions from observations	Medical, electronic, machinery, automotive
Design	Configures objects under constraints	Circuit layout, budgeting
Planning	Designs actions	Automatic programming, military planning
Monitoring	Compares observations in order to plan vulnerabilities	Nuclear power plant regulation, fiscal management
Debugging	Prescribes remedies for malfunctions	Computer software
Repair	Executes a plan to administer a prescribed remedy	Automobile, computer
Instruction	Diagnoses, debugs, and corrects student behavior	Tutorial, remedial
Control	Interprets, predicts, repairs, and monitors system behaviors	Air traffic control, battle management

Problem Solving and Search Strategies in Artificial Intelligence

Problem solving involves the search for a solution through a *state space* by the application of operators, where the state space consists of an *initial state, a goal state* and *intermediate states.* The solution path consists of all states that lead from the initial state to the goal state. Domain independent problem-solving strategies are commonly referred to as *weak methods* and may lead to combinatorial explosions. Knowledge based systems can be considered strong problem solvers since they employ domain knowledge in the solution strategy (Mayer et al., 1984). The overall computational efficiency of a knowledge based system depends upon where along the informed/uniformed spectrum the control strategy lies. There are generally two costs associated with computational costs; costs applied to the application of rules and the application control costs. A control system that is uninformed incurs only a small application control cost because rule selection is arbitrary and does not depend upon costly computations. However, such a strategy results in high rule application costs because it needs to try a large number of rules to find the optimum solution. To inform a control system about the problem domains of interest in Artificial Intelligence typically involves a high-cost control strategy, both in terms of storage and the number of computations required. Control systems of this type, however, results in minimal rule application costs; they tend to guide the production system directly for a solution.

The total computational costs of a knowledge based system is the summation of all rule application costs and the control strategy costs. A goal of the system designer is to decide how to efficiently balance these two costs. The procedures that the control system follows as it makes rule selections can be regarded as a *search* process. The choice of a control strategy affects the contents and how the knowledgebase is organized. In general, the goal is to reach the optimum result by applying an appropriate sequence of operators to an initial task-domain situation. Each occurrence of an operator modifies the situation in the same way.

Forward and Backward Reasoning

The application of operators and procedures to those structures in the knowledgebase that describe the task-domain situation-start with an initial state and progress to a goal state is called forward reasoning or forward chaining. This system contains an initial global knowledgebase of representations for a given set of facts. The control structure of this system then manipulates those facts through formal logic procedures into the optimal goal state. Rule-based problem-solving systems are built around rules which consists of an *if* part and a *then* part. Rule-based systems provide an excellent example of forward reasoning. The following example depicts the structure of a rule in a rule-based system:

```
Rn  If   condition 1
         condition 2
         .
         .
         condition n

    Then action 1
         action 2
         .
         .
         action n
```

When all the conditions in a particular rule are satisfied by the current situation, the rule is said to be *triggered*. When the actions are performed, the rule is said to be *fired*. Triggering however, does not always mean firing, because the condition of several rules may be satisfied simultaneously, triggering them all, making it necessary for a conflict resolution procedure to decide which rule actually fires.

The main drawback of this strategy is that it is extremely wasteful to require as input data all the possible facts for all conditions; in many circumstanees all possible facts are not known or relevant. This strategy is useful however in situations where the knowledgebase contains a large number of hypotheses and few data that is input. The forward reasoning strategy is not appropriate for a design oriented problem if possible goal states of the design problem are not easily represented by a diserete number of hypotheses. Forward reasoning may be used for eertain subtasks of the design process, sueh as seleeting the appropriate modeling options for the analysis of a speeified configuration.

An alternative strategy, reasoning backward or baekward ehaining involves another type of operator (implication), whieh is applied, not to a current task-domain situation, but to the goal. Baekward reasoning starts with the goal state and tries to ehain baekward by proving the initial states. Most human problem-solving behavior is observed to involve backward chaining; the human sets goals and then starts to perform the neeessary tasks to aeeomplish particular goal(s). Many Artifieial Intelligenee teehniques are based on this type of search strategy. Baekward reasoning, in its pure form, is not appropriate for the engineering design proeess, since the possible goal states of the design proeess are not easily represented by theory. However, certain subtasks of the design proeess may be suitable for baekward reasoning. Subtasks of the design process that involve mathematieal computation merit using baekward reasoning. The following example depiets an example for determining the motion of a projectile using baekward reasoning:

$$\begin{gathered} \textit{position}(x,y) :\text{-} \; ([v_o \cos(\textit{theta})]t \; \&\& \; [v_o \sin(\textit{theta})t \\ - [/ \; (g \; (t)^2 \;) \; 2) \end{gathered}$$

In this example, if the position and the velocity (v_o) were known to the system, the system could determine at what angle the projeetile's path was at time t.

Means-ends Analysis

The means-ends analysis process centers around the detection of differences between the current state and the goal state. When a difference is determined, an operator that can reduce the difference must be found. Sometimes an operator cannot be applied to the current state and requires the setup of a subproblem of reaching the appropriate state in which it can be applied. In addition, the operator may be incapable of producing the exact goal that is sought. This requires a second sub-problem of getting from the current state it produces to the correct goal. Means-ends analysis utilizes both the forward and backward reasoning techniques. This strategy however, can only be applied to those tasks where the measures of difference between the various states and the operators can reduce these differences, by formulating from cause to effect.

The first knowledge based system to use means-ends analysis was the *General Problem Solver* (Newell 1963). The design of this system was inspirational by the observation that people often use this technique when they solve problems. The General Problem Solver provides an understanding of the fuzziness of the boundary between designing programs that simply solve algebraic problems and those that simulate the human thought process. Means-ends analysis relies on a set of rules that allow it to transfer one problem state to another. The use of means-ends analysis requires the formulation of possible states in the solution path and of the operators required to move from one state to another. At this time, most engineering design processes are not sufficiently formalized for this representation.

AO Graph Search (Problem Reduction)*

The factoring of problems into smaller subproblems is called problem reduction. In the problem reduction approach, the principle data structures are problem descriptions or goals. An initial problem description is given; it is solved by a sequence of transformations that ultimately change it into a set of subproblems, whose solutions are immediate. The transformation that result are defined as operators. An operator has the ability to change a single problem into several subproblems; to solve the initial problem, all subproblems must be solved. A problem whose solution is immediate is called a primitive problem. Therefore, a problem representation using problem reduction is defined by the following components; 1) *an initial problem description*, 2) *a set of operations for transforming problems into subproblems*, and 3) *a set of primitive problem descriptions.* Reasoning proceeds in this case backward from the initial goal state.

A tree notation can be generalized if it is to represent the full variety of situations that may occur in problem reduction. The generalized notation for problem reduction is the AND/OR graph. Rules for the formation of an AND/OR graph is as follows:

1) All nodes on the tree must represent either a single problem state or a set of problems to be solved. The root node is at the top of the tree graph and will not have any parent nodes.

2) When a particular node has no decedents, this is referred to as a terminal node. This node represents a primitive problem.

3) For each possible application of an operator to a problem, transforming it to a set of subproblems, there is a directed arc from the root node to a node(s) that represent a resulting subproblem set.

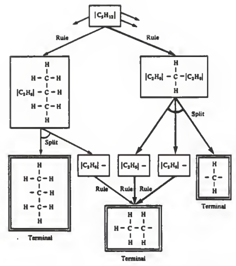


Figure I-2: Partial DENDRAL Graph

DENDRAL, a system that can propose plausible structures for rather complex chemical compounds, would when presented with a chemical formula might produce the AND/OR graph in Figure I-2. The actual DENDRAL system drastically prunes the candidate nodes by using other chemical knowledge. In Figure I-2 the paths that are connected by a horizontal arcs are considered AND nodes and all nodes must be solved before the parent node is proven. The remaining nodes in the graph are referred to as OR nodes, and at least one of these nodes must prove to be true in order for the parent node to be proven.

In summary, for AND/OR graphs to find the solution to the initial problem, the system needs only to build enough of the graph to demonstrate that the root node can be solved. The following rules apply to the solving of a AND/OR graph (Barr 1981).

A node is solvable if -

1. it is a terminal node (a primitive problem),
2. it is a nonterminal node whose successors are AND nodes that are all solvable, or
3. it is a nonterminal node whose successors are OR nodes and at least one of them is solvable.

Similarly, a node is unsolvable if -

1. it is a nonterminal node that has no successors (a non-primitive problem to which no operator applies),
2. it is a nonterminal node whose successors are AND nodes and at least one of them is unsolvable, or
3. it is a nonterminal node whose successors are OR nodes and all of them are unsolvable.

The use of AND/OR graphs is easily applied to the design process, as current design practice typically reduces the design problem into subproblems. For use in a knowledge based system for design would require the development of an appropriate graph to represent a particular design problem.

Hill Climbing

Hill climbing is a search procedure that uses feedback from the testing procedure to help the inference engine decide which direction to move in the search space. This uses a *plan-generate-test* procedure that generates all possible solutions in the search space and tests each solution until it finds the solution that satisfies the appropriate goal condition. This procedure uses constraint-satisfaction techniques that create lists of recommended and contraindicated sub-structures. It uses those lists so that it can explore only a fairly limited set of structures. Constrained in this way, this procedure has proved to be highly effective.

In a pure generate and test procedure, the returned value is either yes or no. But if the test function is changed to include a heuristic function that provides an estimate of how close a given state is to the goal state. This provides the procedure with the ability to select the best possible path.

The hill climbing procedure begins by generating the first proposed solution in the same way as would be done in a generate test procedure. If the result is a solution the procedure terminates. If the result is not the solution then some number of applicable rules generate a new set of proposed solutions. For each of the proposed solutions, the test function determines if it is an acceptable solution. If the proposed solution is found not to be the appropriate solution, then the procedure determines if that solution is the closest to a solution of any of the elements tested so far. If so the procedure remembers it, if not it is discarded. Next it takes the best element that it found above and uses that element as the next proposed solution. This step corresponds to a move through the problem space in the direction that appears to be leading the most quickly towards a goal.

One problem that may occur with hill climbing, namely what to do if the process gets to a position that is not a solution but from which there is no move that will improve things. This will happen if the system has reached a *local maximum, a plateau, or a ridge.* A local maximum is a state that is a better solution when compared to its neighboring solutions, but is not better than some other states that are further away. At this point all moves appear to make things worse. This problem can be frustrating because they often occur almost when a final solution is about to be reached.

A plateau is reached when a whole set of neighboring states have the same value. When this occurs, it is not possible to determine the best direction in which to move by making local comparisons. A ridge is reached when the area of the search space is higher than surrounding areas, but single moves in any one direction cannot be made. A mechanism that is used whenever one of these above problems occur is backtracking, not to be confused with backward reasoning or chaining. The system simply backtracks to some earlier node, which is determined by the system in an attempt to solve the problem along a different path. In order for the inference engine to accomplish this task it must maintain a list of the paths that it took to reach this point. The hill climbing strategy is appropriate for the design process if appropriate tests can be formulated. Typically, there is no unique solution to a design problem; therefore it can be said there exists no absolute test for a solution. This strategy would be well suited for the preliminary design phase, provided that the testing stage of design be restructured into ranking each solution generated by relative values.

Breadth-First, Depth-First Search and Best-First Search

Breadth-first search looks for the goal node among all nodes at a given level before using the *children* of those nodes to push on. A breadth first search procedure is guaranteed to find a solution if one exists, provided that there are a finite number of branches of the tree. The breadth-first method expands nodes in order of their proximity to the starting node, measured by the number of arcs between them. In other words, this method considers every possible operator sequence at a level *n* before solving any state conditions at level *n+1*. Figure I-3 depicts a tree graph representing the breadth-first search. The dashed line represents the flow of control in this search process. There are three major problems associated with the breadth-first search; it requires a

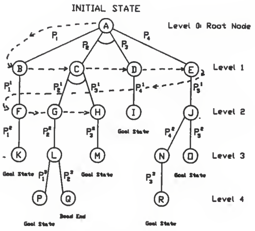


Figure I-3: Breadth-First Search

lot of computer memory, resulting in long periods of compute-bound processing. The number of nodes at each level of the tree increases exponentially with the level number, and each node in the tree must be stored in memory all at once. A lot of work is also required, particularly if the shortest solution path is quite long, since the number of nodes that need to be examined increases exponentially with the length of the path. Problems also occur from irrelevant or redundant operators that will greatly increase the number of nodes that must be explored.

Breadth-first search will be particularly inappropriate in situations in which there are many paths that lead to solutions but each of them is quite long. If this presents a problem, depth-first search is likely to find the solution sooner. Depth-first search is characterized by the expansion of the most recently generated, or deepest node first. The depth of a node is defined as follows; the depth of the root node is zero (0), and the depth of any other node is one more than the depth of it's predecessor. As a result of expanding the deepest node first, the search process follows along a single path through the state condition downward from the root node; only when it reaches a condition that has no successors does it consider an alternative path. Depth-first search can be dangerous. As the inference engine expands nodes downward away form the root node, it is possible to slip past levels at which the goal node appears and this results in wasteful energy in exhaustively exploring parts of the tree lower down. To prevent consideration of paths that may be too long, a maxima is often placed on the depth of nodes to be expanded. This is referred to as being *depth bound*. The hill-climbing search strategy outlined above incorporates depth-first search techniques. Figure I-4 represents a tree graph in depth-first search. The dashed line represents the flow of control through the tree. It should be noted that depth-first search follows in most systems a left-to-right order.

Best-first search provides a way of combining the advantages of both the breadth-first search and the depth first search into a single search strategy. At each level of the search process, the system selects the most promising of the nodes that have been generated thus far. This is accomplished by applying an appropriate heuristic function to each of the nodes. The system then expands that node by using rules that will generate the successor nodes. If one of the successor nodes is the goal, then the search process is terminated. If not, all of the new generated nodes are added to the set of nodes that have been generated thus far. Once again the most promising node is selected and the process continues. The path found by the best-first search is likely to be shorter than those found with other methods, because best-first search always moves forward from the node that seems closest to the goal node.

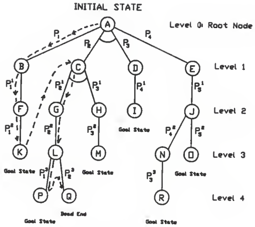


Figure I-4: Depth-First Search

Presented above are some of the more popular search methods used in the field of Artificial Intelligence. It should be noted that no one search strategy is considered ideal, rather the system designer needs to be able to determine which search process will best fit the application. The inference engine may be designed using different modules or subroutines that perform different search strategies based on the intended function of that module or subroutine. In addition, if the knowledge engineer is considering the purchase of commercially available packages, attention should be paid to the type of search strategy that is incorporated in that package. It can be frustrating to reach a certain point in the development of a knowledge based system and then realize that the search strategy of that package is unsuitable for the type of application being attempted. An understanding of all search strategies is invaluable to the knowledge engineer.

Languages And Tools For Building Knowledge Based Systems

There are currently a number of languages and tools available for building knowledge based systems. These packages may be grouped into three major categories:

1) General Purpose Programming Languages

2) General Purpose Representation Languages

3) Domain Independent Knowledge Based System Formulation Languages

General Purpose Programming Languages

Projects in Artificial Intelligence are usually implemented in a high level language. These high level languages need to contain planning and reasoning strategies. A number of knowledge based systems have been developed using a number of different languages, of which *LISP* and *PROLOG* seem very popular. The *C language* has been the choice of some researchers in the development of knowledge based systems applied to diagnostics.

General Purpose Representation Languages

General purpose representation languages are programming languages developed specifically for knowledge engineering. These languages are not restricted to implementing any particular control strategy, but facilitate the implementation of a wide range of problems that involve the derivation-formation spectrum. Some of the general purpose languages available are *KEE*, *SRL*, *OPS*, *IRIS*, *ROSIE* and *LOOPS*. Systems such as these which attempt to facilitate the construction of knowledge based systems, are an important area of current research.

Domain Independent Knowledge Based System Formulation Languages

This type of system provides the knowledge engineer with an inference mechanism, from which a number of applications can be built by adding domain specific knowledge. Systems that fall under this category include: *KES, EMYCIN, HEARSAY-III* and *EXPERT*. These systems also provide knowledge acquisition and explanation modules to simplify the construction of knowledge based system. EMYCIN for example, is a system for building expert systems in any domain and contains a module for the acquisition of knowledge. The EMYCIN system for example is used to construct a *consultation program*, by which is meant a program that offers advice on problems within its domain of expertise. The consultation program elicits information relevant to the case by asking questions. It then applies its knowledge to the specific facts of the case and informs the user of its conclusions. The user is free to ask the program questions about its reasoning in order to better understand or validate the advice given. The EMYCIN system is developed in LISP and a listing of the source code for the knowledge acquisition module is in the appendix. EMYCIN's representation of knowledge is in *attribute-object-value triples*, with an associated certainty factor. Facts in the knowledge base are associated in production rules.

Provided above are just a few of the systems and languages available for the development of knowledge based system. It is beyond the scope of this report to go into detail as to the function of each system and how each may be implemented into a knowledge based system. There are several reference books available to provide the reader with information on the different systems and languages available. In the next section of this report three knowledge based system applications applied to the design process will be covered.

AIR-CYL: Air Cylinder Design System

AIR-CYL uses the architecture of a hierarchically organized community of design agents called *specialists*. These specialists are actually modules or subroutines. This type of hierarchy reflects the hierarchical structure of the component being designed. The language used for the development of this system was DSPL, which stands for Design Specialists and Plans Language. DSPL captures domain knowledge much more lucidly by using primitives that are appropriate to the task. DSPL also makes appropriate classes of control behavior available to the designer in the form of constraints. The DSPL language was developed with the *Rutgers ELISP* language on a DECsystem-20.

Each specialist in the architecture of AIR-CYL has a list of design plans to accomplish certain design tasks at its level of abstraction. The specialists select choices from plans, makes some commitments, and direct the specialists at lower abstraction levels that refine the design. Failures that occur with the system, cause different kinds of actions, such as choosing alternative plans and transferring control to the parent specialist.

The upper level specialists in the hierarchy deal with the more general aspects of the component that is being designed, while the lower levels deal with the more specific subsystems or components. There are several types of agents or active problem-solver modules that exist in the decision-making structure. These agents include *specialists, plans, steps, tasks*, and *constraints*.

The specialists are designed to refine the design. Each specialist is responsible for the design of a major section of the component. This is accomplished through the use of a collection of plans. A *plan* is a sequence of calls to specialists or tasks. A plan therefore represents one method to design the section of the component that is represented by the specialist. A plan specifies the order in which each agent is involved. The *step* is the basic design module. Steps make design decisions and decide on a value for some attribute of the component. The values that are formulated are stored in a *design database*. The decisions that are made depend upon the present state of the design, taking into account any constraints. As an example, one step would decide the material to be used for some sub-component, while another step would specify its length or thickness.

A task is a sequence of steps which design a logically, structurally, or functionally coherent section of the component being designed. Constraints are used by the system to test for relationships between two or more attributes at particular design stages. Constraints can occur nearly anywhere in the hierarchy. An example constraint would check to see if a hole for a bolt is too small to be machinable in the material used.

The top-most specialist is responsible for the total design. The detailed decisions of the design are handled by other specialists lower in the system's hierarchy. Each specialist can make design decisions about the components and functions in its specialty. These decisions are reached on the basis of previous design decision from the other specialists in the system. These specialists may design the components themselves or use the services of other specialists that are below them in the hierarchy. Specialists that reside in the hierarchy attempt to refine the design independently, using their plans. The tasks that are attached to the specialists produce results using groups of steps, while constraints check the integrity and validity of the decisions being made.

Design Layout

The design activity associated with the system is broken down into four phases. These phases are shown in Figure I-5 and are describe below.

Requirement Phase. At this phase, requirements are obtained from the user and are verified both individually and collectively. Once the requirements are validated, the system attempts a rough design.

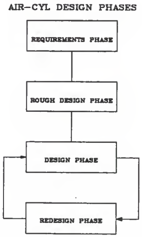


Figure I-5: Design Activity Phases

Rough Design Phase. During this phase those values on which much of the rest of the design depends will be decided and checked. The actual attributes decided depend upon the component being designed and the domain. It is likely that the decision for the higher-level attributes, such as the material to be used will be chosen during this phase. Specialists have both design and rough-design plans to select from, depending upon the current phase of design. Not all specialists will need both plans.

Design Phase. When the system has completed the rough design phase, the design phase can be initiated. The design phase starts with the top-most specialist and works down to the lowest levels of the hierarchy. A child specialist begins receiving a design request from its parent specialist. It also refers to the *specification database* for relevant specifications and then selects a plan using that data and the current design state.

The specialist can fill in some of the design and can call its successors in plan-determined order with requests to refine a substructure's design. The knowledge in the specialist assigns priorities to the plan and invokes alternative plans in case a later specialist fails. When all of a specialist's plans fail, the specialist informs its parent specialist.

Redesign Phase. If any failures occur during design, a redesign phase begins. If a redesign phase succeeds, the design phase will continue where it left off. The system is designed to try to handle all failures at the point where the failures occur before giving up and passing failure information to the parent specialist. A step, for example, may be able to examine the cause of failure and then produce another value to satisfy a failing constraint while still retaining local integrity. The system passes information and control messages between specialists across the connections forming the hierarchy. This provides communication within the system for controlled flow, and the system exhibits clear, well-focused problem-solving activity. This information could represent requests for action, report any failures, ask for assistance, or make suggestions. In addition to dependencies from specialists inside the hierarchy, the system may have dependencies from specialists outside the hierarchy. These outside agencies may be functions or modules that may be needed by several of the specialists within the hierarchy. For example, there may be modules for stress calculation or a database function. A human can also act as an outside agent or problem-solver, since any requests for help from the system will occur at well-defined points in the design.

Routine Design Example of AIR-CYL

AIR-CYL was designed and implemented to aid a company in designing air cylinders. Each air cylinder that the company manufactures had to be redesigned for every new customer. This was done to account for the particular space it had to fit in or the intended operating temperatures and pressures. The air cylinder that is shown in Figure I-6, has roughly 15 parts.

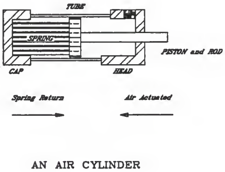


Figure I-6: Sample Air Cylinder

Before designing AIR-CYL, Brown and Chandrasekaran interviewed an air cylinder designer, analyzed the design protocols, and obtained a trace of the design process to establish the underlying conceptual structure in making an air cylinder. The partial conceptual structure established is shown in Figure I-7.

As an example, the structure shows that the cylinder head was treated as a separate conceptual entity. The spring was a parallel activity, while the rest of the design was treated by the designer as a third major activity. Because specialists could be fairly easily identified and plans for each specialist were few and identifiable, designing an air cylinder appeared strongly to be a routine design activity. In the examples that follow are simplified forms of the DSPL language. This task-based language allows expression of design agents, including specialists, and plans to carry out design objectives.

A *plan* consists of a set of actions, some of which may be run in parallel. The example shown on the following page depicts a plan with a task called *Validate and Process Requirements*, a constraint called *Head and Spring Compatible?*, and a specialist called *Rest*. Placed together, they form the design plan. Some of the specialists will also have rough design plans.

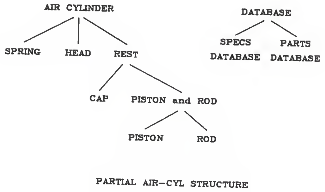


Figure I-7: Conceptual Structure

```
PLAN
  NAME                Air Cylinder Design Plan
  TYPE                Design
  USED BY             Air Cylinder SPECIALIST
  USES                Spring Head Rest SPECIALISTS
  QUALITY             Reliable BUT Expensive
  FINAL CONSTRAINTS   Design details OK?
  TO DO
        Validate and Process Requirements
        Rough DESIGN Air Cylinder
        PARALLEL DESIGN Spring AND Head
        TEST Head and Spring Compatible?
        DESIGN Rest
```

A *task* consists of the sequential use of steps, each of which obtains required information, makes calculations, and makes a decision about the value of a single attribute. The program segment shown below is a step that decides the seat width for the piston seal.

```
STEP
  NAME                Piston Seal Seat Width
  USED BY             Piston Seal
  COMMENT             Written by DCB
  ATTRIBUTE NAME      Seal Seat Width
  FAILURE SUGGESTION  INCREASE Piston Thickness
  REDESIGN            NOT POSSIBLE
  TO DO
        KNOWNS FETCH Piston Thickness
               FETCH Piston Material
               FETCH Minimum Thickness OF Piston Material
               FETCH Spring Seat Depth
        DECISIONS Available IS
               (Piston Thickness
                       MINUS DOUBLE Minimum Thickness)
               Seal Seat Width is 0.156
               COMMENT Using one size only
               TEST Available > Seat Seal Width?
               STORE Seal Seat Width
```

In this step, *Piston Seal Seat Width* is the name of a task, *Seal Seat Width* is the name of the attribute being decided, *Increase Piston Thickness* is what the step will then suggest if it cannot make a decision. Redesign is not possible for this step. *Piston Thickness* is a previously designed attribute, and *Available > Seat Seal Width?* is the name of a constraint. The *FETCH's* used in this step obtain the various values that it needs from the design database. Values are placed into the database with a command such as *STORE Seal Seat Width.*

System Failure Handling

In theory all design agents detect their own failures, try to determine what went wrong, try to fix it locally, do so if they can, and report failures only if all attempts fail. Agents that have some control over other agents can use those agents when trying to correct the detected problem.

There may be different reasons why an agent might fail. As an example, a step finds that a decision violates some constraint, a task discovers that a step's failure can not be handled locally, a plan can fail if it's not applicable to the situation, and a specialist can fail if all of it's plans fail. For every kind of failure, a message giving details is generated and passed back to the calling agent. This message may include if possible, suggestions about what might be done to alleviate the problem.

There are usually many kinds of problems that can occur, an agent will first look at the message to decide what went on below and what to do next. For some conditions, an immediate failure may be specified, while for others a redesign might be attempted. A *redesign* is associated with an agent and contains knowledge of how to change a design according to suggestions. Provided in the appendices in an annotated trace of the AIR-CYL system performing the design of an air cylinder.

Conclusions

There remains much work that needs to be done in the construction of knowledge based systems for this type of routine design activity. There is also a need for tools that represent knowledge at the task level. DSPL is an example of such a tool. DSPL is being studied and refined to make it more powerful, flexible, and easy to use. The designers expect to provide a graphical interface to show the development of the design as it progresses. Finally, work on the failure handling facility is required. It may be possible for the system itself to choose the requirements to relax, but a lot of special knowledge would be required to implement this.

The approach of using a hierarchically structured system with plan selection captures the essential qualities of routine design.

DACON: Design for Assembly Consultation System

The design for assembly consultation system (DACON) is aimed at tackling the problems of assembly rationalization, component optimization for handling and assembly process selection of a target assembly system and the optimum level of automation.

This system starts by trying to satisfy its goals and requests from the user information regarding component design relevant to assembly and its automation. DACON was designed in such a way that it attempts to create designs which can be readily handled and assembled. For example, when considering automatic handling the system will ensure that the component, if possible, can be fed automatically. If a problem arises and is diagnosed, appropriate design modification(s) will be suggested and assigned appropriate levels of confidence. DACON uses a scale from 0 to 1.00 to represent the level of certainty. The rule implications may also have an associated design comment or piece of cost information for use in subsequent assembly cost models. The rule in Figure I-8 is a sample DACON rule and shows both the certainty factor and the design comment, *Consider redesign options.*

IF: 1) Parts will tangle with other parts when in bulk supply.

2) A force and a manipulation is required to separate the parts.

THEN: 1) There is strong evidence (.95) that the part requires manual feeding.

2) Consider redesign options.

Figure I-8: Sample DACON Rule

When appropriate, the certainty of the users response is requested by the system and these certainties are manipulated to modify the hypothesis certainty factors and provide evidence to support the selection of conclusions. When the evidence obtained rules out one hypothesis, the system continues by attempting to prove another goal.

DACON provides interactive communication between the system and the user. DACON can be asked *why?* it is following a line of questioning and can display the rule it is trying to use. In addition, when DACON has reached a conclusion, the user can request to see the rule chain that was used to reach that conclusion.

Once the system has used its knowledge base to diagnose assembly problems and make decisions on design modifications, the system can be used to compute the costs of assembly. Mathematical models are used to compute the cost estimates. This module of DACON's hierarchy is actually a conventional program. The knowledge-based elements are used where the decision making defies scientific treatment. Having evaluated the costs associated with the assembly, DACON displays handling and assembly costs for all the components in the product.

Coding systems are used to analyze the processes involved in the two areas, distinct systems being needed for each area in both manual and automatic assemblies - four codes in total. These coding systems provide the engineer with feedback on the relative ease or difficulty with which the design can be handled and assembled, compared with the ideal or optimum design. The design-analysis procedure also involves the calculation of an *assembly-design efficiency*, which prompts the engineer to question the number of components in their design. This efficiency is obtained by determining the theoretical maximum number of parts that can be handled.

The DACON knowledge base has been constructed from published material, case studies, and with discussions with domain experts; it currently embodies 120 rules. It is planned to explore the capabilities of DACON in advising the professional engineer by applying the system to products in the electrical engineering industry.

The work that has been carried out on cost estimation has indicated that the DACON system can yield estimates within 10% of recorded costs for both automatic and manual assembly systems provided company-specific cost data is used. Work is also being carried out to determine how well DACON performs, as compared to a human expert, when used by an engineer of lesser expertise. Application of the DACON system to assembly redesign studies have shown that it can produce at least as good a result as the teams involved in the studies.

POLYCOAT - Design for Coating

In developing a coatings expert system to go with the DACON system, POLYCOAT is a polymeric coating knowledge based system. Using rules pertaining to operating temperatures, abrasive conditions, component flexing, adhesion requirements, color retention, solvent contact, environment acidity/alkalinity etc., it recommends the optimum polymeric coating material.

POLYCOAT currently incorporates roughly 50 rules and like DACON, was developed with the PROLOG language. The system investigates its hypothesis (coating materials) and asks the user for information regarding the part design and its environmental and operational requirements. Therefore, the knowledge-base is built up interactively by the design engineer. As with the DACON system, the confidence of user responses is requested and the system will

```
Please would you tell me if:
      'part comes into contact with acetic acid'

is true, false, or don't know (or why if you wish me to explain my
line of reasoning)?

==> false

Please tell me your confidence of this

==> 5

Please would you tell me if
      'environmental temperature < 200 deg C'

is true, false, or don't know (or why if you wish me to explain my
line of reasoning)?

==> why

I am trying to use rule 20
IF,NOT,part comes into contact with acetic acid
      and, environmental temperature < 200 deg C
      and, NOT, part requires color retention
      and, NOT, part is required to flex during service
THEN, there is evidence (0.85) that an epoxy ester coating is
      suitable.

Please would you tell me if....

[The consultation continues with many additional questions and
 resultant deductions]
```

Figure I-9: Sample POLYCOAT Dialogue

indicate hose processes which are like to be suitable, allocating a suitability rating on confidence to each conclusion. Figure I-9 shows an example of the dialogue used between the system and the engineer.

POLYCOAT is still in its initial stages. It uses data which is readily available and has a comparatively small number of rules in comparison with other knowledge-based systems. A simplified system representation of the factors considered in building the knowledge base is shown in Figure I-10.

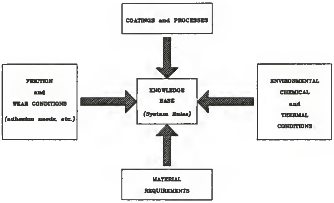

COATINGS and PROCESSES
FRICTION and WEAR CONDITIONS (adhesion needs, etc.)
KNOWLEDGE BASE (System Rules)
ENVIRONMENTAL CHEMICAL and THERMAL CONDITIONS
MATERIAL REQUIREMENTS

Figure I-10: System Representation

The knowledge base is a key factor in the success of a consultation system, after all it is the knowledge of the human expert. In general, expert design assessment and cost estimating is a time consuming problem that can span weeks of intensive effort. DACON and POLYCOAT should have the capability to reduce this activity duration giving an instantaneous response.

The CARTER System

Carter is an expert system that designs crankcases without any intervention from the user (other than setting the problem by describing the mechanism to be encased). To reduce the cost of algorithms such as plate theory, hamiltonian path in a graph, and finite element analysis, the system relies mainly on the engineer's knowledge to infer solutions, or at least to set the boundary conditions and limit the search space.

The user supplies as inputs the shafts, which at this time, must all be parallel or perpendicular to each other; giving the length and diameters of each cylindrical or conical subsection. Next the user inputs descriptions of the gears and bearings supported by these shafts, and the torques and powers. In addition, other indications can be supplied, such as clamping points, lubrication, boundary values of strains, precisions, safety factors, manufacturing method and heat sources.

The first step is to determine the frame thickness. Space required for the drive shafts can be divided by planes such that the slice of the mechanism that lies between two consecutive planes is entirely described by its projection onto those planes (Figure I-11). From the three slices shown in Figure I-11, the external tangents to the projections mentioned above are combined, giving the intersections of the required frame with all planes (Figure I-12). Those intersections are then linked together by means of planes, and bits of cones or cylinders. Figure I-13 shows the completed solution graph.

Design of the bearing housing is the next phase. The problem that occurs is finding the minimal set of bearing housings and thin walls needed. This type of problem is the same as finding the minimal path within a graph, where nodes are the bearings and the external case. Valuation or the arc between two nodes is the surface (cost) of the minimal wall that might link the two corresponding objects (Figure I-14).

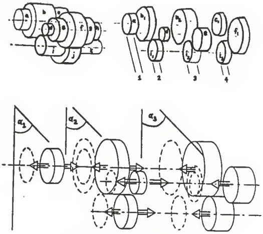


Figure I-11: Plane Slicing

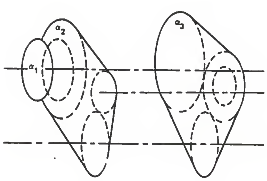


Figure I-12: Plane Graph

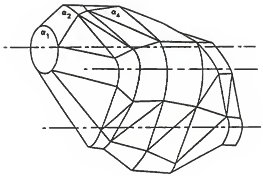
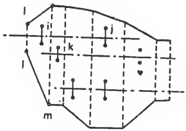


Figure I-13: Solution Graph



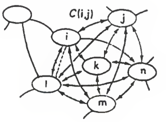


Figure I-14: Conceptual Designed Crankcase

The minimal path that goes once and only once through every node (hamiltonian path) is found by using the *Little* algorithm. When the system has found a suitable shape for the crankcase, the system will switch from geometry to mechanics, and dress up the skeleton.

An assumption is made that the engineer who has designed the mechanism computed the bearings under the hypothesis that they would be held by housings of infinite stiffness. Seeking the lightest possible solution, no provision is made for infinite stiffness. The concern of the engineer is really the fatigue life of the bearings. Therefore the main concern of the system is to compute the thickness of the housing so that it will guarantee the expected endurance.

Once the system has found a solution, it tests that solution. The test deals with strains, stresses and the first eigen-frequency. This employs the use of finite element analysis. The crankcase shell is modelled as an assembly of plane elements. The basic elements chosen are rather sophisticated, and ensure edge continuity, so providing as few elements as possible.

The eigen-vectors and the deformed shape returned by the structural analysis provide the system with information about local and global modes. To alter these, the system must decide where to put the reinforcing ribs. The finite element modules is then called interactively until a satisfying compromise is found; if this results in too many ribs being placed on the structure, some thicknesses are increased in the crankcase.

Knowledge Base Structure

To model the huge amounts of knowledge needed, the system designers felt that conventional programming languages - however sophisticated - were completely inaccurate. The knowledge required was quite fuzzy, and needed to be constantly updated. Therefore a *production rule-based system* was used to build a *pattern-directed inference system.*

In order for the system to be easily modified, the system designers placed guidelines upon how to formulate rules in the knowledge base. The rules should be placed in random disorder so that no rule has priority over another rule. In addition, rules should not refer to other rules in the knowledge base. The user is responsible for providing chunks of knowledge, and must not be concerned with how that knowledge will be put into action. The basic syntax of a production rule is:

$$IF\ P_1\ and\ P_2\ and\ \dots\ and\ P_n\quad THEN\ P'_1\ ;\ P'_2\ ;\ \dots\ ;\ P'_n$$

where the P_j are predicates and the P'_j are either predicates or calls to routines provided by the user. An example of a rule used in the CARTER system is as follows:

IF SEARCH IS INSIDE THE ZONE, AND THERE ARE TWO BEARINGS IN THE ZONE, AND ONE MAY WORK ON THIS SIDE, AND IF THERE IS ONLY ONE SHAFT IN THE ZONE, AND IF OVERCROWDING ALLOWS AXIAL CONNECTING OF THE HOUSINGS OF THE ZONE, AND IF MOUNTING OF THE SHAFT IS 'O-LIKE'

THEN BUILD A SHARED HOUSING IN THE ZONE

There are currently 300 such rules in the Carter system, that deal mainly with:

* Technological Options
* Shapes and Thicknesses
* Tools and Costs
* Setting the hypothesis for mechanical computings
* Piloting these computings

The principles of independence allows the user to modify this set of rules, and run time is almost independent of the number of rules in the knowledge base (approximately 20 ms per inference, on a 32-bit mini-computer).

Inference Engine

The multipurpose system that runs this knowledge base is called *GOSSEYN*. It comprises of a rule compiler, and the engine proper. The rule compiler:

1) reads the rules, checks for syntactical errors and gives out various listings (source, object, cross-reference, etc.)

2) builds up a network of pointers from the variables to the predicates containing them, then from the predicates to the rules (this network will enable the inference engine to be much more efficient than an ordinary rule interpreter.

3) translates the French source code into a first-order prefixed internal representation (this translation uses a dictionary and a set of rewriting rules, both provided by the system designer)

The Engine itself:

1) can be seen as a *pattern matcher* that explores the working zone (short-term memory) to find facts that will satisfy all the conditions of a rule.

2) is fully data driven because of the compilation mentioned above (ie., the advent of a new situation immediately fires the appropriate rule)

GOSSEYN is domain dependent and is being used for various computer-aided design systems (CAD). It is composed of approximately 20,000 Fortran statements and runs on a NORSK Data 560 mini-computer; routines that are specific to the CARTER system amount to roughly 5000 statements (1000 of which are borrowed from NASTRAN).

The design of the system involves some rather sophisticated computing. This includes the use of finite element analysis, plate theory, etc., combined into one package. The amount of calculations that is needed however, can be reduced considerably if the knowledge, the expertise, and the tricks retained by the design engineers are called upon. This system is enhanced by the use of a natural language interface, which in this case is French.

CONCLUSIONS

1) Design is a highly creative activity involving diverse problem-solving techniques, and many kinds of knowledge. Design is also a complex activity, one that artificial intelligence has only relatively weak theories of, especially for more creative design activity.

2) The distinction between databases and knowledge bases, is based on the distinction between data and knowledge. A database consists of mathematical structures together with a computational theory that states how the structures can be implemented efficiently. A knowledge base contains symbolic structures with a notion of interpretation of a specific subject matter.

3) Logic rules can already be processed in a database management system. These correspond to the definitions in a database management systems knowledge representation. These definitions however are not recursive, and recursive definition would provide a dramatic increase in the knowledge representation power of database management systems. This would allow more processing to be off-loaded from the knowledge based system.

4) Several semantic problems remain to be resolved, and performance issues are largely unexplored. Most work on spatial databases has been application specific and involved ad-hoc extensions to a database management system.

5) Artificial Intelligence and Database research has a fundamental difference in time frames. The artificial intelligence timeframe is at least a decade away. Rather than discussing how to improve and integrate existing AI and Database

technology now, AI researchers concentrate on longer term issues such as developing the next generation of AI systems. The database timeframe is tomorrow.

6) The solution to integrating AI and Database technology is not to simply stick them together now. Researchers in this area must determine how the two technologies might fit together. Knowledge based systems can be readily extended over a period of time by researchers refining old rules and adding new ones.

7) Systems can be developed to be introspective, provided that they can check the consistency of their own rules, and evoluntionary, if they have the capability to modify their own rules and add new ones.

8) These systems can give explanations of their line of reasoning. For example, the rules being used can be displayed and the rule chain leading to a conclusion can be explored.

9) The analysis of the applications in the previous section identified three capabilities of using knowledge based systems and database management systems. A deeper analysis of these, and other applications will doubtlessly reveal additional capabilities. The full impact of moving capabilities to the database management systems will be understood only when processing and optimization techniques have been developed.

REFERENCES

Amarel, S. *Basic themes and problems in current AI research.* Proceedings of the Fourth Annual AIM Workshop, Ceilsielske, V. B. ed., Rutgers University (June 1978) pp 28-46.

Barr, A. and Feigenbaum, E. *Handbook of Artificial Intelligence Vol I* Heuris-Tech Press (1981) pp 216-222.

Birmingham, W. and Siewiorek, D. *MICON: a knowledge based single board computer designer.* Proc. 21st DAC IEEE (1984) pp 565-571.

Bowen, J. *Automated Configuration using a Functional Reasoning Approach.* Proc. of AISB 85 University of Warwick, UK (April 1985) pp 1-4.

Brodie M. L. *Knowledge Base Management Systems: Discussions from the Working Group.* Proc. 1st International Workshop on Expert Database Systems, (October 1984) pp 19-34.

Brown, D. *Failure handling in a design expert system.* Computer-Aided Design Vol 17 No 9 (November 1985) pp 436-442.

Brown, D. and Chandrasekaran, B. *Expert systems for a class of mechanical design activity.* Knowledge Engineering in Computer-Aided Design. North Holland (1985) pp 295-282.

Dixon, J., Simmons, M. and Cohen, P. *An Architecture for Application of Artifical Intelligence to Design.* Proc. 21st DAC IEEE (1984) pp 634-640.

Ellison, B. and Boothroyd, G. *Applying Design for Assembly Handbook to Reciprocative Power Saw and Impact Wrench.* National Science Foundation Report 10197 No. 10 University of Massachusetts (April 1980).

Grinberg, M. *A knowledge-based design system for digital electronics.* Proc. 1st Annual National Conference AI AAAI (August 1980) p 283.

Haeusler, J. *Design for Assembly - State of the Art.* Proceedings of 2nd International Conference on Assembly Automation (May 1981).

Kowalski, T. and Thomas, D. *The VLSI design automation assistant: prototype system.* Proc. 20th DAC IEEE (1983) pp 479-483.

Latombe, J. *Artificial intelligence in computer-aided design: the Tropic system.* TR 125, Stanford Research Institute (February 1976).

Maher, M. Spiram, D. and Fenves, S. *Tools and Techniques for Knowledge Based Expert Systems for Engineering Design.* Advanced Engineering Software. Vol 6 No 4 (October 1984) pp 178-188.

Maher, M. *HI-RISE and beyond: directions for expert systems in design.* Computer-Aided Design Vol 17 No 9 (November 1985) pp 420-426.

Matthews, A. and Swift, K. *Expert computer systems in engineering design.* Engineering (London) Vol 223 No 9 (September 1983) pp 673-675, 677-678.

McDermott, J. *R1 - a rule-based configurer of computer systems.* Artifical Intelligence Vol 19 No 1 (September 1982) pp 39-88.

Minsky, M. *A Framework for Representing Knowledge.* in P. H. Winston (ed.), The Psychology of Computer Vision. McGraw Hill (1975)

Mitchell, T. *An intelligent aid for circuit redesign.* Proc. AAAI Conference. (1983) pp 274-278.

Newell, A. and Simon, H. *GPS, A Program That Simulates Human Thought.* Computers and Thought. E. Feigenbaum and J. Feldman (Eds.). McGraw-Hill (1963).

Preiss, K. *Data frame model for the engineering design process.* Design Studies (1980) 1 (4), 231 (IPC Business Press).

Rehak, D. and Howard H. *Interfacing expert systems with design databases in integrated CAD systems.* Computer Aided Design Vol 17 No 9 (November 1985) pp 443-454.

Schank, R. and Abelson, R. *Scripts, plans, goals, and understanding.* Hillsdale, N.J.: Lawrence Erlbaum. (1977)

Sussman, G. *Electrical Design - a problem for AI research.* Proc. Int. Joint Conf. Artificial Intelligence (1977) pp 894-900.

Van Melle, W. *A domain independent system that aids in constructing knowledge-based consultation programs.* Memo HPP-80. Stanford (June 1980).

Watterman, D. and Hayes-Roth, F. *Pattern Directed Inference Systems.* Academic Press, New York (1978).

APPENDIX

Machine-Readable Files of Materials Engineering Data

File Name	Custodian	Materials
WBD	Betriebsforschungs-institut VDEh Dusseldorf West Germany	a) measured values 300 grades steel b) standard values 1000 grades steel
—	AMMRC Watertown MA	52 materials
SACRD	Oak Ridge National Laboratory	Approximately 70 materials used in breeder reactor safety analyses
—	Fracture Control Corp under EPRI sponsorship	60 heats of 7 base metals
—	Institute of Materials Zagreb Yugoslavia	58 tool steels
—	General Dynamics San Diego CA	3 advanced composite materials
—	Fiat Milan Italy	300 materials— ferrous Cu A1
—	Electrotechnical Laboratory Ibaraki Japan	Compound semiconductors
EDB	Faculty of Engineering University of Tokyo	Materials for energy applications
—	ICI Ltd	100 plastics
—	Lawrence Livermore Lab DOE	Fiber composites for flywheel technology
MPDC	Battelle Memorial Institute Columbus OH	Several hundred metals and alloys
CHAT	International Harvester	Steels
—	SINTEF Trondheim Norway	26 A1 alloys
—	Copper Development Association	89 cast Cu alloys 121 wrought
—	GE Aircraft Engine	High-temperature materials
—	J Deere & Co	Approximately 600 materials mostly steel some A1 alloys
CASIS	Metalsoft Ltd	Approximately 200 steels
—	Ford Motor Co	Steels A1 alloys
—	Rolls Royce	High-temperature alloys
INFOS	Rheinisch-Westfalische Technische Hochschule-Aachen	NA
FPBD	EPRI	Light water reactor fuel performance
—	Materials Research and Computer Simulation Corp	Approximately 100 nuclear reactor structural materials
—	Institute of Leichtbau Okonomische Verwendung v Werkstoffen	Steels, numerous plastics-based composites
—	Aluminum Association + MPC	2 alloys of A1
—	GKN Wolverhampton England	Approximately 400 materials mostly steels some A1 alloys
—	Pure Carbon/Triboon Bearing	400 bearing materials
PEFAC	Power Engineers Association England	NA
RAFSAS	Technical Institute Kawasaki Heavy Industries Ltd Akashi Japan	NA
HYDROGEN DATA	CNRS-MIDIST Paris, France	Metals and alloys
EIerS	Institution of Electrical Engineers	22 electronic materials
FMDB	Japanese Ministry of Education Science & Culture	Materials for fusion, mechanical and nuclear reactors (metals and ceramics)
CEMP	CNRS-MIDIST	Plastics
—	Microswitch Division Honeywell	Thousands of engineering polymers
POLYPHONE	Cordura Corp, The International Plastics Selector	Approximately 8500 commercial plastics
COMPAT	US Army Plastics Technical Evaluation Center	Various
C-C composites	Battelle	Carbon-carbon composites
CUTDATA	Metcut Research	Ferrous and nonferrous
—	Kaman-Tempo	Metal-matrix composites
—	Institut Textiles de France	Textile fibers
—	Royal Aircraft Establishment Farnborough	Polymers
—	Cambridge University	304 stainless steels
—	JRC Petten Netherlands	800-type Fe-Ni-Cr alloys
—	CINDAS Purdue University	Approximately 3000 mostly elements, compounds and alloys, some ceramics and polymers
—	Naval Ordnance Laboratory	356 nonferrous 173 ferrous
—	University of Illinois for State Highway Department	Structural steels welded and bolted joints thereof
—	University of Dayton Research Institute	Ferrous and nonferrous structural materials
MIDAS	ASM/SDC	Ferrous and nonferrous alloys
—	Institute of Ferrous Metallurgy Gliwice, Poland	Structural materials
MATERIALY	Tenoma Warsaw Poland	Approximately 3000 structural materials
—	Czechoslovakian Institute of Standardization and Quality Bratislava	Materials, tool steels, refractory steels, 600 plastics
SACDA	University of Western Ontario	NA
—	Hercules Inc Salt Lake City	Polypropylene
PLASTICS SELECTOR	Northern Telecom Brampton, Ontario	300 plastic materials
Mechanical Properties Data Analysis Center	Oak Ridge National Laboratory	8 nuclear reactor structural materials
CAMS	Naval Research Laboratory	15 metals
Capricornia Materials Selector	Capricornia Institute of Advanced Education Australia	31 materials
—	Chrysler Motor Company	Steels

TABLE A.1: DATABASES OF ENGINEERING MATERIALS

Manufacturer and Model	Year	Comments
Xerox 1100 series	1981	Personal workstation supporting the Interlisp programming environment, with bytecode emulation facilities.
Symbolics 3600 series	1983	Provides direct hardware execution of data type checking at runtime, garbage collection, and instruction pipeline with prefetch.
Lisp Machines Inc. (LMI), Lambda	1983	Modularly expandable multiprocessor centered around a fast NuBus; provides almost unlimited virtual memory space.
Fujitsu ALPHA	1983	Hardware support for multiple processes through virtual stacks augmented by a cache that stores stack addresses of free variables.
Tektronix 4400 series	1984	Lower end workstations built around conventional Motorola 68010/20 processors.
Texas Instruments Explorer	1984	32-bit-NuBus-based open architecture, with a dedicated Lisp processor and multiprocessor expansion capabilities.
C-Lisp Machine University of Kyoto, Japan	1984	Shared memory multiprocessor with a master-slave control. Stack with medium and course execution granularity.
SNAP (UCLA)	1985	Semantic-net processing; production systems; and discrete relaxation for vision.

TABLE A.2: ARTIFICIAL INTELLIGENCE TOOLS

LISTING A.1: AN ANNOTATED TRACE OF AIR-CYL

This is a trace generated by the AIR-CYL system. It has been highly edited for brevity and for presentation in this format. The trace is of a successful design with step redesign and selection of alternative plans. The final design has been omitted.

```
              ****** AIR-CYL Air-cylinder Design System ******

*** Requirements input
    From file DCB:AC-Requirements-Test

!!!NOTE: There are about 20 values given as requirements, including the maximum
operating temperature and pressure and the size of the envelope in which the
air-cylinder must fit.

* Do you wish to alter the requirements? >>>????> yes

EnvelopeLength              7.83
EnvelopeHeight              1.5
EnvelopeWidth               1.75
MaxTemperature              250
OperatingMedium             Air
OperatingPressureMax        60
OperatingPressureMin        30
RodLoad                     1.4
Stroke                      1.75
RodThreadType               UNF24
RodThreadLength             1.031
RodDiameter                 (LNGTH 0.312 0.0 2.e-3)
Environment                 Corrosive
Quality                     Reliable
MTBF                        100000
AirInletDiameter            0.374
MountingScrewSize           (LNGTH 0.19 5.e-3 5.e-3)
MountingHoleToHole          (LNGTH 0.625 5.e-3 5.e-3)
MaxFaceToMountingHoles      (LNGTH 0.31 5.e-3 5.e-3)
```

```
* Alterations from user

System name for requirement is >>>????> EnvelopeWidth
            Current value is 1.75
            New value is >>>????> 1.35
```

!!!Note: We have cut down the width of the envelope without altering any other requirement to make the design harder.

```
System name for requirement is >>>????> quit

* End of alterations from user
*** Requirements Input Complete

--- Entering Specialist
   ...AirCylinder... Mode = Design


--- Entering Plan
   ...AirCylinderDP1... Type = Design


--- Entering Task
   ...CheckRequirements


--- Entering Step
   ...CheckEnvelope


--- Leaving Step
   ...CheckEnvelope... Result = Success Msg
```

!!!Note: Here, the system continues to check requirements. Next, the design plan being followed specifies the use of the AirCylinder specialist in rough design mode. A rough design plan is selected and followed, leading to a successful rough design. The AirCylinder specialist then leaves rough design mode and continues in design mode. After quite a lot of decision making involving subspecialists, we get to this point.

```
--- Entering Specialist
   ...Rest... Mode = Design


--- Entering Plan
   ...RestDP1... Type = Design
```

```
--- Entering Specialist
    ...PistonAndRod... Mode = Design
```

!!!Note: At this point the system is working on the design of the piston and rod assembly. This is where the trouble starts.

```
--- Entering Plan
    ...PistonAndRodDP1... Type = Design

          .

          .

          .

--- Entering Task
    ...PistonSeal

--- Entering Step
    ...PistonSealType

--- Leaving Step
    ...PistonSealType
    ...Result = Success Msg

--- Entering Step
    ...PistonSealSeatWidth
```

!!!Note: The constraint test that follows will discover that there isn't enough space in the piston for the seat for the seal that will go around the piston. Its failure produces a message that shows in detail how the failure occured. Here is shown only a part of the message.

```
--- Entering TEST-CONSTRAINTS
    ...(Available > Width)

--- Leaving TEST-CONSTRAINTS
    ...(Available > Width)... Result = Failure "Constraint Failure"
                    Explanation "Seal width is greater than available space
                                 in piston"
                    Suggest (INCREASE PistonThickness BY 1.517e-2)
                    Suggest (DECREASE PistonSealSeatWidth BY 1.517e-2)
```

!!!Note: The step failure handlers, which are built into the system, determine that a domain specific failure handler will be able to decide what to do. Domain specific failure handlers are written in DSPL by the expert or knowledge engineer.

```
--- Entering FailureHandler
   ...PistonSealSeatWidthFH
```

!!!Note: The domain specific failure handler says to try redesign.

```
--- Entering Redesigner
   ...PistonSSWRedesigner
              Step = PistonSealSeatWidth
              Suggest = (DECREASE PistonSealSeatWidth BY 1.517e-2)
```

```
--- Leaving Redesigner
   ...PistonSSWRedesigner
   ...Result = Success Msg
```

!!!Note: The piston seal seat width redesigner was able to decrease the width as suggested.

```
--- Leaving FailureHandler
   ...PistonSealSeatWidthFH
   ...Result = Success Msg
```

!!!Note: We leave the failure handler and return to the step. The redesign was successful, so the step is successful and acts as if no problems were encountered.

```
--- Leaving Step
   ...PistonSealSeatWidth
   ...Result = Success Msg

          .
          .
          .

--- Leaving Plan
   ...PistonAndRodDP1
   ...Result = Success Msg
```

```
--- Leaving Specialist
   ...PistonAndRod...Result = Success Msg


--- Entering Specialist
   ...Cap... Mode = Design
```

!!!Note: Now the attempt is made to design the cap - and discover another problem.

```
--- Entering Plan
   ...CapDP1... Type = Design

          .
          .
          .

--- Entering Task
   ...CapInternal


--- Entering Step
   ...CapInternalDiameter
```

!!!Note: The constraint tests to see if the internal diameter of the cap is larger than the outside diameter of the spring (one must fit in the other). It fails.

```
--- Entering TEST-CONSTRAINTS
   ...(CapID > SpringOD)


--- Leaving TEST-CONSTRAINTS
   ...(CapID > SpringOD)...Result = Failure "Constraint Failure"
                    Explanation "Cap internal diameter too small for spring"
                    Suggest (DECREASE SpringOD BY 9.9e-2)
                    Suggest (INCREASE CapInternalDiameter BY 9.9e-2)


--- Entering FailureHandler
   ...CapIDFH
```

!!!Note: The domain specific failure handler says to try redesign.

```
--- Entering Redesigner
    ...CapIDRedesigner
          Step = CapInternalDiameter
          Suggest = (INCREASE CapInternalDiameter BY 9.9e-2)


--- Entering TEST-CONSTRAINT
    ...(CapID > SpringOD)
    ...Result = Success Msg


--- Leaving Redesigner
    ...CapIDFH
    ...Result = Success Msg
```

!!!Note: The step is successful, since the failure was handled.

```
--- Leaving Step
    ...CapInternalDiameter
    ...Result = Success Msg

          .
          .
          .

--- Leaving Plan
    ...CapDP1...Result = Success Msg


--- Leaving Specialist
    ...Cap...Result = Success Msg

          .
          .
          .

--- Entering Specialist
    ...Bumper...Mode = Design
```

!!!Note: The bumper flange diameter must be large enough to support the spring. The constraint tests this and fails.

```
--- Entering TEST-CONSTRAINTS
    ...(BFD > SpringOD)
```

```
--- Leaving TEST-CONSTRAINTS
   ...(BFD > SpringOD)
   ...Result = Failure "Constraint Failure"
        Explanation "Bumper flange is to small for spring"
        Suggest (DECREASE SpringOD BY 2.995e-2)
        Suggest (INCREASE BumperFlangeDiameter BY 2.995e-2)


--- Entering FailureHandler
   ...BumperFDFH
```

!!!Note: The domain specific failure handler says to try redesign.

```
--- Entering Redesigner
   ...BumperFDRedesigner
        Step = BumperFlangeDiameter
        Suggest = (INCREASE BumperFlangeDiameter BY 2.995e-2)
```

!!!Note: The redesigner fails because there is no knowledge about increasing the value of that attribute.

```
--- Leaving Redesigner
   ...BumperFDRedesigner
   ...Result = Failure "Redesigner action section fails"
```

!!!Note: The failure handler reports failure and eventually the step gets told the bad news.

```
--- Leaving FailureHandler
   ...BumperFDFH...Result = Failure "Redesigner action section fails"

        .
        .
        .

--- Leaving Step
   ...BumperFlangeDiameter
   ...Result = Failure "Step failure"
```

!!!Note: The task passes the failure message from the step to its failure handler. It will determine if the task can do anything about the step failure.

```
--- Entering FailureHandler
    ...BumperFlangeFH
```

!!!Note: The failure handler for the task discovers that no suggestions have been passed up from below. This means that no redesign can be considered. The failure handler fails because it couldn't handle the problem.

```
--- Leaving FailureHandler
    ...BumperFlangeFH
    ...Result = Failure "No relevant suggestions for task redesigner"
```

!!!Note: The step failure and subsequent failing redesign attempt leads to a failure in the task.

```
--- Leaving Task
    ...BumperFlange
    ...Result = Failure "Task failure"
```

!!!Note: The plan fails due to the failing task.

```
--- Leaving Plan
    ...BumperDP1
    ...Result = Failure "Plan failure"
```

!!!Note: The next plan is selected since the last one failed.

```
--- Entering Plan
    ...BumperDP2... Type = Design

--- Entering Task
    ...BumperFlange2

--- Entering Step
    ...BumperFlangeDiameter2

--- Entering TEST-CONSTRAINTS
    ...(BFD < CapID)
```

!!!Note: This is the same constraint that failed in the last plan. This time it is OK. The step succeeds.

```
--- Leaving TEST-CONSTRAINTS
    ...(BFD < CapID)
    ...Result = Success Msg


--- Leaving Step
    ...BumperFlangeDiameter2
    ...Result = Success Msg

          .
          .
          .

--- Leaving Plan
    ...BumperDP2
    ...Result = Success Msg

--- Leaving Specialist
    ...Bumper...Result = Success Msg


--- Leaving Plan
    ...RestDP1...Result = Success Msg


--- Leaving Specialist
    ...Rest...Result = Success Msg


--- Leaving Plan
    ...AirCylinderDP1...Result = Success Msg


--- Leaving Specialist
    ...AirCylinder...Result = Success Msg


...Design attempt succeeds
                   ****** AIR-CYL Air-Cylinder Design System ******
```

LISTING A.2: KNOWLEDGE ACQUISITION MODULE FOR EMYCIN

```
 ********************************************************************************
 This is the Knowledge Acquisition Module for the CRIB Implementation Project
 ---------------------------------------------------------------------------
 This function is the routine that decides who is calling the KA (knowledge
 acquisition) module and then invokes the specific routine which responds
 to the contents of args.

 ********************************************************************************

(declare (special tunit Subunit-list) (*fexpr MIS) (macros t))

(eval-when (load) (MIS-SET-REAL 'KA))

(defun KA (who &rest args)
   (cond ((eq who 'UI) (cond ((eq (car args) 'edit)
                              (prog (t-list)
                                    (setq t-list (get-subunit-list))
                                    (cond ((eq t-list nil)
                                           (return 'done))
                                          (t (print (edit-subunit t-list))
                                     (terpri)
                                                 (return 'done))
                                     )
                               ))
                             ((eq (car args) 'add) 'add-subunit-HOOK)
                             (t 'ERROR->invalid-KA-request)
                        )
          )
         (t 'unrecognized-calling-module)
    )
)

;
;******************************************************************************** ;
;   When the UI -user interface- module edit request is received by the KA  ;
;(knowledge acquisition) module this function will edit the Subunit-list    ;
;that was requested by Subunit Number
;
;******************************************************************************** ;

;(defun edit-subunit (Subunit-list Last-Subunit-list)
;  (cond ((eq Subunit-list nil)
;          (patom '|ERROR->Subunit List Exhausted, Press 't' to Continue|)
;                 (terpri)
;          (continue)
;          (edit-subunit (car Last-Subunit-list) (cdr Last-Subunit-list)))
;                (t (list-subunit Subunit-list)
;
;               (caseq (get-option)
;                    (d (edit-subunit (cdr Subunit-list)
;                       (cons Subunit-list Last-Subunit-list)))
;                                       (u (cond ((eq Last-Subunit-list nil)

;                                      (patom '|Top of List, Press 't' to Continue|)
;                           (terpri)
;                            (continue)
;                            (edit-subunit Subunit-list nil))
;                        (t (edit-subunit (car Last-Subunit-list)
;                                  (cdr Last-Subunit-list)))))
;               (h (help-list)
;                   (edit-subunit Subunit-list Last-Subunit-list))
;              (q 'quit)
;               (s (MIS KA MON 'Include 'add)
;                  (MIS KA KB 'add))
;                     (t (patom '|ERROR->Illegal Command Press 't' to Continue|)
;                   (terpri)
;                   (continue)
;                  (edit-subunit Subunit-list Last-Subunit-list))))))


(defun edit-subunit (Subunit-list)
    (do ((current-list Subunit-list)
    (last-list)
    (option 'go (get-option)))
   ((eq option 'q))
        (list-subunit current-list)
        (caseq option
               (d (setq last-list (cons current-list last-list))
          (setq current-list (cdr current-list)))
               (u (cond ((null last-list)
                            (msg "Top of List, Press 't' to Continue" N)
                     (continue)
                            (setq current-list Subunit-list))
                      (t (setq current-list (car last-list))
                         (setq last-list (cdr last-list)))))
               (h (help-list))
               (s (MIS KA MON 'Include 'add)
                  (MIS KA KB 'update-the-rules Subunit-list))
               (go)            ; for first time through
               (t (patom "ERROR->Illegal Command Press 't' to Continue" N)
                   (continue)))))

;
;****************************************************************************** ;
;                                                                             ;
; Input the Subunit Number and Retrieve the Key Group, and Sub-group data    ;
; for the Subunit
;
;****************************************************************************** ;

(defun get-subunit-list ()
   (prog (tunit)
         (patom '|Enter the Subunit you wish to Edit |)
         (setq tunit (read))
         (MIS KA MON 'Include 'retrieve tunit)
         (return (MIS KA KB 'retrieve tunit))
   )
 )

;
;***************************************************************************** ;
; Utility functions
;
;***************************************************************************** ;

(defun cls ()
   (Swipescreen)
)

(defun continue ()
   (cond ((neq (read) 't) (continue))
         (t)
   )
 )


;***************************************************************************** ;
; List processes
;
;----------------------------------------------------------------------------;
; List out current Subunit list
;
;***************************************************************************** ;

(defun list-subunit (Subunit-list)
   (cls)
;  (print '<EDIT>) (terpri) (terpri)
;  (patom '|Current Sub Unit List|) (terpri)
;  (patom '| => |) (print Subunit-list) (terpri) (terpri)
;  (patom '| Code          Description <HOOK>|) (terpri)
;  (patom '| ----          -----------|) (terpri)
;  (patom '|  |) (print (car Subunit-list))
;  (terpri)
   (msg "<EDIT>" (N 2) "Current Sub Unit Info." N " => "
        ($prpr Subunit-list) (N 2))
   (msg "________________________________________________" N)
   (cond ((atom (car Subunit-list))
      (let ((decode (Decode (car Subunit-list))))
             (cond (decode (msg decode N))
                 (t (msg (car Subunit-list) N))))))
 )
```

```
;****************************************************************************** ;
; List out current Edit Options
;
;****************************************************************************** ;

(defun get-option ()
   (terpri)
   (patom '|Options -> u, d, h (help), q, s|)
   (patom '|   Enter Option => |) (read)
 )

;
;****************************************************************************** ;
; This function prints the descriptions of the edit commands when a h command
; is issued to the editor
;
;****************************************************************************** ;

(defun help-list ()
   (cls)
   (terpri)(terpri)
   (patom    '|----------------------------------------------------------|)
(terpri)
   (patom '|Command Descriptionn|)
   (terpri)
   (patom '|===================|)
   (terpri)
   (patom '|  u   Output Previous Subunit List Level|)
   (terpri)
   (patom '|  d   Go To Next Subunit List Level|)
   (terpri)
   (patom '|  q   Quit the Knowledge Acquisition Editor|)
   (terpri)
   (patom '|   s    Save the Current Subunit List in the Knowledge Base|)
(terpri)
   (patom    '|----------------------------------------------------------|)
(terpri)
   (patom '| Press 't' to Continue |)
   (continue)
 )
```

UTILIZATION OF DATABASES AND EXPERT SYSTEMS

IN THE DESIGN PROCESS

by

G. BENTON GIBBS

B.S., Kansas State University, 1985

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTERS OF SCIENCE

Department of Agricultural Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1987

The computer has relieved the engineer of many of the mundane, tedious, and error-prone jobs required to complete the design process. Using a combination of present technology for knowledge based systems development and database management systems may prove to be beneficial to future design systems. By the 1990's knowledge based management systems are expected to become one of the most important application development tools. Already an increasing number of applications, including military projects, office automation, diagnostic, and CAD/CAM have a need for this technology. With the use of knowledge based management systems a database can be more efficiently and intelligently enhanced. The merging of databases and knowledge based system technology should produce systems capable of managing a large database of complex knowledge in a integrated way.

Knowledge based systems are interactive computer programs which are designed to emulate the reasoning process of a specialist or expert in a particular domain. These knowledge based systems contain either declarative knowledge or procedural knowledge in their knowledge base. The inference engine of the system in turn controls the processing of the information that is stored in the knowledge base. Together this provides the user of the system with the expertise and advice for a wide range of problems.

One major difference between knowledge based systems and conventional programs is the separation of the expert knowledge (the rules) from the general reasoning mechanism. Knowledge based systems use a variety of search techniques such as forward and backward reasoning, means-ends analysis, AO* graphs, and Hill-climbing. Although knowledge can be formulated to accommodate anyone of these techniques, care must be taken to insure that the appropriate search techniques efficiently and successfully matches the task at hand.

Frames are the most widely used primitives for knowledge representation in the field of Artificial Intelligence. Frames are capable of representing both general and specific knowledge, and representing both descriptive and prescriptive computations. Domain independent knowledge based system formulation languages provide the knowledge engineer with an inference mechanism from which a number of applications can be built. These systems may also provide knowledge acquisition and explanation modules to simplify the construction of knowledge based systems.

Three design-oriented knowledge based system applications were reviewed; namely AIR-CYL, DACON, and the CARTER System. AIR-CYL is a knowledge based system that is used to design air cylinders for a manufacturing system. DACON is a design for assembly consultation system that is aimed at tackling the problems of assembly rationalization, component optimization for handling and assembly processes. The Carter System is a knowledge based system that is used to design crankcases for engines without any intervention from the user, with the exception of supplying inputs and constraints.

Conclusions are drawn on the fact that knowledge based system tools are now available and that techniques are now perfected for the use in early applications. The development of knowledge based systems is recognized internationally and funding for these applications are now being committed here in the United States and abroad.