NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

A Knowledge-Based System for Integrating Design Tools

by

Raymond M. Sassine,
B.S.EE, M.S.EE, M.S.CS

A thesis submitted to the Faculty of Graduate Studies and
Research in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Computational Analysis and Design Laboratory

Department of Electrical Engineering
McGill University
Montréal, Québec, Canada
December, 1992

ISBN 0-315-87913-0

Canada

Abstract

The complete design of many electromagnetic devices requires the solution of a coupled problem. Typically, the coupling is in the form of magnetic/thermal, magnetic/structural, magnetic/electronic, or possibly a combination of several disciplines. Computer based tools exist for many of these engineering specializations but they are usually "stand-alone" and each requires an experienced designer to use it effectively. This requirement for an expert user places a major constraint on the design cycle, and a lack of communication between the various experts can result in major design errors.

This thesis proposes a software architecture that is capable of providing loose coupling between currently available design tools and of absorbing new tools in the future. The structure provides an integration environment for a suite of design programs. The environment automatically allows the iterative solution of coupled problems by loosely coupling individual tools through a comprehensive database and organizing their execution via a rule-based control program. In order to effectively integrate a diverse set of tools and to define the kind of coupling between the various analyses, it is necessary that the input and output requirements of each tool be carefully defined. The BlackBoard for Computational Analysis and Design (BBCAD) is a hybrid knowledge-based system which uses the blackboard architecture, and generates a systematic method of integrating the "stand-alone" design tools, together with an automatic method of ensuring that, if a change is made to the design, all the relevant design tools are re-run.

Résumé

La conception de plusieurs appareils électromagnétiques requiert la solution d'un problème couplé. Typiquement, le couplage est sous forme magnétique/thermique, magnétique/structurelle, magnétique/électronique, ou parfois une combinaison de nombreuses disciplines. Des outils informatiques existent dans plusieurs de ces champs d'application, mais ils ne s'appliquent généralement qu'à une catégorie spécifique de problèmes et chaque outils requièrent l'intervention d'un concepteur expérimenté. Ce besoin d'un spécialiste contraint le cycle de conception et un manque de communication entre les différents experts peut être la cause d'erreurs de conception importantes.

Cette thèse propose une architecture pour logiciels, capable de fournir un couplage sans contrainte entre les outils de conception présentement disponibles, ainsi que d'intégrer de nouveaux outils à venir. La structure fournit un environnement d'intégration pour une multitude de programmes de conception. Cet environnement permet de résoudre automatiquement, par itérations, les problèmes couplés en joignant sans contrainte les outils individuels à l'aide d'une base de données exhaustive et en orchestrant leur exécution par un programme de contrôle basé sur des règles. Pour intégrer efficacement un ensemble d'outils et définir le type de couplage entre les diverses analyses, il est nécessaire de détailler soigneusement les entrées et les sorties de chaque outil. La méthode "Blackboard" d'analyse par calculs et de conception automatique (BBCAD) est un système hybride à base de connaissances qui utilise l'architecture "Blackboard" et génère une méthode systématique d'intégration des différents outils de conception. Simultanément, cette méthode garantie qu'une modification à la conception sera suivie par l'exécution des outils pertinents.

Acknowledgements

I am extremely grateful to Professor Dave A. Lowther, my research advisor, for his friendship, patience, encouragement, constructive criticism, and of course, technical support. I am honoured to graduate under his guidance.

I wish to express grateful acknowledgement to Prof. P.P. Silvester for his inspiration, the staff of Infolityca Inc. for their support, especially my ex-roommate the NewFoundlander Craig Brett (alias Craigy), and Allan Kobelansky, the research director of Sun Canada Microsystems Inc. in Montreal, for his willingness to provide software and other materials, and of course, his friendship.

I wish to acknowledge two influential people in my life. My father, Maroun, departed on February 4 1992, the time of writing up this thesis, and my uncle, Tannous, on January 2 1989, one month before my qualifier. I could not thank you enough for what you have done for me.

To my mother, Souad, don't worry; I will get you to this country. To my aunt, Salma, I thank you for being a mother too.

The support of the Natural Sciences and Engineering Research Council of Canada, Fonds pour la Formation de Chercheurs et l'Aide à la Recherche, the Centre de Recherche Informatique de Montréal and Bell Northern Research is gratefully acknowledged.

Table of Contents

CHAPTER 1
Introduction

1.1 Introduction

Many large, complex electrical engineering design tasks require the use of a considerable number of existing inter-related design tools. They typically perform computational functions such as simulation modeling, i.e. the process of exercising the model and obtaining some results, dynamic analysis, and optimization. Many of these design tools are "stand-alone" and require an experienced designer to use them effectively. It is becoming more and more important that a systematic method of integrating these design tools be developed, together with an automatic method of managing the design process, and ensuring that, if a change is made to the design, all the relevant tools are *re-run*. Thus integrating the different parts of the design process into a fully automated system can often increase the quality of the results and avoid the need for human intervention.

In order to integrate a diverse set of electrical engineering design tools effectively and to provide a form of weak coupling between the various analyses, the input and output requirements of each tool must be carefully defined. Because of the complexity of the interrelationships among the design tools, numerous delays and errors occur during their integration. These delays and errors can increase costs, cause scheduling crises and reduce design quality. The result is that the current approach for the solution of electrical engineering design problems has been the combination of fully automated tasks, and a set of symbolic or heuristic tasks, that are performed by the engineer.

Recent advances in *knowledge-based system design* have provided some of the tools

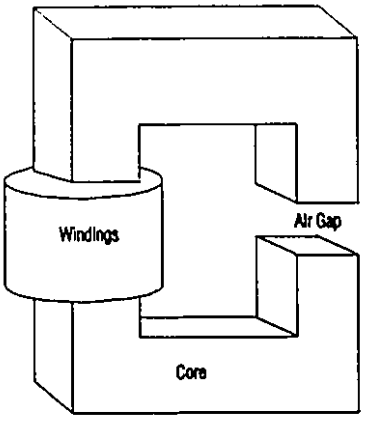

Figure 1.1: C-core Magnet model

necessary to overcome these difficulties, thus solving electrical engineering design problems. A hybrid Knowledge-Based System (KBS) is most convenient for this purpose. There is a large set of KBS architectures being developed and tested [Maher 88; Mittal and Araya 86; Sriram et al. 89; pp.15-16 of Dym and Levitt 91], the most notable of which is a hybrid system known as *The Blackboard Model Architecture*. This thesis is intended to investigate the application of this architecture to the integration problem in electromagnetic device design. A C-core device model, shown in Figure 1.1, will be carried right through the treatise to help illustrate the concepts.

1.2 The Engineering Design Process

Engineering design is the creative process which follows the identification of a need for a device that satisfies the design requirements [Gregory 66], for example, an electrical machine - motor, generator, transformer. The process varies widely depending on the size, maturity and specification of a particular need. As shown in Figure 1.2, the traditional subparts of the engineering design process are a combination of encoded routines, and symbolic problem-solving. Thus, the process of designing requires the use of many

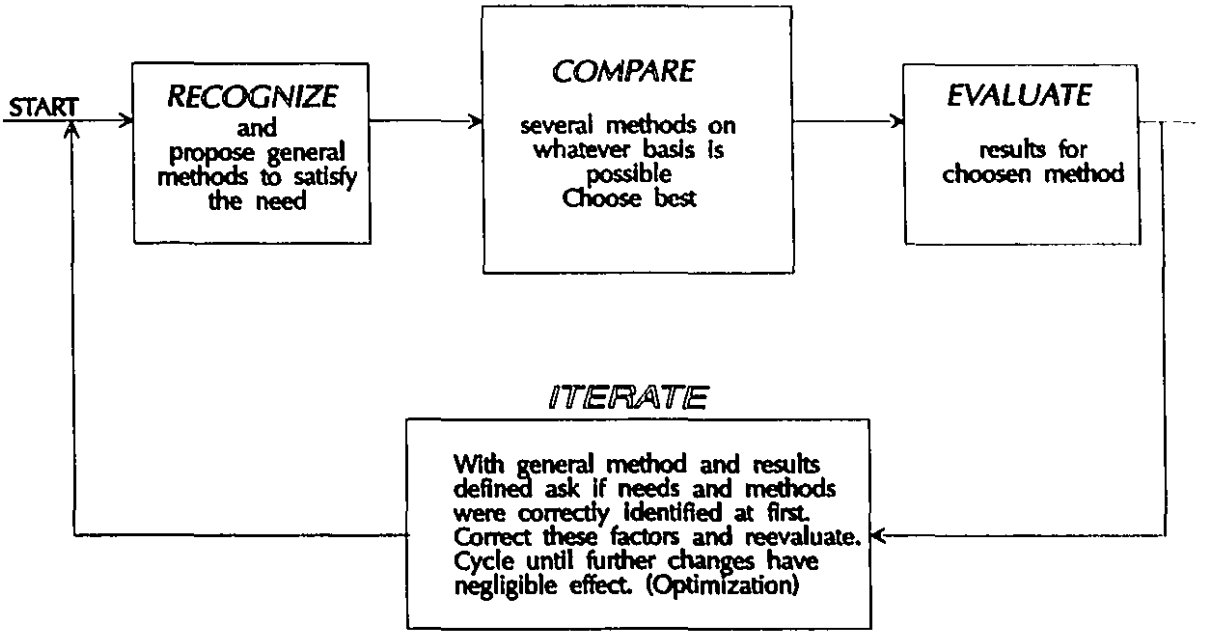

Figure 1.2: Engineering Design Process

strategies. The general pattern in engineering design (Figure 1.2) consists of preparation, gathering information, incubation, verification and finally communication [Gibson 68]. It is an iterative and incremental process as demonstrated in Figure 1.3. The design task consists of generate-test-refine processes. It involves the modification, refinement, enhancement and combination of existing solutions into a new hybrid solution that satisfies the given design device requirement [Gero et al. 88]. The designer uses a variety of cognitive operators to generate a design, test it under specific conditions and refine it until a stopping criterion is reached. The design process can be characterized as a cycle of levels in which a new model of the design or a part of the design is generated at each level (Figure 1.4). Although the design process proceeds in stages (Chapter 1 of [Brett 90]), these stages are not clearly delineated nor are they necessarily performed sequentially. Thus, the design problem is formally a search problem in a large space for objects that satisfy multiple constraints.

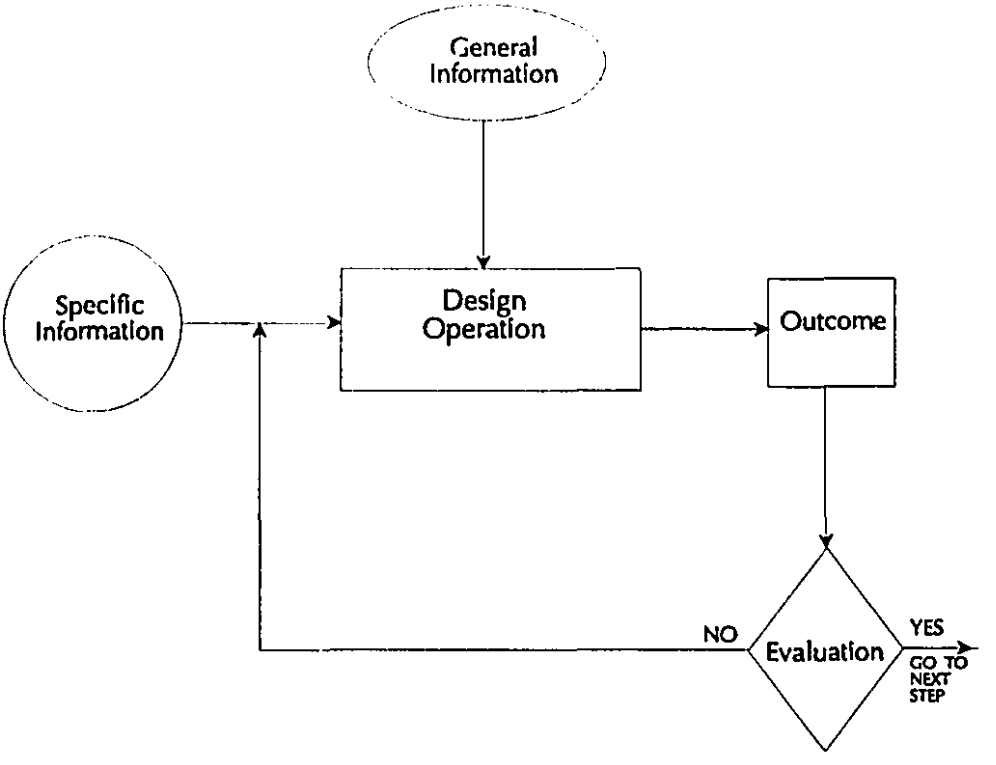

Figure 1.3: Basic Module in the Design Process

Engineering design is a problem-solving activity of optimization but in an under-specified system. It is inherently iterative and interactive, utilizing analysis, synthesis and evaluation [Rychener 88]. In an interactive engineering design system, the program module and the designer (user) work together on the design synthesis task which requires methods of quantitative and qualitative reasoning [DeMori and Prager 90; Brett at al. 90]. This dictates that they share a common knowledge of the goal of the design task, the strategy they are pursuing to achieve the goal, and the current state of their effort to execute the strategy. For the development of such a program module, the designer must have computer models of human thinking (cognitive model): what he or she knows, what he or she is trying to find out, and what he or she is trying to do. Also, a discourse must be planned by which the program module may gain information from the user, and relate to the user the important knowledge developed. It is crucial for the success of this

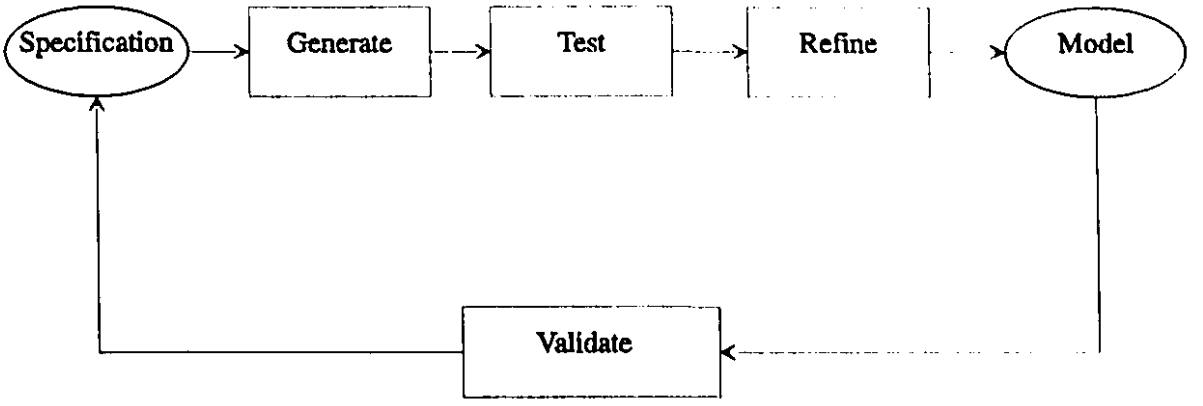

Figure 1.4: The design cycle

approach that the design strategy being employed by the program module be comprehensible to the user. It is essential that there be a real-time critic of the design being developed, and of the design strategy being employed by the user. The system should also be capable of interactive justification and explanation of design decisions and analysis results.

Design knowledge is dynamic. As a design progresses from the early stages of requirement formulation towards the detailed design, the knowledge grows not just in volume but in complexity. Growth in complexity means that dependencies, constraints and interrelations missing at early stages are inserted throughout the design information. Design knowledge also undergoes changes in status, quality and consistency. Status defines knowledge by labels such as: assumed, factual, validated, proved, checked, etc. Quality change means that the knowledge may pass through different levels of confidence, from preliminary estimate to final analysis results. Consistency refers to the different sets of constraints that the design must satisfy [Saldanha and Lowther 86; Brett et al. 90]. Thus, engineering knowledge must meet a high standard of integrity and robustness. It must satisfy constraints imposed by the laws of physics and chemistry, and must also conform to the engineering standards specific to each discipline.

Knowledge-based systems provide some of the capabilities outlined above, and can make significant contributions toward the automation of electrical engineering design. A description of KBS architectures can be found in Appendix A.

1.3 Interdisciplinary Research and Development

More recently, the formalization, representation, and manipulation of knowledge in computers has made it possible to construct knowledge-based design systems [Coyne et al. 90].

Over the last few years, many researchers in Artificial Intelligence (AI) have come face to face with the task of trying to formalize problems in order to produce intelligent problem-solvers. Design problems have proved especially difficult. Simon [Simon 69] characterizes design as an ill-structured problem, i.e. one which does not have a clearly defined algorithmic solution, which is difficult to formalize and thus difficult to solve. Advances in computers and engineering design methodologies led to an extensive use of computers in Computational Analysis and Design (CAD). However, the involvement of computers in engineering design has been very much limited to fast "number crunching", i.e. to algorithmic solutions, such as finite-element methods. However, developments in AI techniques, in particular the KBS technology, have made it possible for computer programs to simulate human expertise during the problem-solving process. The KBS intelligent behaviour is derived from the incorporated reasoning capability, intelligent search techniques, and the ability to monitor and evaluate performance and alter a course of action to optimise a design. It is these capabilities, coupled with the heuristic and "non-algorithmic" expertise knowledge, which offer a solution for ill-structured design problems.

Research in knowledge-based systems and design tools has generated a large number of systems. Among the first applications of knowledge-based systems in design

has been the use of expert systems as design analysis and synthesis tools [Rychener et al. 84; 86; Maher 88; Dym and Levitt 91]. While much of the work in this area is currently focused on single domain advisory systems, a small number of integrated computer-aided design systems with coordinated multiple domain experts are emerging [Dym 85].

The DARPA DICE [Sriram et al. 89] project uses the blackboard architecture approach to achieve communication and coordination of problem-solving between designers. DICE is oriented toward engineering design environments and is used to assist experts from several domains who need to collaborate in the design of new products. Once the experts agree on a particular design, the design is posted onto a database. HOBS [Carter and MacCallum 91] is a software architecture based on a blackboard model with knowledge sources related in a hierarchy as a means of supporting design co-ordination. COCASE [Gentilhomme 91] is a expert system based on the blackboard architecture to design magnetic relays.

1.4 A Role for Knowledge-Based Systems

Artificial intelligence has been used in knowledge-based systems as an interactive means of gathering and controlling information with human operators [Hayes-Roth et al. 83]. It originated with the idea that if one could simulate most of the functions of the human brain on a computer, then scientists might understand the human brain. KBSs were the natural applications of such an activity.

KBSs employ human knowledge to solve problems that ordinarily require human intelligence [Simon 69]. They simulate expert human performance and present a human-like facade to the user. Human knowledge consists of elementary pieces of "know-how", thus applying a significant amount of knowledge requires new ways of organizing decision-making fragments into useful entities. Knowledge-based systems collect these fragments

in a knowledge base, then access it to reason about specific problems.

AI and its subfield, KBS, are equipped with the ability to represent *knowledge* in different forms. Practical results can be attributed to the design and use of KBS, in that they can reach a level of performance comparable to that of human expert "know-how" in some specialized domains such as Engineering Structural Design [Rychener 88; Kowalik and Kitzmiller 88], Fault Diagnosis [Talukdar and Cardozo 88], Medicine [VanMelle et al. 81], Science [Engelmore and Terry 79], etc. [Barr and Feigenbaum 81,82; Cohen and Feigenbaum 83]. What distinguishes a KBS from a conventional application program is the model of problem solving [Newell and Simon 72]. Application programs make use of specialized problem-solving knowledge and many of them reach high levels of performance. The model of problem-solving in KBSs is based on a separate entity or knowledge base, while in an ordinary application program it appears implicitly, as part of the coding of the program. As a result, KBSs instead of being programmed to follow step-by-step procedures, follow a few general procedures which are generally opportunistic rather than deterministic. Facts, heuristics, models and other general knowledge about solving a particular class of design problems are encoded and stored in the computer's memory. In order to solve a specific design problem, the computer uses facts provided by the user plus the design domain knowledge and general problem-solving procedures to find and apply specific solutions. The domain knowledge is acquired from a domain expert who provides the key to expert performance.

The KBS paradigm has provided a whole realm of potentially useful tools to enhance the human-computer interface, but what is more important, its use as a controller of complex engineering programs provides a way of using formalism and conventional algorithms in a Computational Analysis and Design (CAD) system whilst keeping them separate from heuristic procedures representing informal knowledge about engineering design. The criteria for improved CAD design include several complex notions: multiple

levels of representation of a design [Nicklaus et al. 88], known as abstraction, facilities for meta-level reasoning about design strategy [Buchanan and Shortliffe 84], etc. The structure that controls this system will have to be versatile and robust. One candidate structure is the *blackboard*, a feature of the HEARSAY-II speech-understanding system [Lesser et al. 75; Lesser and Erman 77; Erman et al. 80]. A background on the blackboard architecture and an overview of variations in the architecture can be found in Appendix B.

1.5 A Role for Blackboard Architecture

The blackboard is a paradigm that allows for the flexible integration of modular pieces of design code into a single problem-solving environment. It provides a way to organize a large amount of knowledge into an intelligent program and is based on the paradigm of several experienced designers working together on a problem. Each designer can "see" the current state of the problem as it is described on the "blackboard" and can make a contribution to the problem solution if his or her knowledge applies to the current state. In software terms, the "blackboard" becomes a shared memory area, and the expert designers are replaced by computer based design tools. In practice, each tool monitors only a small region of the blackboard and is activated only when entries are posted in that region by another module. The central issues in any engineering design problem are: what pieces of knowledge should be applied, and when and how to apply them. The blackboard model answers these questions by separating the problem-solving framework into three major components [Nii 86(a)]: the blackboard data structure, the knowledge sources (KSs), and the blackboard control.

In the context of a CAD system for electromagnetics, entries on the blackboard would be design modules suggested as refinements of a particular level of representation, e.g. the analytical module, the finite element analysis tool, the structural module, the

thermal module, etc., in the design of the C-core magnet. Entries can also be intermediate results generated during design problem solving [Hayes-Roth 85 (a)]. *Knowledge Sources*, in the present terms, are the design knowledge modules which govern the search for, or design of, device units and their combinations into partial designs. The blackboard itself serves a unique purpose in separating the different kinds of design modules from each other, whilst allowing them to communicate with, and influence, each other via blackboard entries. The control mechanism which governs blackboard activities is basically a scheduler which looks at a list of "knowledge source activation records" (KSAR) to decide which action to invoke next. In this sense the system functions like an operating system for a virtual machine. However, triggering a design module does not lead to the immediate activation of the appropriate patterns in the design module. This allows the control mechanism to explore the range of possibilities for action, construct a priority list, and take a global view of the activities of the design process.

The implementations of blackboard systems vary considerably, but they all exhibit the same major architectural constructs: an explicit global data base, i.e. blackboard, KSs which affect and react to changes on the blackboard, and a control mechanism. The type of control mechanism adopted depends on what it is intended to determine from the design system. Blackboard systems have been implemented for problem domains ranging from speech understanding [Erman and Lesser 75; Lesser et al. 75] and protein crystallography, CRYSALIS [Terry 83], PROTEAN [Hayes-Roth et al. 86] to new applications in model-based materials processing, DECADE [Bañares-Alcántara et al. 87, 88(b)] and share the same components described above. However, they differ in the details of their component structures and functions. Differences among blackboard systems involve control regimes and mechanisms for determining when KSs are executable.

1.6 Scope and Motivations of this Research

The requirements for the design of a complex electrical engineering system are diverse and often contain conflicting goals. The complete design of many electromagnetic devices is a difficult problem which accounts for magnetic, structural and thermal aspects, requiring the solution of a coupled problem. Typically, the coupling is in the form of magnetic/thermal, magnetic/structural, magnetic/electronic, or, possibly, a combination of several disciplines. Design tools exist for each of these stages and require trained designers to use them effectively. Many of these problems may be described using computational analysis methods coupled with knowledge-based design techniques, which offer a simple and flexible way for introducing the diverse knowledge (e.g., geometric specification, material characteristics, problem specifications etc.), and an easy way for evaluating the numerical solutions and post-processing them. Currently, a project is under way to integrate tools into the design and analysis of magnetic devices in general, and into the design of electrical machines in particular, to minimize the level of human expertise, and to allow the production of designs with minimal, and preferably zero, hardware prototyping.

These research efforts have been concentrated in the development of a framework for integrating engineering design tools, to provide a loose coupling between the currently available tools, and to absorb new tools in the future. BBCAD (BlackBoard for Computational Analysis and Design) is a hybrid KBS based on the blackboard architecture model [Lesser at al. 75; Lesser and Erman 77]. The objective of BBCAD's development has been to investigate and evaluate the potential of KBS in machine design; to make it possible to take full advantage of existing design tools; and to allow the automation of a larger portion of the design process.

Whilst the thesis so far has centred on the use of the blackboard architecture to integrate high level software tools, it can also be applicable at a much lower level. For

instance, the conventional architecture of a finite element magnetics analysis system usually consists of a set of relatively large scale design modules, each of which is relatively difficult to modify. The BBCAD architecture allows each of these modules to be broken down into their constituent parts, interacting through the common database. Thus an automatic mesh generator might be a separate process in the pre-processing phase of a solution. Changing the mesh generator would simply mean replacing that small module, or having more than one module in the system, and depending on the design requirements, the appropriate module will be invoked. Alternately, several mesh generators, each having specific parameters in terms of the quality of the mesh generated and the cost of generation, might co-exist in the blackboard environment. Thus the system leads to more maintainable and more easily expandable software systems.

The following are some of the tools that exist:

- The Electromagnetic Design System (EDS) [Saldanha and Lowther 86] is a knowledge-based expert system aimed at automating the computer-aided design of electromagnetic devices such as transformers, actuators, and motors. EDS is a programming environment, and considers the different categories of knowledge in design. The mathematical model of a device provides one category of knowledge, and its representation and function are implemented by a sub-system of EDS called the Computer-Aided Design Algebraic Constraint System (CADACS) [Saldanha 88]. Another sub-module of EDS is the interval mathematics package module (INTSYM) [Brett 90]. It allows the user to put practical limits on certain parameters enabling the program to deduce the valid design space of the device.

1.7 Approach and Contributions of this Research

The emerging fields of artificial intelligence and knowledge engineering [Genesereth and Nilsson 87] offer means to carry out qualitative reasoning on computers. These techniques allow us to model the intuitive knowledge, judgement, and experiences that expert designers use, and to integrate them with the quantitative tools. What is needed is a **knowledge-based** program that encompasses knowledge associated with not only our understanding of the design tools and features of the product, but also the integration issues of new design tools and the role of the tools in the design process.

The approach taken in this thesis is, first, to design a blackboard software architecture, next to develop several applications to test the system, and finally, to examine the integration of different electrical engineering design tools, to make sure that the system is capable of providing a loose coupling between currently available design tools, while absorbing a wide range of applications. The most important goal of this research study is to evaluate the feasibility of successful application of knowledge-based systems to the engineering design area. This goal is two-fold: to propose methods of solution for tasks that traditionally have been approached "by hand", and to integrate those solutions with the results of the algorithmic parts of the problem. Although the BBCAD uses the blackboard model, the basic architecture, knowledge representation, and knowledge utilization techniques differ from other blackboard systems. The differences can be attributed to many factors: the nature of the problem (electromagnetic device design); implementation language; design constraints; quality and amount of available knowledge; and, last but not least, the designer's problem-solving strategy.

This research contributes to:

a) The field of software engineering in general, as well as to

b) The application of the problem solving mechanism to the domain of electrical machine design.

BBCAD itself is a system consisting of a single problem-solving module. The major contribution of BBCAD is not in any extension of the technology of the blackboard architecture, but in the unification of blackboard technologies with a development system for high-performance design applications. Thus, the focus is on generalizing control capabilities to provide a means of integrating dramatically different systems via blackboards.

More general issues were investigated:

- Open-Ended integration (Numerical/Symbolic integration).
- Integrating learning and problem-solving into a unified process.
- Maintaining consistency of data structures (Integrity).
- Identification of knowledge-based activities.

1.7.1 Claims of Originality

In this thesis the following original contributions are made:

a) Software architecture which is founded on knowledge-based systems and the blackboard model.

b) The architecture is capable of providing loose coupling between currently available design tools.

c) The architecture is capable of absorbing new design tools into the system.

d) There is no restrictions on the kind of tools the architecture can handle.

e) The architecture provides a problem solver that can search the design space more thoroughly in short time.

f) The search is guided by the knowledge of the problem space structure.

g) A tree structure which identifies the many ways of slicing the knowledge in KBS in design.

h) The architecture is capable of integrating KBS methods with existing optimization design tools used in other engineering disciplines.

i) The architecture automates the design process of an electromagnetic device.

j) A multilayer blackboard architecture that could be used recursively.

k) A software approach which executes the original existing design tools as independent processes.

1.8 Organization of this Thesis

This dissertation is organized as follows:

Chapter one sets the scope of this research thesis, and presents some related work in engineering design. Chapter two states the purpose of this research and its relation to electric machine design. It explores a frame-oriented approach to design tool integration for allowing coordination and communication. It also demonstrates how the concepts of KBS and blackboard architecture can be combined to develop a hybrid knowledged-based system.

Chapter three presents the knowledge organization structure and the functionality of the blackboard framework BBCAD. Each of the modules constituting BBCAD is introduced along with an explanation of the system's overall control, and internal control of the module. Chapter four discusses the decision adopted by the blackboard controller on the next step to take in the design process.

Chapter five emphasizes the tool integration aspect. The development is initiated by transmission of parameter specifications of the C-core magnet, constructing the design

alternatives, i.e. design synthesis, evaluating the design synthesis, i.e. design analysis, and finally applying analysis optimization techniques to improve the design. Two different optimization techniques are presented in automating the design process of the C-core device.

Finally, the last chapter summarizes the most important facts. From these facts, a set of remarks and conclusions is drawn from which the contribution of the work can be judged. Also, some recommendations and suggestions for future research in this and related areas are given.

Appendix A attempts to provide a background on knowledge-based systems in engineering design and cites the advantages of using such systems for performing symbolic manipulation procedures that are used in BBCAD. Appendix B provides a background on the blackboard architecture and reflects its use in engineering design. An overview of variations derived from the formal descriptions of a number of influential blackboard architectures is also presented. Appendix C reviews many possible implementation procedures which are available inside the KBS domain. They include general purpose programming and representation languages, and skeletal systems. Appendix D demonstrates how to calculate priority ratings of a design knowledge module. Rule constructs that are generated from the design module are represented in Appendix E. Appendix F deals with more specific details of the implementation of BBCAD. It also describes the front-end user interface of the whole system.

CHAPTER 2

An Architecture for Engineering Design

The objective of this chapter is to state the purpose of this research and its relation to electric machine design. It examines a frame-oriented approach to design tool integration for permitting coordination and communication. It defines the design process for the general class of electromagnetic devices and suggests that there are several hierarchical levels of abstraction of the physical device. It also provides a background on knowledge-based systems in engineering design, the blackboard architecture, and how these concepts can be combined to develop a hybrid knowledge-based system for design, such as the one on which BBCAD is based.

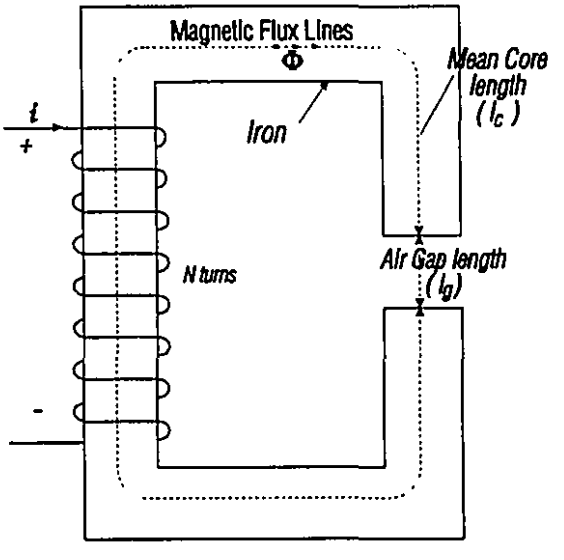

Figure 2.1: C-core Magnet

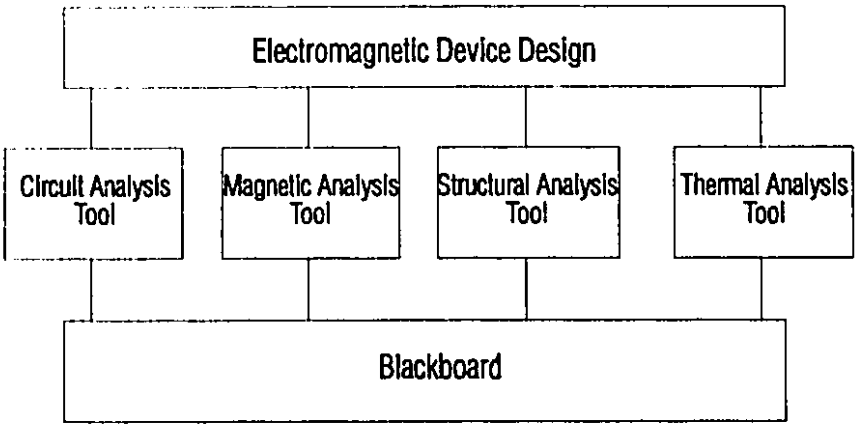

Figure 2.2: C-core design architecture

2.1 A Coupled Problem in Electromagnetic Design

As in the design of most complex electric machines, the designer of the physically simple C-core device attempts to satisfy a complex set of interrelated design constraints. The designer starts with the specifications for the C-core and proceeds to compute the other parameters to obtain a final design. Once all the parameters for the initial C-core have been assigned values, the designer executes an analysis tool in order to evaluate the quality of the proposed design. The results are used as a basis for the design modifications before beginning the next iteration of the synthesis procedure. If the analysis parameters are within the admissible values and in some cases, if a cost function is minimized, the design process terminates successfully. This rarely occurs on the first iteration, since varying the physical dimensions of the C-core affects the magnetic requirements, and in turn the thermal demands. Thus the complete design of the C-core requires the solution of a coupled problem. Coupled with experience, the designer allows intelligent modifications to be made for the next phase of design. The designer decision-making ability is usually limited to just the few designs he could produce by hand.

The design system is outlined in Figure 2.2, and contains a device model, a set of design analysis tools, and the blackboard database which consists of a set of datastructures.

These structures provide a uniform method for dealing with diverse forms of knowledge, and contain information relevant to a particular design tool. For instance, the structure contains the conditions under which the design tool (program) should be activated. These conditions may be specified in two parts; the first is the context in which the tool is relevant, the second is a set of pre-conditions which have to be satisfied. For example, the context for a magnetics analysis tool includes the fact that a geometric description of the device must be present, along with a specification of the materials used and excitations expected. The context might be fairly specific in that the problem definition might well indicate that the magnetostatic solver is applicable. A similar context might well apply to a thermal analysis system. Thus if the context matched that specified for the particular design tool, the program could be executed.

In the coupled problem scenario, the context might suggest that a magnetics analysis problem could be executed because all the relevant input data is present and the program has not been executed previously. If the coupling is to a thermal analysis, its context would require that a description of the losses be present. These might only be generated by running a magnetics analysis. Thus the magnetics analysis would execute, the losses needed by the thermal program would be written into the blackboard and the thermal program might now execute. The result of running the thermal program might be a change in the temperatures of the device leading to a change in the magnetic and electric properties. This change might well be one of the preconditions for executing the magnetics analysis and thus it would re-execute.

By this process, an iterative loop can be set up without the user specifically requiring that it occur. Eventually, the system should converge to the solution of the coupled problem. However, it is possible that the process will be divergent and thus a controller is required to monitor the changes happening on the blackboard and to ensure that these are, indeed, leading to a convergent solution.

2.2 Issues in Design

In order to illustrate the notions that will be presented in this thesis, the simple problem of designing a C-core magnet, shown in Figure 2.1, is carried right through this dissertation. The complete design of the C-core device requires the solution of a coupled problem, where the coupling is a combination of several disciplines: magnetic, thermal, electrical, etc. The goal is to optimize the design of the C-core by altering certain variables, e.g. the shape of the core at the air gap, such that the change of flux density in the air gap is infinitesimal.

The design process is an iterative mechanism of evaluation and modification, i.e. analysis and refinement (refer to Chapter 1). The iterative process continues until at least the specification criteria of the design, or a stopping criterion for a design module, are met, thus giving a final design. The goal of the design process is twofold: the first objective is to define the full set of parameters which describe the structure and operation of the C-core device, the second is to optimize the automatic design to most closely reach the design goal. BBCAD uses the following problem solving strategy: "To run a design module, if all the inputs to the module are present, execute the program module and return the results. Otherwise find all the required input variables whose values are not known and consult the spaces of design for starting values. Then, if all the needed inputs are still not present but there is another design module, the output of which will provide the missing inputs, run that module first." The design module must contain all the necessary information for it to integrate and to execute in the proper order.

As more and more design modules are added, the choice of which program to execute next is not a simple task. A tool projected on the design space can increase the dimensionality. The design space is the space of all possible designs of the device and their global parameters (e.g. structural which includes the core, the gap, and the windings, magnetic which includes the flux density, electrical which includes the current, etc.)

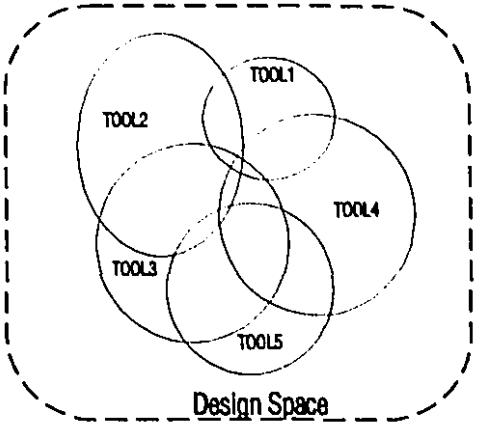

Figure 2.3: Design space of a current design

(Figure 2.3). The design can be viewed as a search of a multi-dimensional space of possible designs. In effect, it is a searching of a space of all possible structures for the one which most closely satisfies the specifications. The problem of design is to narrow the search space as fast and as effectively as possible.

One basis on which to make the choice of which tool to run is the amount of new knowledge which might be provided by a particular tool. If several tools can execute, then the one adding the most new knowledge to the design space should be chosen. There is no reason why all the tools should be linearly independent; it is entirely possible that several tools capable of doing the same, or a similar, job might be included in the design space. Each tool would have its own cost in terms of resources needed, execution times, and accuracy of results as well as output information.

Lowther [Lowther 89] stated that design is the process by which the complete structure of a device is defined such that it operates within the range detailed in the specifications. In the issue of integrating diverse design tools into the life-cycle engineering environment of the design process, two pragmatic requirements are placed on the complete design:

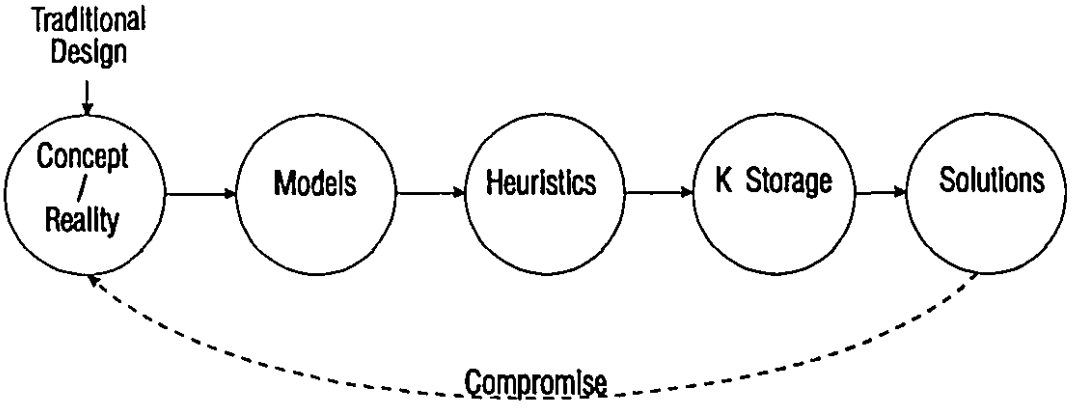

Figure 2.4: Simple abstraction of the design process

- Knowledge of the problem space structure, and the datastructure of the design module and the device.
- Guiding the search in the design space.

The approach must therefore consider the large number of tools that support various phases of the design process. The diversity of such tools makes them hard to integrate into an environment in such a way that they can support design coordination and can communicate with each other.

One simple abstraction of the design process for an electromagnetic device is depicted in Figure 2.4. The objective is to provide effective automated support for the design process. To determine requirements, models of reality/concept are constructed from traditional design. These models are intended to abstract from the concept the essence of one or more aspects of an existing or proposed design, and focus on the knowledge architecture. Some of these models are general statements of the requirements; others are very detailed. From models of design systems, one can extract heuristic rules that should govern the design proce-- Based on these rules, one can plan the design knowledge modules. These modules are precise statements of tool interactions, of data relationships, and of constraints and conditions, and their architecture become the

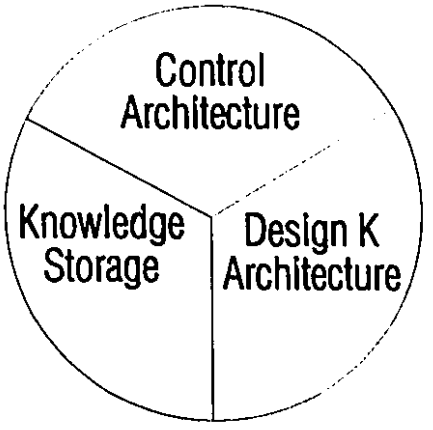

Figure 2.5: Views of knowledge

foundations of knowledge-base oriented automated solutions. One measure of these solutions is the extent to which they compromise reality. The truer the models that are used in the design process, the lower the probability that the resultant device design solution will force compromise.

2.3 A Framework for Integration

There are three distinct views of knowledge in the BBCAD architecture, shown in Figure 2.5. The integration problem of the design of an electromagnetic device is explored in the context of the following architectural characteristics:

- A knowledge storage and communication area serves as the repository for all design knowledge.
- The knowledge is organised in a way that allows easy representation of all the existing design tools involved in the problem solving process and their interrelationships, and better cooperation among the tools.
- The BBCAD environment allows diverse types of existing tools to be integrated regardless of the idiosyncrasies of their operation. It also allows new tools to be added

with minor modifications to the tools.

- The BBCAD environment has at least a primary notion of the new tool and its functions in relation to other tools such as scheduling, multilevel invocation, and concurrency.

The decision to integrate a design tool, i.e. design knowledge modules, is not merely a data conversion issue. Integrating a tool involves resolving such issues as the nature of the designer-tool interface, the tool's input and output, and the policies one wants to enforce on tool usage. To interface a new tool to the BBCAD structure, requires that a datastructure be created associated with the design knowledge module and describing its contributions to the blackboard space, its context and its preconditions. This integration process requires an understanding of the design and analysis variables, and functions of the tool, i.e. mapping analysis space to design space. The analysis variables refer to the variable values used in the analysis to calculate the analysis function values, and the design variables are comprised of some subset of the analysis variables. Also, some level of data translation program is needed. An expert designer supplies qualitative descriptions of variable parameter (input/output) dependencies, specifying which direction an input parameter changes will yield in which direction the output parameter will change. A useful structure to use for the underlying datastructure of the blackboard, from a point of view of magnetics design coupled with other analyses, is one of the neutral file structures currently widely accepted, e.g. IGES [Smith et al. 83], where the object of standardization is not only the data format for information storage, but (implicitly or explicitly) also the command structure of the systems that access this information. This minimises the amount of data translation needed since many numerical programs can read and write several different neutral files. A neutral file provides a projection of the tools' I/O onto an intermediate space. Thus all changes made to the parameters of the C-core device are re-written into a neutral data file, where data exchange is carried out. This neutral file

contains the coordinates of the data model for this specific C-core magnet. In order to effectively integrate a diverse set of tools and to provide a coupling between the various analyses, it is necessary that the input and output requirements of each tool be carefully defined.

One effective approach to design tool integration is based upon several assumptions:

- Different design tools need to view the same data in different ways.
- It is necessary for design tools to dictate or to constrain the limits and types of data entered by users.
- Since some data are interchangeable among tools and devices, it is essential that the data manipulated ought to be stored externally in a standardized format.
- It is desirable to maintain the independence and modularity of tools that already exist.

The effects of these assumptions will be discussed in more detail in the following sections.

2.4 Hybrid Knowledge-Based Systems

The knowledge-based approach to designing a system for any kind of task starts by determining what knowledge, i.e. facts and reasoning abilities, is used by human experts to achieve a solution. This knowledge is then encoded in data structures and procedures that represent the knowledge explicitly, and that are separate from the inference procedures that apply it to solve design problems. The inference procedures apply *knowledge* about a domain to the current state of the design in order to narrow down the design space, i.e. generate a new or "next" state, and draw conclusions. In a C-core device design [Magnet 85; Lowther and Silvester 86], for example, one important category of knowledge is factual data about the design that specifies the shape and some relationships,

i.e. a structural model and a simple algebraic model. Developing a knowledge-based system for magnetic design involves constructing a set of rules that summarizes various design tactics, as well as a general rule interpreter that applies combinations of these rules to solve individual design problems. A rule can be considered to be declarative or imperative knowledge of particular forms. Although the rules are often heuristic, and are used to search the design space, it is their application which reduces the size of the solution space.

In engineering design, there are knowledge modules that solve a part of a given device design problem (e.g. simulation, optimization, etc.). However, it is often neither convenient nor feasible to rewrite a program into another format only to make it compatible with the overall system design. Hybrid knowledge-based systems make it possible to take full advantage of the existing programs, and to allow the automation of a larger portion of the design process. Engineering design problems are well suited for hybrid implementations. A hybrid knowledge-based design system may be characterized by the following structures:

- Representation of the knowledge domain, such as rules, frames, semantic networks, object-oriented programs, or combinations of these.
- Problem-solving strategy: a control mechanism is generally required in order to limit the amount of searching in the design space and to point the process in the right direction. Design is seen as a problem-solving process of searching through a state space [Simon 69; Newell and Simon 72], where the states represent the design solutions.
- The implementation tools, that is the expert system shells and/or the programming languages used in the design, etc.

The components enumerated above are not completely independent. A choice of problem-solving strategy may influence the choice of knowledge representation, and, in turn, these two can determine the implementation utilized. In some other cases the

implementation tools limit the choice of the rest of the factors. The following characteristics have resulted in the construction of the BBCAD system.

2.4.1 Knowledge Representation

The ultimate product of engineering design appears in the form of devices, machines, services or structures, whilst the intermediate products are specifications, software, reports, or graphics. Knowledge and integration play a salient role in the engineering design process. Knowledge in engineering design can be described as interpreted statements about mappings between facts, specifically how new facts can be derived from existing ones [Coyne et al. 90]. The most explicit way to represent such relationships is as rules of the form A → B, i.e. "B is true if A is true". Sources and uses of these facts, authorization for their use and their release, and dependencies among their relationships are carried along with the knowledge itself. The need to operate on this knowledge base is the most distinguishing feature of engineering design in KBSs.

There are several methods used to represent design knowledge in a KBS [Barr and Feigenbaum 81, V.1, Chapter 3]. In [Coyne et al. 90], the authors presented three methods for defining design spaces other than rules. These comprise existing designs, descriptions of generic design, and procedures. The representation may be selected from various presentation methods developed for encoding facts and relationships that constitute design knowledge. The following methods are those which were found to be relevant in the development of BBCAD and are the most familiar:

- Frame: a data structure containing information relevant to a particular design module. This formalism exploits several useful ideas pertinent to engineering design: instantiations, inheritance, defaults, constraints, and attached procedures.
- Production rules: information about frames and instances can also be represented as IF-

```
<frame-name>
        (<options>=     (:print-name {string})
                        (:is {frame})
                        ...)
        <slot>
                <slot-facet>
                        (<options>=     (:print-name {string})
                                        (:explanation-string {string})
                                        (:default-values {value})
                                        (:constraints
                                                <option>=       (:lisp-type {value})
                                                                (:instance-of {value})
                                                                (:child-frame-of {value})
                                                                ...)
                                        (:when-modified {daemons})
                                        (:when-accessed {daemons})
                                        ...)
        <slot>
        ...
                <slot-facet>
                ...
```

Figure 2.6: Generic frame structure

THEN rules.

- Procedures: a set of Lisp functions and algorithmic procedures that cannot be defined easily in production rules.
- Semantic network: a graphical analogy for representing design modules and relations, and it could be the basis for reasoning. This formalism can make it clear how the properties of frames are inherited by subframes and instances.

Each of these will be looked at in more detail below.

2.4.1.1 Frames

BBCAD uses the *frame-based* representation approach in the design task, i.e. hierarchical abstraction and property inheritance [Minsky 75; Bobrow and Winograd 77; Hayes 77; Fikes and Kehler 85; Brachman and Levesque 85]. The *frame* structure is implemented to represent design modules, devices, and goals, as well as explanation and

information messages. A generic BBCAD frame is illustrated in Figure 2.6. In BBCAD, the knowledge storage consists of a set of data structures usually referred to as *GW-frames* [GoldWorks 87], each frame consisting of a set of *slots* (Figure 2.6), a slot is a structure that contains a variable-sized memory area. The slots represent functional or dimensional parameters of the design, or may link to more detailed frame structures indicating refinement of the design. Links may allow information from one frame to be passed to and used in another frame; this is referred to as "inheritance" [Bobrow and Winograd 77; Goldstein and Roberts 77]. Slots can contain rules about application area situations and actions to take under certain conditions. The slot may also have meta-slots, associated with it such structures are known as *facets* and contain additional functionality, often referred to as a *daemon* [Hayes 77], when a slot is accessed or instantiated. Daemons, alternatively known as the technique of "procedural attachment [Hewitt 69]", may be thought of as procedures which reside in a slot in a frame-based system. Such procedures are usually Lisp functions, attached to some data item and are typically used to perform actions that are linked to changes in a slot's value. In a sense, there is an equivalence between these structures and the background tasks or *daemons* in an operating system [Peterson and Silberschatz 85]. In much the same way as an operating system daemon is a suspended process waiting for an event, such as an interrupt, to occur before it activates, a procedural attachment is "waiting" for a particular state of the design system to occur. The daemon activates when the special event occurs, performs the job, and either terminates or suspends while awaiting another event [Barr and Feigenbaum 81, V.1, p.219-220; Winston 84, p.317-320]. These structures provide a uniform method for dealing with many diverse hierarchical forms of knowledge. The problem-specific dimension represents the frame or design module under investigation.

```
<rule-name>    (<option>*      (:print-name {string})
                               (:priority {number})
                               (:direction {:forward | :backward | :bidirectional})
                               (:sponsor {symbol})      ...)
          IF   (antecedent)
               <pattern>*      (<condition> <condition> ...)
          THEN (consequent)
               <pattern>*      (<action> <action> ...)
```

Figure 2.7: Generic IF-THEN rule structure

2.4.1.2 Production Rules

A production rule is an ordered pair of symbols with a left hand (LHS) and a right hand side (RHS), similar to the IF-THEN statement of Figure 2.7. The list of symbols on the LHS constitutes the premises or the conditional part (the antecedent), whereas the RHS symbols constitute the action part (the consequent). The conditions of a rule have to be satisfied in order for the rule to fire. Satisfaction is determined by matching against facts/assertions in the design space. The actions of a rule execute a series of operations that modify the state of the design space, thus causing a change of state in the design.

Production rules are a fundamental part of a *production system* [Hayes-Roth 85 (b)], which also contains an inference mechanism (an interpreter), and a context.

2.4.1.3 Procedures

A procedural representation of a model is a set of instructions that, when carried out, arrive at a result consistent with reality. Thus conventional equation solving maps in a straightforward way into procedures. Specific procedures are used to solve specific problems in the design space. Such procedures may, in fact, be complex analysis tools. Procedures are also executed in the action part of a production rule and are also implemented in the daemon formalism. Conventional programming techniques are very effective in implementing procedures.

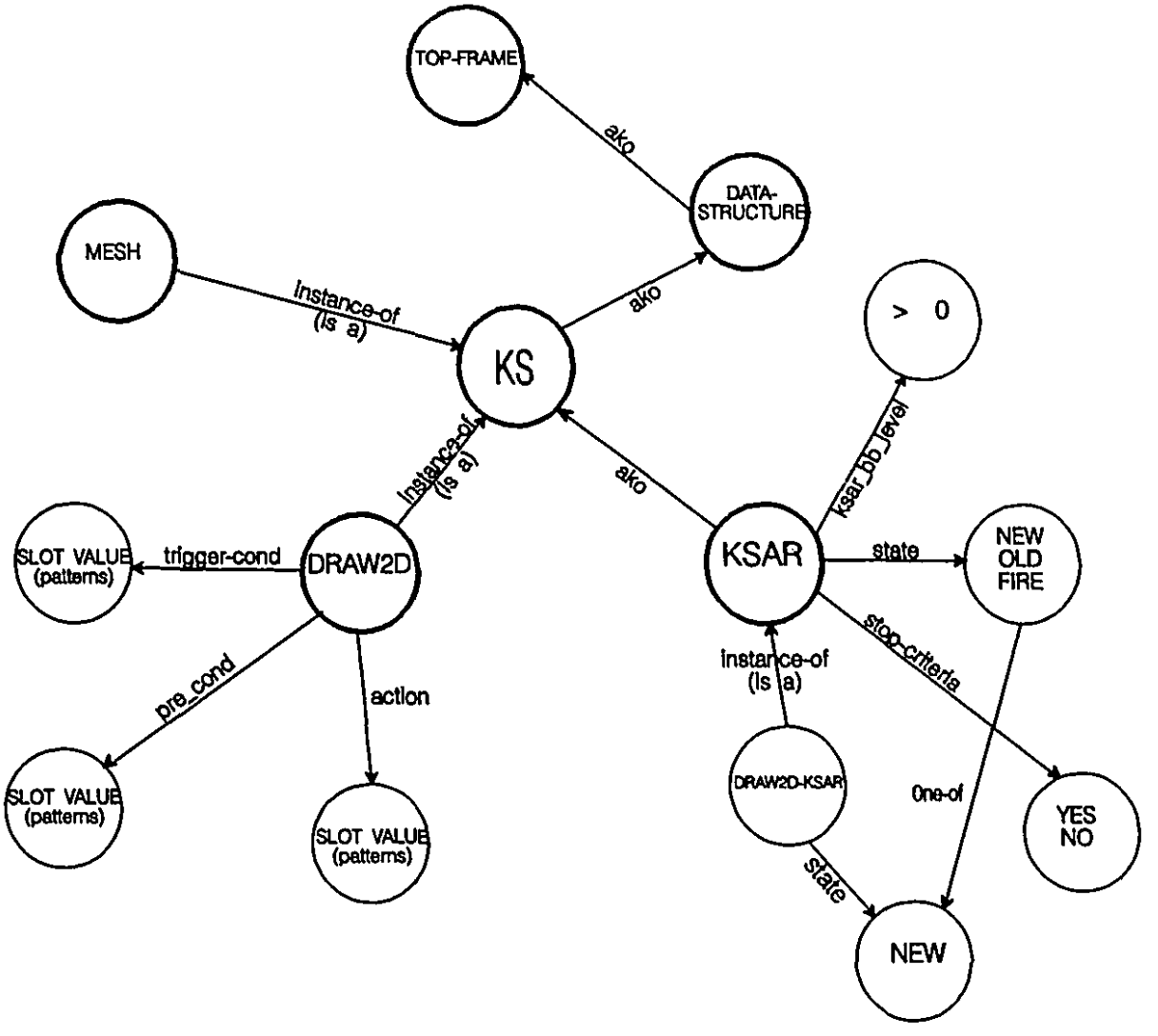

Figure 2.8: A semantic network for describing frames and instances of KS

2.4.1.4 Semantic Networks

A semantic network is a representation of knowledge that emphasizes relations [Quillian 68; Hayes 77]. The network representation consists of nodes and labelled arcs. Nodes usually represent objects, values, concepts, or situations, and the labelled arcs represent the links that indicate the relationships between them. It is a data structure that represents the declarative knowledge of a device. The structure is a graph in which the

nodes represent concepts (e.g., design tools) [Dym and Levitt 91, Chapter 1], and the arcs, which may be labelled, represent relationships among concepts. In a frame formalism, the idea of frames, instances and slots can be represented in a semantic network where frames, instances and values are nodes, and slot attributes are the labels attached to arcs joining nodes. In Figure 2.8, the "ako" label, which stands for "a kind of", indicates links between frames and child frames, while the "isa" link indicates that the object is an instance of the frame to which it is connected. Semantic networks are a convenient and natural way to represent descriptive knowledge about a device design, that is, design tools, devices, their properties and their relations.

2.4.2 Knowledge Abstraction

Abstraction is the decomposition of knowledge about design into hierarchies of design models (e.g., frames are abstractions of semantic network knowledge representation). One of the ideas that emerges in the discussion of symbolic representation is that of abstraction [Quillian 68; Goldstein and Roberts 77], in which the goal is to look at descriptions at various design levels of detail and try to abstract at a given level just that set of attributes which are needed to answer a certain question. It is a common practice to break down a complex design task into subtasks, each of which could then be used to formalize specific problem-solving methods. Whilst more abstract representations hold less information about the problem they are, however, easier to work with. Diagnosis [Buchanan and Shortliffe 84] may be considered to be a type of knowledge abstraction, in which specific knowledge patterns are classified as belonging to particular problem subclasses or classes. The representational abstractions, shown in Figure 2.9, are related, perhaps hierarchically, and design can be considered as a process in which there is free communications between systems abstractions, i.e. frames of design description, and levels of representational abstractions.

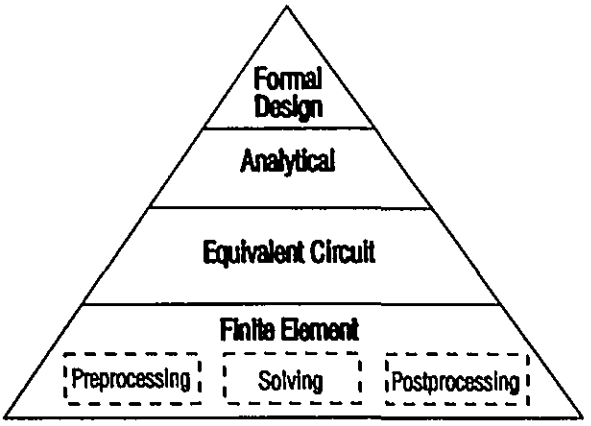

Figure 2.9: Multiple abstractions of electromagnetic device design

2.4.3 Problem-Solving Methods

As the name of this paradigm suggests, it contains the set of methods that can be used when performing a design task [Newell 62]. The methods describe how to manage the available knowledge and how to obtain the missing information in order to reach a goal state. Problem-solving can be viewed as the process of starting in an initial state and searching through the legal states, i.e. design solutions, for a device looking for the goal state. More detailed descriptions of a number of problem solving strategies can be found in [Nilsson 80; Rich 83]. In a "well-structured" design problem one knows the initial state, the goal state and the operators, which cause state transitions. From these one can systematically generate all the intermediate states, and hence one can theoretically develop a map of the entire design process. Not all engineering design problems are "well-structured". "Ill-structured" problems deal with heuristic programming [Newell 69] and occur for several possible reasons: the goal state is not stated explicitly, the design problem states are not discrete, the operators are not specified, the design space is unbounded, or time places additional constraints on the design. Using heuristic knowledge obtained from experience, some of the above reasons can be voided, e.g. operators can be specified or created, then the design process becomes well-structured.

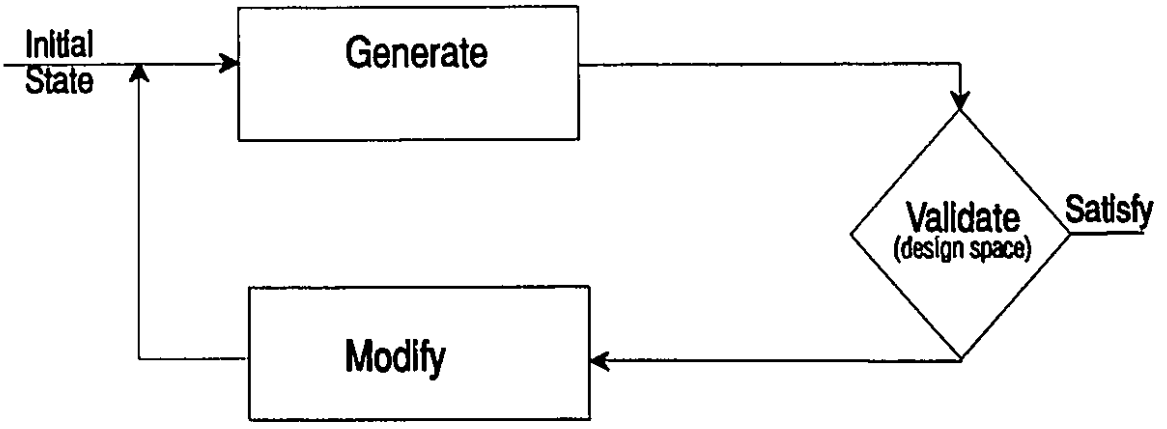

Figure 2.10: A problem solving strategy (Generate-Validate-Modify)

Perhaps the most basic problem-solving strategy is "generate-and-test" (Figure 2.10), where possible states are generated in a systematic manner, and then each one is tested to see if it is a goal state. A number of problem solving strategies are currently used in knowledge-based systems design. [Coyne et al. 90] propose several control tasks to model a design, including search strategies, goal satisfaction, failure handling, constraint manipulation, non-monotonic reasoning, multiple knowledge sources, multiple abstractions, and multiple control levels. For further details on these, the *Handbook of Artificial Intelligence* [Cohen and Feigenbaum 83] is recommended.

2.4.4 Conclusions

The above methods are by no means a complete set for building knowledge-based systems for engineering design. They were studied for their appropriateness in the development of BBCAD. Given the research interest in expanding engineering design to include qualitative as well as quantitative knowledge; the ability to use numbers, rules, frames and object oriented programming approaches to express knowledge of varying types of design is much needed. Thus, the ideal engineering programming environment

would support all of these representations in an integrated and convenient package. A knowledge-based system using only a general purpose programming language could not be easily modified or understood, and furthermore, all the capabilities have to be created by the programmer. A general purpose representation language was the most appropriate for BBCAD (GWII [GoldWorks 87]) coupled with a few of the general programming languages (LISP, C, FORTRAN, and script files).

2.5 Electromagnetic Device Design

In designing a device one begins with a request to perform a certain function based on numerous constraints such as basic structural (e.g., physical components) and dimensional parameters, i.e. mathematical models, thermal demands, and magnetic and electric requirements. Consider the problem of designing a C-core magnet, shown in Figure 2.1, where the mean core width is 1 centimetre, the mean core length is 26 centimetres, the air gap length is 1 centimetre, and a total of 400 amperes is injected in the copper coils. The initial design stage starts from the above specification for the device, and proceeds to the next levels of the design where constraints in a particular domain such as physical dimensions, magnetic requirements, and structural demands, may be imposed narrowing the search space to reach a final design. This is normally achieved through an iteration procedure of refinement and analysis until at least the specification criteria for the device are met.

In a hierarchical design system [Lowther and Saldanha 86], devices are represented in a modular fashion as a combination of sub-parts. This is shown in Figure 2.11 which depicts the structure of a simple C-core device. Also refer to Figure 1.2 of Chapter 1 in [Brett 90], and Figure 2.1 of Chapter 2 in [Zhu 91] where the partial structure of a stepping motor and the structure of DC machines are represented respectively. These different structures illustrate the several levels of abstraction of physical devices.

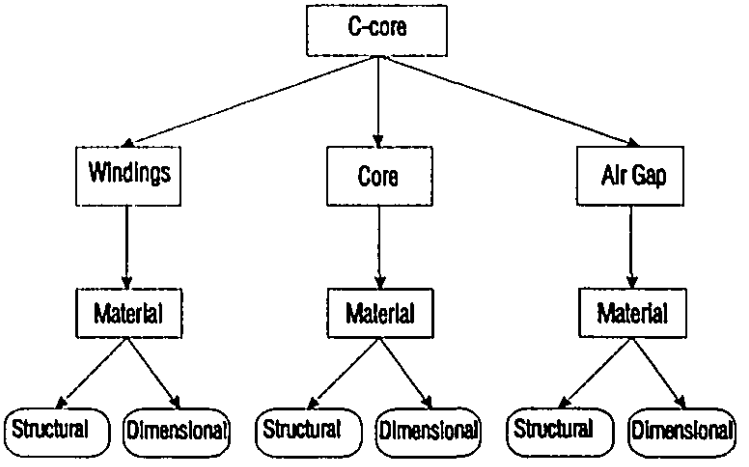

Figure 2.11: C-core structure emphasizing hierarchy

The structural model of the device consists of two or more independent sub-parts, each sub-part has an electric and a magnetic circuit. The structure is drawn out as a kind of tree, each independent sub-part is decomposed into constituent sub-parts and this tree structure grows downward until the fundamental elements of the tree are reached. These derived specifications form the basis of the structural knowledge about the device. The function of each sub-part of the structure may be represented by parameters that are usually related through a numerical model to other quantities such as the operational and performance variables.

The creation of an initial design continues by determining values for the device parameters using an improved model at each level down the tree hierarchy, and by modifying constraints on parameters when backtracking through the tree. Thus operating within the range detailed in the specifications provides bounds on the search space. Fine tuning the search space may lead to a faster convergence to the device design.

2.6 BBCAD Framework and Knowledge Architecture

The blackboard architecture represents an approach to the problem of engineering design in which an assembly of design tools are permitted to respond opportunistically, i.e. interrupt driven, as the design develops. The BBCAD blackboard framework is a problem-solving architecture based on the cooperation of several logically independent design knowledge modules, called *Knowledge Sources* (KSs), which are accessing the same information on a global knowledge storage system, the blackboard. It specifies how knowledge must be organized into independent tasks which read and write design objects on the blackboard. The KSs have a condition/action construct, i.e. they include the knowledge to be applied and the knowledge about when it should be applied. The condition part looks for a particular object configuration on the blackboard and when such a configuration is recognized, it creates a *Knowledge Source Activation Record* (KSAR). KSARs are instantiations of the KSs with contextual information. A control mechanism is responsible for the characterization, comparison and selection between the KSARs. When interrupts occur, a KSAR is triggered and modifications prescribed by the action part are made to the blackboard.

As already has been shown in Figure 2.5, there are three different kinds of knowledge category in the BBCAD environment: knowledge storage, design knowledge architecture, and control architecture. They offer different facilities in the design process, and each of these will be looked at in more detail below.

2.6.1 Knowledge Storage

Knowledge storage refers to the knowledge space, i.e. the actual facts describing the current design, of the design problem and how it is translated to build the design space. As depicted in Figure 2.12 which is a more detailed view of Figure 2.5, knowledge

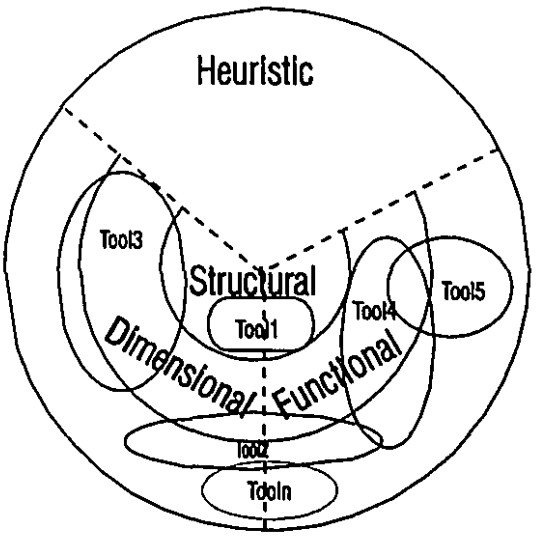

Figure 2.12: Design tools projected on the knowledge space

in the design process is divided into three basic classes:

- Deep or structural.
- Mathematical model or dimensional.
- Shallow, surface, or heuristics.

Deep knowledge involves the key components which make up the device. It consists of well-known facts, or refers to reasoning from basic principles; that is, from basic laws of structural and behavioral models, such as Maxwell's equations and Newton's laws. It is founded on frame-based systems [Minsky 75] where the intention is to incorporate structural knowledge of a design problem into the knowledge space, i.e. by definition deep knowledge is structured knowledge and can be represented in a well defined data structure.

The mathematical model is used to express knowledge relating the *dimensional*, performance and functional parameters of a device to guarantee its operation [Saldanha and Lowther 86; Lowther 89].

The third term is the most abstract and is used to provide expertise. The term *Shallow knowledge* is that heuristic, experiential knowledge that comes from having successfully solved a large number of similar problems. It is usually a rule-based [Buchanan and Shortliffe 84; Hayes-Roth 85(b)] inferential process, in which the reasoning requires a set of rules acquired from an expert. These rules are based on experience, as is the case in diagnosis systems [Talukdar and Cardazo 88].

Projecting design tools onto the knowledge space (Figure 2.12), will lead to new solutions, freeing off constraints, thus narrowing and fine tuning the search space. Thus, knowledge provides the means by which designs progress from known facts to new facts. It also embodies general descriptions of designs. In order to process knowledge, there must first exist techniques to represent this knowledge and methods to control its processing. The exact form and content of structure depends on the particular design domain and its implementation. The requirements imposed on these structures are that they support effective modes of communication, are adequate to represent the information pertinent to the design domain, and allow for effective organization of the information contained in the blackboard.

The BBCAD blackboard contains two types of knowledge: *static* and *dynamic*. Static knowledge is typically the domain-specific knowledge that is relevant to the design problem and that will have a relatively long life during the design process. It generally consists of factual data relating to initial conditions, parameters, values, relationships, etc. Dynamic knowledge is typically the knowledge that is generated during the execution of a design module. It will consist of short-term communications such as goals to be pursued, requests for data, and suggestions. The dynamic data will frequently be modified or deleted after a short period of time.

In [Gero and Coyne 85; Coyne et al. 90], two other types of knowledge are proposed. The first is *semantic knowledge* which is concerned with the meanings of

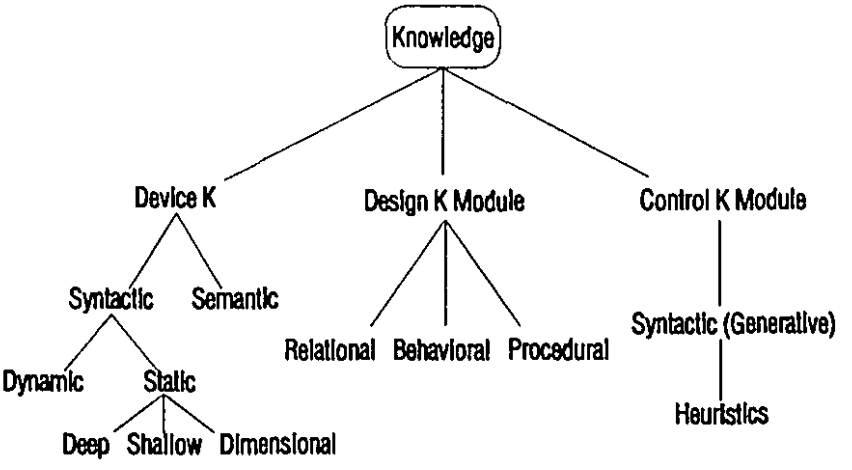

Figure 2.13: Tree structure of BBCAD knowledge

objects, and normally deals with relationships between objects and among their attributes. The second type of knowledge is *syntactical knowledge*. It deals with the connection of an object with its domain or with other objects, and with data which supports facts. This object approach can be more formally extended in object-oriented systems [Stefic and Bobrow 86]. Such systems organize information so that objects have associated with them knowledge about how they behave. Knowledge in engineering design was viewed as the tool whereby the designer conceptualises the semantic content of a certain domain, and by which he or she represents his or her ideas about that domain as the syntactical relations between the variables (facts) and the actions by which these variables and relationships are manipulated (control knowledge). A tree structure of BBCAD knowledge is shown in Figure 2.13.

2.6.2 Design Knowledge Architecture and KSs

Before knowledge can be represented, the type of knowledge involved must first be identified and classified. The following discusses two types of BBCAD knowledge:

- "Device Knowledge" represents the physical structure of a device, i.e. the full set of specifications and parameters which describe the operation of a device.
- "Design Knowledge Modules" are objects which combine relational, behavioral, and procedural knowledge of design tools. *Relational knowledge* represents design rules, programs, and requirements to run the programs. *Behavioral knowledge* contains information, i.e. data placed in the context of design needs, on how the parameters of different tools affect each other, and how an input parameter of one design tool is related to an output parameter of a previous tool. *Procedural knowledge* represents the algorithms that are concerned with the use of that knowledge in the design process.

The central data structure is the blackboard, through which the various design modules communicate. The declarative knowledge is structured through the use of hierarchies of frames. The representation has a hierarchy of abstraction levels which contains different degrees of detail [Preiss 80]. The facilities of GoldWorks [GoldWorks 87] are used to define relationships and inheritance semantics between the design tools of the device. The most commonly used relations are "IS" and "INSTANCE". The "IS" relation defines hierarchies of classes where each higher level subsumes the lower level classes. The "INSTANCE" relation declares that a particular object belongs to a class and the description of the class serves as a prototype of the instances.

The design knowledge modules, i.e. KSs, contain information about when they are applicable, they usually have some mechanism to maintain local context, and have specific knowledge which, when invoked, changes information maintained in the blackboard. The KSs are independent, event-driven processes which manipulate the global knowledge storage.

In BBCAD, each KS represents some particular specialized existing tool pertaining to the design problem being solved. The KSs are also implemented in a frame-based fashion as an assembly of rules, functions and facts, thus representing an action, i.e. a

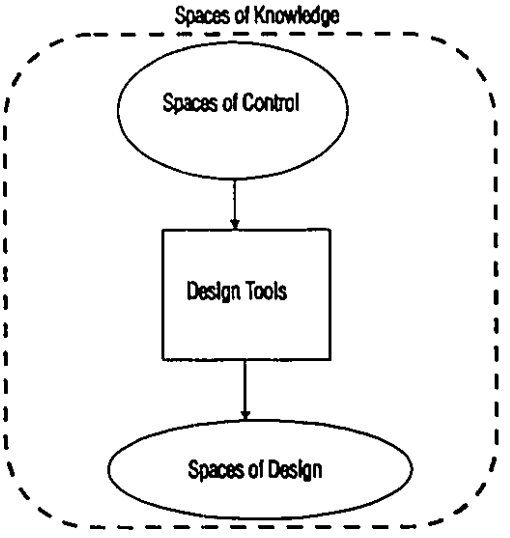

Figure 2.14: Control knowledge spaces

change to the blackboard, under appropriate conditions. Only the frames placed on the blackboard affect other KSs; the internal workings of a KS are invisible. In BBCAD each knowledge module has the following two constituents: The condition and the action. The *condition* part determines when the design knowledge module can be satisfied. The *action* part specifies the actions, i.e. rule type structure, that are to be taken based on the global knowledge base. They are similar to production rule-based systems [Hayes-Roth 85(b)], in that they consist of two parts, an antecedent and a consequent. Firing or executing the rules is determined by the matching of patterns among entries in the design space. The executable part of the rule can involve the modification of the design space and can contain a collection of logical relationships, or functions programmed in a conventional procedural programming language (e.g., C or Fortran). Thus KSs are a natural unit of representation for domains requiring hybrid knowledge representation and where much knowledge is procedural.

2.5.3 Control Knowledge Architecture

The spaces of control contain a set of general control knowledge modules, i.e. control strategy, that operate on the design modules (Figure 2.14), and these control modules are domain-independent. This space of knowledge uses the various control modules to reason about the design process, and the notion of separate *domain* and *control* KSs has been preserved [Hayes-Roth 85(a)]. This knowledge is concerned with the generation of appropriate action sequences (Figure 2.13), and leads to scheduling the various design modules. The generative knowledge serves to define a space of designs. These control knowledge modules deal with important issues, and they are:

- How to focus the search: this idea can be exploited explicitly by partitioning the design space in certain ways and by using heuristics to decide the part of the design space to which a tool should be applied to produce a new partial solution to the device design.
- How to deal with sub-design problems.
- How to integrate and coordinate multiple design tools.
- Goal satisfaction and failure handling in the device design.

The design process needs to be guided or controlled in its search of the design space for a solution to the current device problem. Thus, a major component of a blackboard architecture is the control mechanism. This mechanism determines which design knowledge module can be activated in the event that several have been satisfied. In BBCAD, the process is as follows:

1. Specify the design problem.
2. Post the goal on the blackboard: using the finite-element analysis tool [MagNet 85], determine the flux density in the air gap of the C-core. The initial goal in the design of the C-core is to run the drafting pre-processor, Draw2d, where the basic geometry is entered from an initial design specification file.

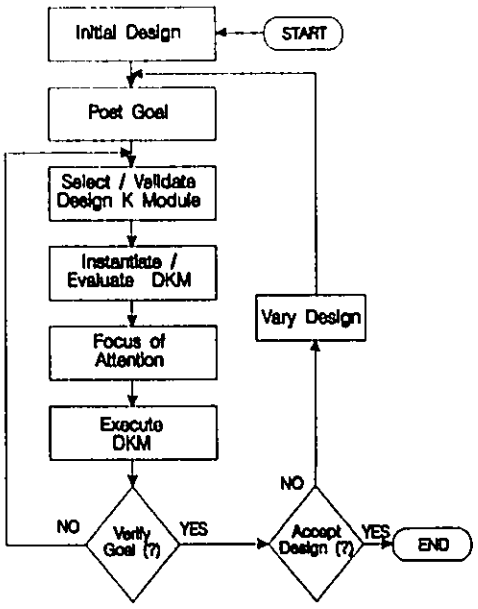

Figure 2.15: Control flow chart

3. Select and validate the design knowledge module: this involves matching the condition part of each design knowledge module to the global knowledge base.
4. Instantiate: a knowledge source activation record (KSAR) is created for each satisfied design knowledge module.
5. Evaluate events: this involves the use of heuristic searches to assign priorities to events.
6. Focus Attention: evaluates the estimates received, and the best knowledge source activation record is selected for firing, i.e. the design module with the highest priority.
7. Execute the event with the highest priority: the drafting module, Draw2d, is fired to display the geometry of the design, and prepares the C-core for the next step which is the mesh generation.
8. Verify the goal of the design: the next goal is to specify the finite element mesh for the C-core problem. At each step of the design process different goal is posted and the appropriate module is executed. The process continues until the goal of the specific design tool is reached, and the desired performance characteristics of the device are

obtained.

This is illustrated in the flow chart of Figure 2.15. Essentially, this process continues until at least the requirements for the design are met, giving an initial design. The design process is regarded as a search through a design space for a solution that closely satisfies the specifications of the device. The function of control in a blackboard system is to provide for focus of attention and optimal use of the available resources.

2.6.3.1 The Scheduler

The mechanism of control may be implemented by using an agenda-based scheduler. At each level of the design process, a different representation of the device may be necessary. The agenda lists all pending knowledge source activities, and the scheduler decides which of these activities to execute first. One basis for making the choice is the amount of new information which might be provided by a particular tool. The scheduler can calculate a priority for each waiting task and select for execution, for example, the task with the highest priority, i.e. a simple best-first strategy.

The scheduler has to decide which of the currently applicable design tools is most likely to make the largest contribution towards the final design. There are a number of different scheduling techniques which can be implemented in the control knowledge modules. Some of these are:

- Place design modules that produce the largest amount of new design information higher up in the schedule.
- Place design modules affecting the least recently derived information higher up in the schedule. This would tend to produce a breadth-first search.
- Place design modules that are implemented by the most recently fired action at the end of the schedule.

- Place the design module with the highest priority value at the end or at the top of the agenda list.

At every cycle of the design process, several design modules may compete to fire their action rules, the scheduler determines which module to execute and in what order.

2.6.4 Conclusions

The use of the blackboard as a single mechanism for data sharing and communication, together with the gathering of knowledge and processes into independent knowledge sources, provides a neat, conceptually clean framework for developing a complex integrated engineering design system. Also, the modularity of each knowledge source localizes problems and simplifies the task of changing the module, expanding its capabilities, or adding new modules to the system. Furthermore, additional executive control expansion can be accomplished by making similar changes to the control level knowledge sources and blackboard structures. BBCAD uses a frame structure to represent design modules, goals, and explanation and information messages.

2.7 Implementation

There are many languages, operating systems, and processors that can be used to program knowledge-based systems. These items are interrelated, but these relations are limited to the specific processors and operating systems used. The implementation step involves encoding the design knowledge into the chosen tool. The development of BBCAD required the selection of a programming environment (tool) for building the knowledge-based system. The tools currently available provide varying degrees of assistance in the development task. Their choice depends on the basic approach taken [Hayes-Roth et al. 83]. Appendix C reviews some of these languages.

Apart from the approaches described in Appendix C, other considerations affect the choice of implementation in engineering design: the sort of design to be solved, the desired capabilities of the KBS, and the availability of the desired tools. In engineering design, knowledge-based skeletal systems (Appendix C), have the advantage of allowing one to create a new knowledge-based system in a very short time with relatively modest effort. The main disadvantages with this approach are that many times the skeletons are unchangeable and difficult to extend to deal with engineering design tasks not originally mapped out by the tool designer. As well, the generalization step (Appendix C) is not completely successful and some domain-specific characteristics that could interfere with the new design task remain.

CHAPTER 3

Structure and Knowledge Organization

In this chapter, the knowledge organization structure of the BBCAD framework is described. It explores the kinds of knowledge that are to be used, the way they are represented, and the methods by which the knowledge should be accessed. It also discusses the dependencies between the various design modules and their parameters, the rule inferencing approach, and a locking mechanism to focus on a specific design module.

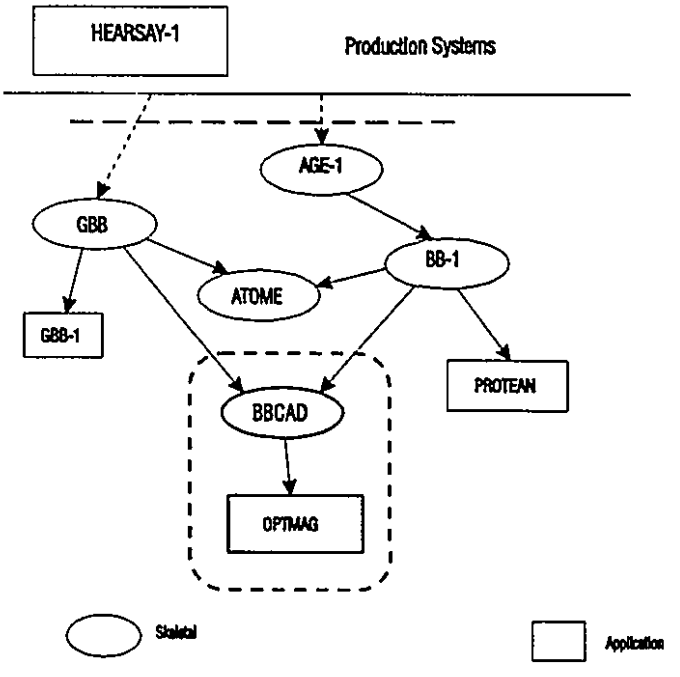

Figure 3.1: Influences on BBCAD (Extension to Nii [86(a)])

3.1 Characterization of BBCAD

For easy manipulation of the design space, BBCAD uses a combination of knowledge representation schemes and multiple levels of knowledge abstraction to help and to facilitate the decision making process in designing electrical devices.

BBCAD is a hybrid knowledge-based system environment with a blackboard architecture, rule-based systems and frame-based structures utilised together in the integration of design-dependent and design-specific knowledge for solving electromagnetic design problems, such as the C-core magnet. Figure 3.1 is an extension of the family tree of the various application and skeletal systems described by Nii in [Nii 86(a)] showing BBCAD in the general development of blackboard systems (also refer to Appendix C). BBCAD is not in any way an extension to the systems shown in Figure 3.1, it only reflects its likeness to previous systems.

3.2 System Structure

The overall system structure of BBCAD is illustrated in Figure 3.2. It consists of hierarchically organized categories of frames, a knowledge storage area, i.e. a blackboard, a control mechanism, a scheduler, and a user interface.

The design module frame, i.e. the Knowledge Source, represents the characteristics of the tools to be coupled and is structured so that the selection and the sequencing of the modules is carried out effectively and efficiently. A tool can have more than one design module, for example MagNet is a modular computer-aided design package, used to model magnetic and electric field problems, with modules enabling geometric specification, material modelling, problem specification, solution evaluation, and post processing [MagNet 85]. Associated with each design module is a set of rules generated from the design module for checking the adequacy of the requirements and the use of the module

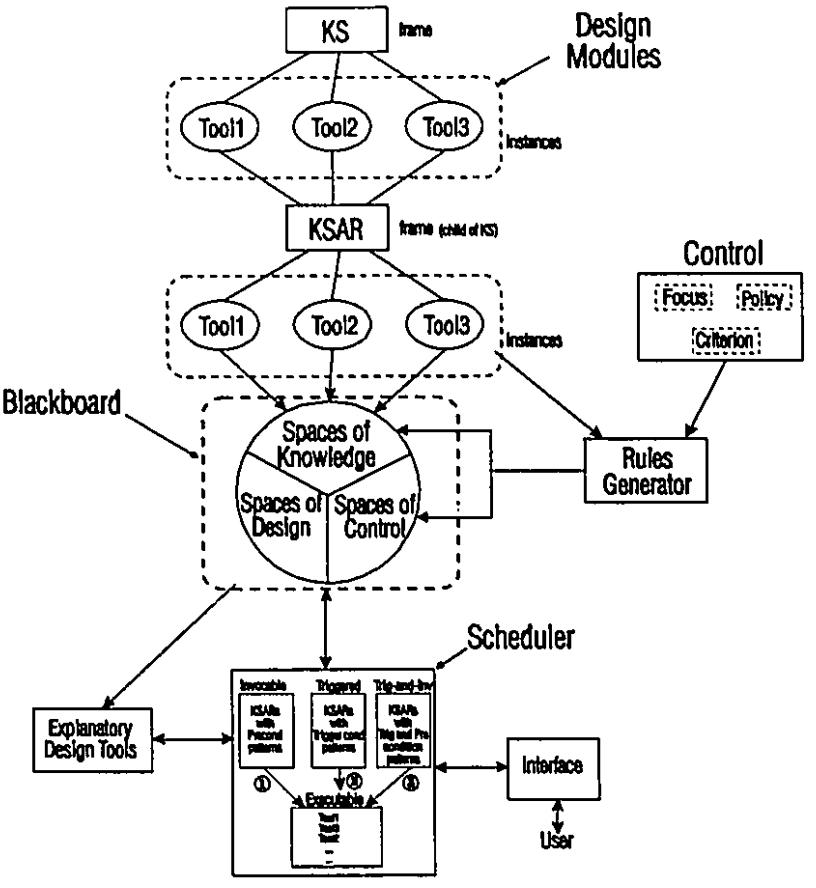

Figure 3.2: System structure of BBCAD

(the rules generator in Figure 3.2). A set of strategies and heuristic control rules are also generated from the domain control knowledge modules and used by the scheduler to determine which of the executable design modules is most desirable at a particular cycle of the design process. The scheduler places design alternatives in three queues (the Scheduler of Figure 3.2): the invocable design queue ①, the triggered queue ②, and the triggered and invocable queue ③, the order in which they are addressed.

The explanatory component is used to enable BBCAD to explain the functions and actions of the design module, and to interpret the accomplished activities achieved by the module. Although this module is not necessarily required to get BBCAD to do its task,

it is very important in terms of user acceptance and design transparency. The user interface, discussed in more detail in Appendix F, is the communication medium between the device designer and the BBCAD system. Within the interface, the user can enter design specifications, and has facilities for viewing, adding, modifying, and deleting design modules, rules, and instances. A graphic object approach is used to define images, screen layouts, popup menus, and other elements of the shell. A menu-driven user interface presents default values for all the requested variables and accepts user inputs, prompting for re-entry of values that are out of range and supplying help as requested.

3.3 Structure and Types of Knowledge

The quality of knowledge-based systems depends heavily on the robustness, integrity and amount of knowledge incorporated into the design space. For the BBCAD architecture, various forms of knowledge are used, the structure is mainly based on two types of knowledge (Figure 3.3):

1. **Design independent knowledge** represents the background knowledge of the design tasks. It encompasses heuristics and general rules of thumb about specific design tools, and contains design knowledge, procedures and algorithms. It relates to the physical laws and general design strategies common to all design problems. The design independent knowledge is divided into two classes:
   a. Domain specific knowledge: KS frames and instances. In the design of the C-core device, Magnet is used as the finite-element analysis package to determine the flux density in the air gap. DRAW2D is a module of the Magnet package which defines the basic geometry of the magnet. Thus, the DRAW2D frame, KS, should contain all the necessary information that satisfies the execution of the module, as well as other elements such as its relationships with other modules (e.g. priority factor, etc.).

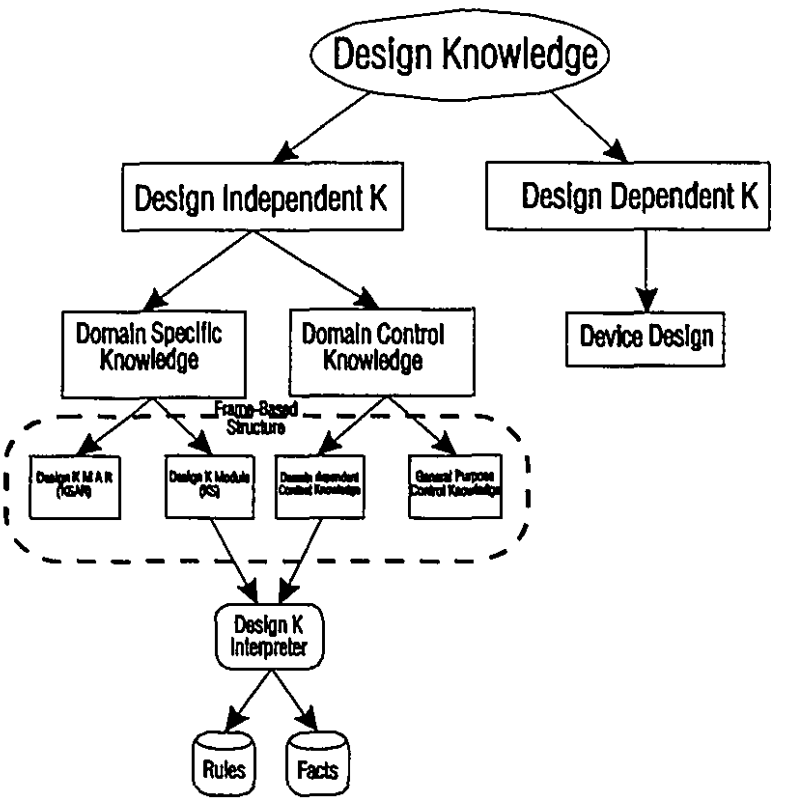

Figure 3.3: Different types of knowledge in BBCAD

b. Domain control knowledge: deals with domain dependent control knowledge (e.g. competence, criterion, etc.) and general purpose control knowledge (e.g. number of cycles, integration rules, policy rules, etc.).

2. **Design dependent knowledge** represents facts and relations of the specific device design to be solved. It refers to the full set of parameters which describe the structure and operation of a device, e.g. the C-core magnet, its material properties, and specifications. The detailed description of the representation of the magnetic device will be carried out in Chapter 5.

3.4 Knowledge Representation

BBCAD uses various knowledge representation mechanisms such as:

- Frames for the description of the modules characterizing the design application problem, and for message-passing.
- Production rules for the representation of control and heuristics.
- Procedures for solving conventional device equations, expressions involving relations between parameters from the same or different tools, and arithmetic functions.

Details of these representations are given subsequent to each type of knowledge.

3.4.1 Frame-based Representation

The BBCAD tree structure, shown in Figure 3.4, is founded on a frame-based approach. Frames are nodes of the BBCAD tree that represent general classes of design knowledge modules, where the tree is a network representation composed of frames, "parent" and "child" frames, and instances. Each frame provides a level of abstraction which reduces the size and complexity of the search space. It can also be seen that the "parent" frame of BBCAD is *'top-frame'* (Figure 3.4). The frame *'data_structure'* is the sub-frame, called the "child" frame. A child frame inherits all of the slots from the parent frame and can have specific local slots of its own that describe any unique attribute of the frame. Table 3.1 depicts the BBCAD levels of abstraction.

Frames are used as templates to organize and structure the various knowledge components in the spaces of knowledge storage (Figures 2.5 and 2.12 of Chapter 2). Since frames are templates for structuring information, slots in frames do not hold values. However, a copy of a frame in which the slots hold actual values, i.e. the factual knowledge, is said to be instantiated and represents a specific case of the object in the design process.

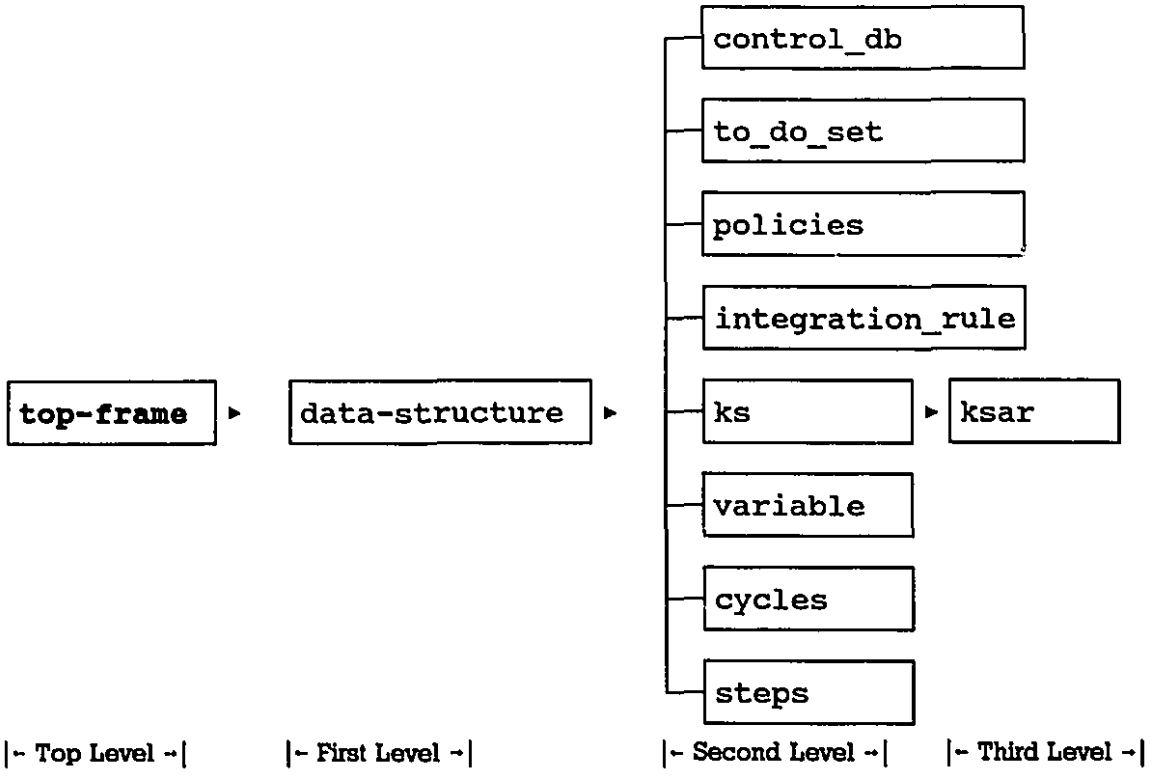

Figure 3.4: The BBCAD frame-based tree structure

Table 3.1: Basic BBCAD frames levels of abstraction

Frames	Child-frames	**Definitions**
data-structure		Top level data-structure.
ks		Design knowledge module representing the domain specific knowledge.
	ksar	Design knowledge activation record: each instance of the design knowledge module produces a unique ksar.
control_db		Control knowledge module frame leading to different levels of abstraction of the domain dependent knowledge modules.
to_do_set		Frame structure identifying all pending ksars on each cycle of the design process and distinguishing the "invocable_list", "triggered_list", and "trig_and_invoc_list" of pending ksars.
policies		General purpose control frame identifying the pending "trig_and_invoc" ksars.
integration_rule		General purpose control frame identifying the lists of weights of the design knowledge modules.
variable		General purpose frame tracking down the activities of the control mechanism.
cycles		General purpose frame identifying the design cycle.
steps		General purpose frame identifying the step within a design cycle.

```
(define-frame ks  (:print-name "ks"
                   :doc-string "Knowledge Source..."
                   :is data_structure)
 (action           :multivalued t                    ; ( (rule1 prio (IF) (THEN))          ②
                   :default-values (true) )          ;   (rule2 prio (IF) (THEN)) ...)
 (explain_action   :multivalued t                    ; ( (rule1 "st1") (rule2 "st2") ...)  ③
                   :explanation-string  "This is the rule explanation string" )
 (post_action      :multivalued t                    ; (rule1 prio (IF) (THEN)) (rule2...  ④
                   :default-values (true)
                   :explanation-string  "Specific info why KSAR can not run" )
 (action_level     :constraints (:lisp-type integer)                                       ⑤
                   :default-values (0) )
 (bb_type          :constraints (:one-of (domain control distributed)) )                   ⑥
 (efficiency       :constraints (:lisp-type float)                                         ⑦
                   :default-values (0.0) )
 (pre_cond         :multivalued t                                                          ⑧
                   :default-values (true) )
 (trigger_cond     :multivalued t                                                          ⑨
                   :default-values (true) ) )        ;end define-frame
```

Figure 3.5: Frame structure of the design knowledge module (ks)

One interesting consequence of this, of course, is that the dimensionality and structure of the search space is controlled by the frame structure. The two are closely related. Thus by defining a frame one has defined the space.

3.4.1.1 Domain Specific Knowledge

Domain specific knowledge (DSK) is that which is relevant to the tools involved in the design process of a particular device, e.g. the C-core magnet. DSK has knowledge about each design tool, the conditions under which this tool is executed, what is needed to run the computer programs, and relationships with other tools, all of which are embedded in a production rule structure. The two major frame structures of BBCAD DSK are: the design knowledge module frame *'ks'*, and the design knowledge module activation record frame *'ksar'*. These frame structures are shown respectively in Figure 3.5 and Figure 3.8. The *'ksar'* frame is a child frame of *'ks'*, and inherits all the slots of the parent frame. Each of these will be looked at in more detail below.

1. **Design Knowledge Module Frame** (Figure 3.5): the module *"draw2d"*, depicted in Figure 3.6, of the analysis tool Magnet [MagNet 85] is described as being a specific instance ①, of the design knowledge module frame *'ks'*. It is defined by the following:

→ Precondition (**pre_cond**, ⑧) and trigger condition (**trigger_cond**, ⑨): these are used to monitor the BBCAD blackboard for elements matching the desired precondition and trigger condition. They have the double purpose of instantiating a knowledge source activation record (KSAR) of the design module that is appropriate for an action and of invoking the module at the appropriate time. For each instantiated design module a KSAR, is created and inherits the properties of the KS frame in addition to its own.

The above terms do not correspond to the traditional blackboard definitions of BB1 [Hayes-Roth 85(a)]. Generally, triggers are used in the conditions of KSs to specify which state changes, or events, are of interest to the KS (i.e., the trigger declares when the knowledge in a KS becomes solely relevant in light of what should happen in the course of design). In order to capture a useful KS condition, the triggering mechanism is augmented with an extra check on the blackboard state. This extra check is called the precondition (that part of the condition of a KS that determines the applicability of the blackboard by using its state information).

To facilitate the integration and the control of the tools, BBCAD places the design modules in three different queues depending on the values of the **pre_cond** and **tigger_cond** slots (Figure 3.2). Design modules with precondition terms to satisfy are placed on the first queue, the second queue contains modules that have trigger condition terms to satisfy. When the module contains precondition and trigger condition terms to satisfy, the module is placed on the last queue. This strategy easily implements the phases of the analysis tool of electromagnetic devices by the finite element technique [Lowther and Silvester 86]: preprocessing, solving, and postprocessing (Figure 2.9 of Chapter 2). For example, the preprocessing tool contains

```
(define-instance draw2d   (:print-name "draw2d"                                    ①
                           :doc-string
                            "Draw2D is the first MagNet module, and is designed to allow
                             the user to GENERATE the outline of problem GEOMETRY..."
                           :is ks)
                ;(r_name  r_priority (if-part) (then-part))
 (action        (rule1    600      ;antecedent                                    ②a
                                   ((instance draw2d is ks)
                                    (instance ?ksar_draw2d is ksar
                                              with ks_name draw2d) )
                                   ;consequent                                    ②b
                                   ((setf (slot-value 'out_window 'display)
                                    (format nil "~&Running Draw2d") )
                                    (format "file_out" "~&Running Draw2D !")
                                    (run_draw2d)
                                    (instance ?ksar_draw2d is ksar
                                              with ks_name draw2d
                                              with state fired) ) ) )    ;rule1
 (explain_action           (rule1   "Draw2D is the drafting pre-processor,          ③
                                     where the basic geometry is entered.") )
 (post_action              ((true))                                                 ④
 (action_level             10)                                                      ⑤
 (bb_type                  domain)                                                  ⑥
 (efficiency               .9)                                                      ⑦
 (pre_cond                 (instance draw2d is ks)  )                               ⑧
 (trigger_cond             (true) )  )                                              ⑨
```

Figure 3.6: "draw2d" design knowledge module

$$\textit{((rule-1) (rule-2) ... (rule-n))} \qquad \text{(a)}$$

$$\textit{rule-n} = \textit{((rule-n}_{\textit{name}}\textit{ priority (antecedent-n) (consequent-n))} \qquad \text{(b)}$$

Figure 3.7: Rules expression form

precondition terms, the solving tool contains trigger condition terms, and the postprocessing contains both precondition and trigger condition terms.

→ Knowledge specific module (**action**, ②a,b): this slot value contains rule patterns specific to the domain knowledge of the design module and the execution of these rules depends solely on the slot values of the precondition and trigger conditions. The slot value is a list of similar rule structures, as depicted in Figure 3.7 (a), and these rule

patterns are defined by the user when creating the design tool. Each rule structure is of the form of Figure 3.7 (b) where the second term of the list is an integer indicating the priority of this rule relative to other rules in the list. To add more flexibility to the search space, BBCAD has extended the capability of this strategy by assigning different values to the rule's priority when defining rules (e.g., rule1 in Figure 3.6 ② has a value of 600), i.e., different rule constructs can have different priority values. The rule priority can change during a design process depending on the situation. The second term of the antecedent part (②a of Figure 3.6), includes an identifier, i.e. ?ksar_draw2d. Identifiers that begin with a "?" are pattern matching variables. The variable will be *bound* to the value of the item it matches against. The pattern matcher tries to find a set of values and objects, which, when bound to the variable, match the instance that has "draw2d" as slot value in **ks_name**. When the rule fires, the binding made in the antecedent holds in the consequent, Figure 3.6 ②b, so that the value "fired" is asserted in the slot **state** of the ksar instance. The firing of the rules is subject to the result of the FIFO strategy which will be discussed later in this chapter.

→ Explanation module (**explain_action**, ③): this contains an explanation of the rules in the **action** slot. This is to inform the user of the activity occurring at each step of the design.

→ Meta-knowledge specific module (**post_action**, ④): this slot is concerned simply with the relationship between clauses in the action slot, i.e. rules, and it holds a list of rule patterns of the form: *(rule-name (if-part) (then-part))*.

These rules are similar to the AND-THEN clause in "IF (a) THEN (b) AND-THEN (c)". This is used to add a clause to the consequent that will be asserted when the antecedent of the action rule is satisfied and the consequent is asserted. This is demonstrated when using the solver in Magnet: IF the problem is magneto-static, non-linear, in a Cartesian geometry THEN conclude that XYPM is the solver to be used

here to produce a magnetic vector potential solution AND-THEN execute the XYPM solver. In the next cycle around, there is no need to check the above rule if no changes has been made to the problem specifications, thus converging faster to a solution state by narrowing down the search space for the rule to fire.

This slot can also be used as a run-time explanation of what is happening when a design module fires and detects misleading results in the action rules, in which case information and an explanation are posted, and user intervention is imminent. For example, in designing a C-core magnet a solver is needed to produce a magnetic vector potential solution. The exact combination of the coordinate system and material properties determines the solver to be used. If the requirements are satisfied but the solver program does not exist, the user is asked to intervene and make a decision on the solver type to be used.

→ Knowledge domain level (**action_level**, ⑤): the value of this slot is an integer number. It is used by the meta-scheduler to execute the design module with the highest value, in the case of conflict. If two or more design tools have an equal value, the decision to invoke a design module is subject to the FIFO strategy. This term is used only if the priority value, determined by the efficiency component ⑦, of two or more design tools is the same.

→ Knowledge type (**bb_type**, ⑥): this slot is used to indicate whether the design module is "domain", i.e. the tool is located on the same machine, or "distributed", i.e. the tool is on different machine. This feature is used to expand the BBCAD to include a distributed environment.

→ Knowledge efficiency (**efficiency**, ⑦): a floating point number which is used to calculate the priority rating of a design module (refer to Appendix D). This value plays a salient role in scheduling the design tools. In the case of a conflict, the action_level ⑤ will assist in scheduling the tools.

```
(define-frame ksar          (:print-name "ksar"
                             :doc-string "Knowledge Source Activation Record"
                             :is ks)          ; ksar inherits all the slots of ks frame
  (ks_name          :multivalued t)
  (ksar_bb_level    :constraints (:lisp-type integer)
                    :default-values (0) )
  (cycle            :constraints (:lisp-type integer)
                    :default-values (0) )
  (cycle_update     :multivalued t
                    :default-values ((0 none value))     ; cycle# slot value
                    :when-modified (cycle_update_when_modified) )
  (priority         :constraints (:lisp-type number)
                    :default-values (0)
                    :when-modified (priority_when_modified) )
  (ratings          :multivalued t)                    ;list of control_db (focus)
  (status           :multivalued t)
  (state            :constraints (:one-of (new old fired rerun none))
                    :default-values (none)
                    :when-modified (state_when_modified) )
  (stop_criteria    :constraints (:one-of (yes no)) )
                    :default-values (no) ) )
```

Figure 3.8: ksar frame structure - child-frame of ks

Table 3.2: Basic slots of the ksar frame structure

Slot	Definition
ks_name	Name of the parent design knowledge module.
ksar_bb_level	Level of BBCAD at which the design tool is being processed.
cycle	Cycle number at which the design module was invoked.
cycle_update	Keep track of the activities at each cycle.
priority	Priority of design tool (calculated in Appendix D).
ratings	List of the control knowledge modules to calculate the priority (Appendix D).
status	Status of the design tool, i.e. invocable, triggered, or triggered_and_invocable.
state	State of the tool: new, old, fired, or rerun.
stop_criteria	Stopping criterion for the design process.

2. Knowledge Source Activation Record Frame (Figure 3.8): the *'ksar'* frame represents a unique triggering of a particular design knowledge module by a particular event and is created at runtime. Each ksar is object-directed in that it monitors the blackboard for data matching its precondition and trigger condition. It is a "child" of the frame *'ks'*, thus inheriting all the slots of the parent frame and has additional slots shown in Figure 3.8.

Table 3.2 gives an overview of the slot definitions. When a ksar is chosen by the scheduling mechanism (to be discussed in Chapter 4), its tool's action executes in the context of its precondition and trigger condition, typically producing new information in the knowledge space. When several design tools are competing for execution, the one which can contribute the most information is fired first. The slots **cycle** and **cycle_update** are used to backtrack and resolve deadlocks among competing tools.

3.4.1.2 Domain control knowledge

This section deals with control domain dependent knowledge modules and general purpose control knowledge modules (Figure 3.3). Details of these modules will be looked at in more detail below.

1. **Control Knowledge Modules:** they primarily operate on the design knowledge modules (KSs), and interpret and modify the design process activity and behaviour. The control module is represented as a frame data structure, *'control_db'* shown in Figure 3.9, where all the instances of the frame are available for interpretation and modification (Figure 3.10). All these instances respond to, generate, and modify solution elements in the design space of BBCAD, under the control of a scheduling mechanism. BBCAD uses two classes of control decision heuristics (levels of abstraction) to integrate and to schedule design tools in designing a device: the *focus* and *policy* levels. These two decision levels describe desirable actions, thereby determining which of the BBCAD's control heuristics operate on a particular design knowledge module of the design process. Design solutions at the control level, i.e. focus and policy, are decisions about what actions are desirable, feasible, and actually performed at each step in the design process. Each control decision is represented as an instance of the *'control_db'* frame where slot definitions and instances are shown respectively in Table 3.3 and Figure 3.10.

```
(define-frame control_db   (:print-name "control_db"
                            :doc- tring "Control Focus / Policy..."
                            :is data_structure)
  (competence     :constraints (:one-of (domain_ks_focus planned_ks_focus
                                control_ks_focus class_1 class_2 class_3 class_4)) )
  (criterion      :multivalued t )
  (decision_type  :constraints (:one-of (focus policy)) )
  (goal           :multivalued t )
  (first_cycle    :constraints (:lisp-type integer)
                  :default-values (0) )
  (last_cycle     :constraints (:lisp-type integer)
                  :default-values (0) )
  (status         :constraints (:one-of (active inactive)) )
  (weight         :constraints (:lisp-type integer)
                  :default-values (0))  )              ;end define-frame
```

Figure 3.9: Control decision type frame (control_db)

Table 3.3: Basic slots of the control_db frame

Slot	Definition
Competence	Competence level of design module
Criterion	Expiration condition (pattern)
Decision_type	Level of abstraction (e.g., focus, policy)
Goal	Actions to be taken (if-then form)
Status	Status in design process (e.g., active, inactive)
Weight	Design module importance
First_cycle	First executable cycle number
Last_cycle	Last executable cycle number

Focus decisions establish local design objectives to execute the ksar of a particular design knowledge module. They operate independently of one another, and are used to rate the design modules. As a consequence, they influence the scheduling decisions. At the highest level, the default ordering of the agenda items, i.e. design modules, is a breadth-first (FIFO) strategy only if none of the design knowledge modules has been fired. If two or more modules are competing, the execution is altered if one of them has already been fired and no new information is added to the spaces of knowledge. A typical instance of a focus decision control type is shown in Figure D.1 of Appendix D.

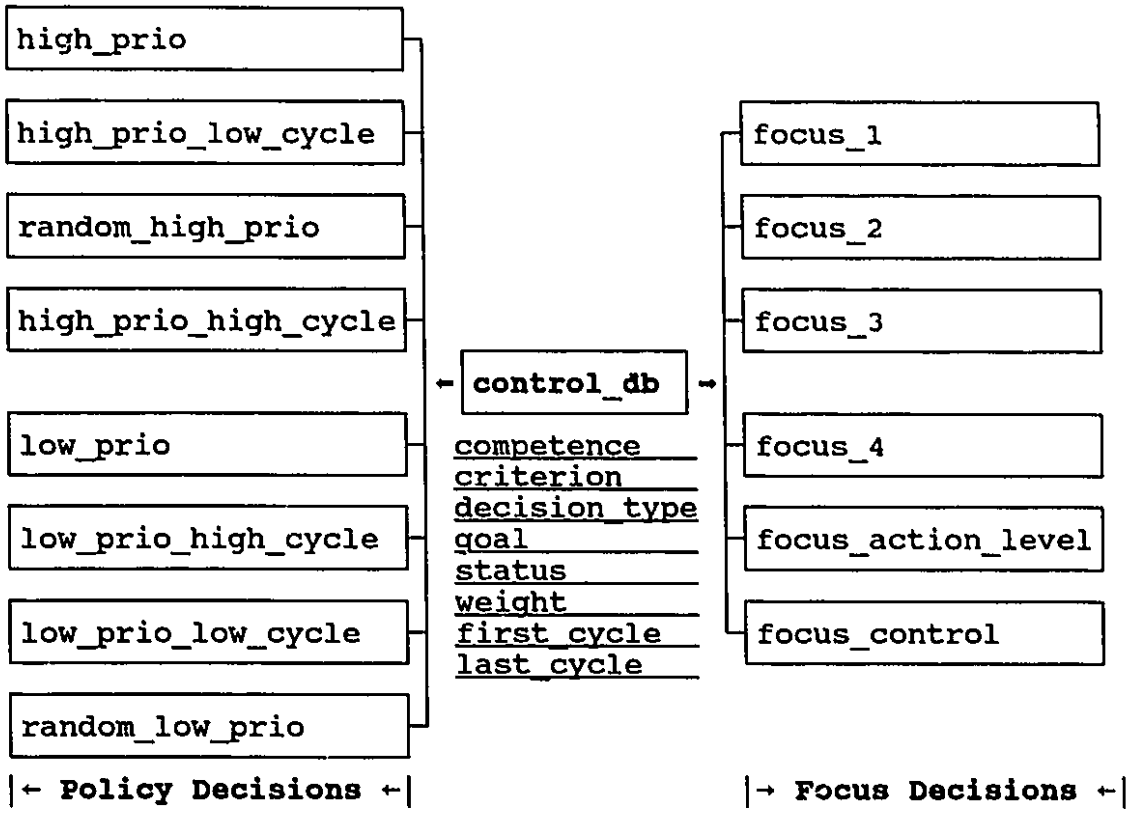

Figure 3.10: The control_db slots and instances

Policy decisions also influence the scheduling mechanism, are used to evaluate the design modules and to converge to a new solution state. They initiate global scheduling criteria favouring ksars with particular attributes and values, and carry no internal criterion values. In contrast to focus decisions, policy decisions actually influence scheduling decisions only if the executable agenda list contains design modules with the attributes and values the policy decisions describe, and these policies ordinarily remain operative for the whole design process. These type of decisions are implemented to resolve conflict between the satisfied design modules, i.e., modules that are ready to be selected and executed. The policy decisions of Figure 3.10 give the process further control over firing design modules, thus influencing the decision of which ksar to fire first. Sorting the

```
(define-instance  high_prio        (:print-name "high_prio"
                                    :doc-string "High_prio policy decision type"
                                    :is control_db)
 (decision_type   policy)
 (goal            (bind ?pairs (sort_priority_trig_invoc '> 'priority) )
                  (bind ?ksar1 (cadr ?pairs))
                  (bind ?prio1 (car ?pairs))
                  (variable-bound-p ?prio1)
                  (instance ?ksar2 is ksar
                            with status trig_and_invoc
                            with priority ?prio2)
                  (equal-instances ?ksar1 ?ksar2)
                  (= ?prio1 ?prio2) )
 (status          active)
 (weight          10) )            ; end define-instance
```

Figure 3.11: Example of control decision high_prio instance

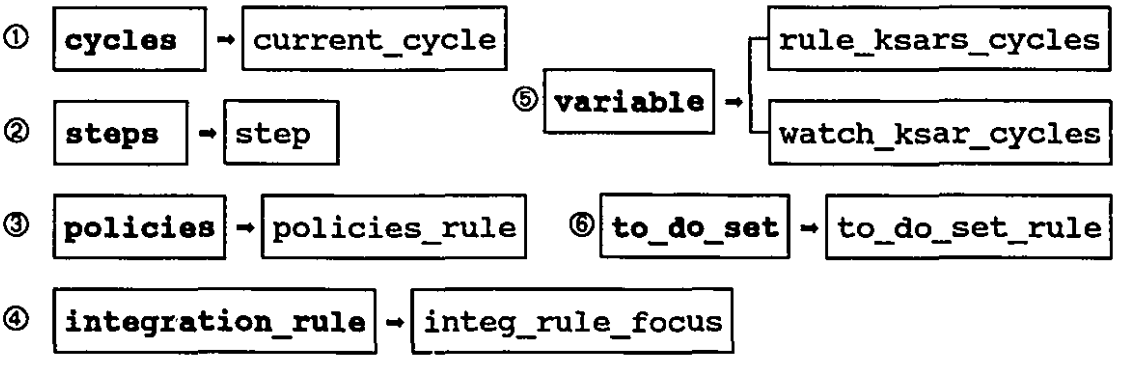

Figure 3.12: Frames and instances of general purpose control knowledge

modules in decreasing order of the priority value, calculated in the focus decision, is the default policy. A specific instance of this policy is shown in Figure 3.11, where the goal is to find the ksar with the highest priority.

2. General Purpose Control Knowledge (Figure 3.12): this type of knowledge module is used in the general control mechanism to keep track of various activities of the design process such as identifying KSARs at each cycle, cycles performed, steps in each cycle,

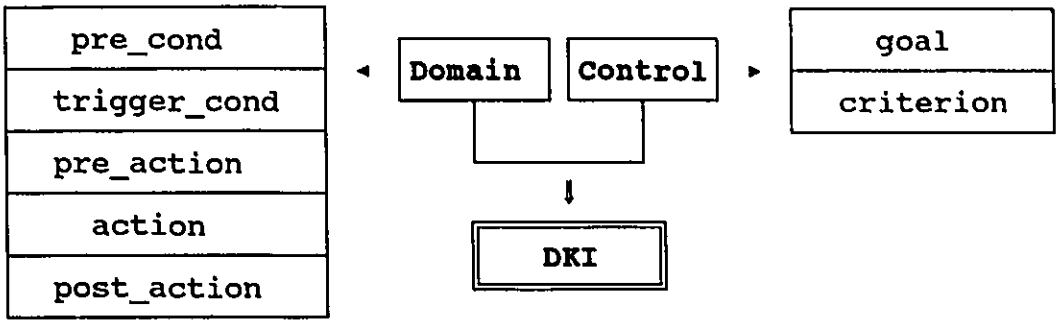

Figure 3.13: Slots in BBCAD frames that provide the production rules of the modules

```
( rule-name ({option}*)  {pattern}*                                    (a)
                  then          {pattern}*
                  [and-then     {pattern}*]  )

RULE:  DRAW2D_RULE1                                                    (b)
                  Doc String:        "action rules"
                  Explanation String: "Draw2D is the drafting pre-processor,
                                       where the basic geometry is entered."
                  Priority:          600          ;==> highest FIFO      ①
                  direction:         :FORWARD
                  dependency:        nil
                  Sponsor: ACTION_RULES           ;==> FIFO              ②
IF      {patterns}*
THEN    {patterns}*
```

Figure 3.14: (a) BBCAD rule structure, (b) Option definitions (draw2d)

weight factors connecting the control modules to the design modules, and other internal control manipulations such as the integration rule frame of Appendix D, Figure D.2. These control knowledge are also used to keep the user informed of what is going on by providing run-time explanation and tracing of what is happening, what design modules, i.e. processes, are being used, and what knowledge has been applied to make indicated decisions, thus making the design process transparent and easy to trace and comprehend.

3.4.2 Rule-based Representation

The rules are embedded in the design knowledge module formulation. The role of the design knowledge interpreter (DKI) (Figure 3.3) is to interpret this formulation and generate explicit rules corresponding to the design modules. Not all the knowledge present in the design knowledge modules can be interpreted by the DKI. Only a few slots of the design module frame, *'ks'*, and the control module frame, *'control_db'*, are translated (Figure 3.13). In BBCAD, the production rules represent cause and effect relationships using the "if-then" form, so that *if* one combination of conditions is true, *then* other patterns become true as a consequence, i.e., "IF *antecedent* patterns, THEN *consequent* patterns" (Figure 3.14(a)).

Rule antecedents may call functions to bind and assign local variables, compare slot values or local variables, and check for facts in the spaces of design. Consequents can call functions and foreign programs, assign variables, assert or delete facts, modify slot values, and change the BBCAD control states.

3.4.2.1 Rule Inferencing

BBCAD uses forward chaining to infer all possible information from the spaces of knowledge that already exist in the design space. Figure 3.14(b) depicts a rule structure of the design knowledge module "draw2d". A more detailed construct of this rule is shown in Figure E.1 of Appendix E, where the rule is presented as an example of a production rule and it is the generated outcome of the slot **action** portrayed in Figure 3.6 and referred to as rule1. The forward chaining strategy is initiated when the antecedent, or "if" portion of a forward rule matches a set of objects in the design space. When the rule is matched and ready to fire, the inference engine creates an *agenda item* from the rule and its matching objects. An agenda item is a representation indicating that some event will

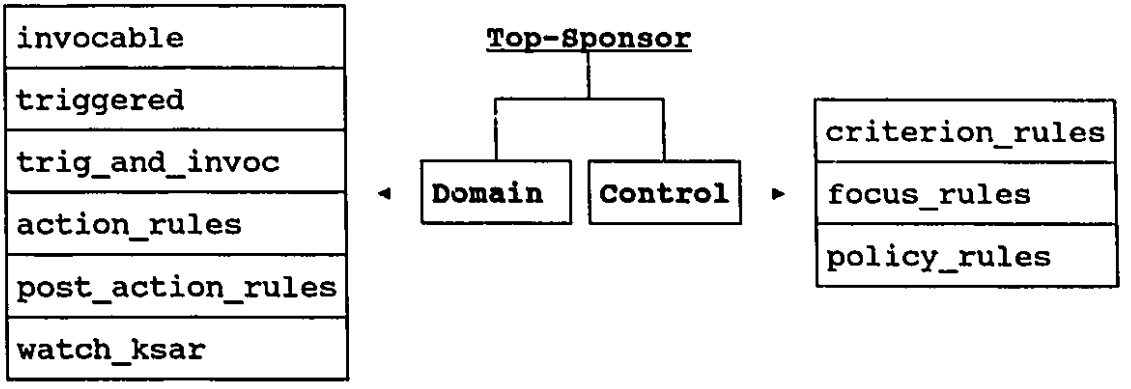

Figure 3.15: BBCAD Sponsors

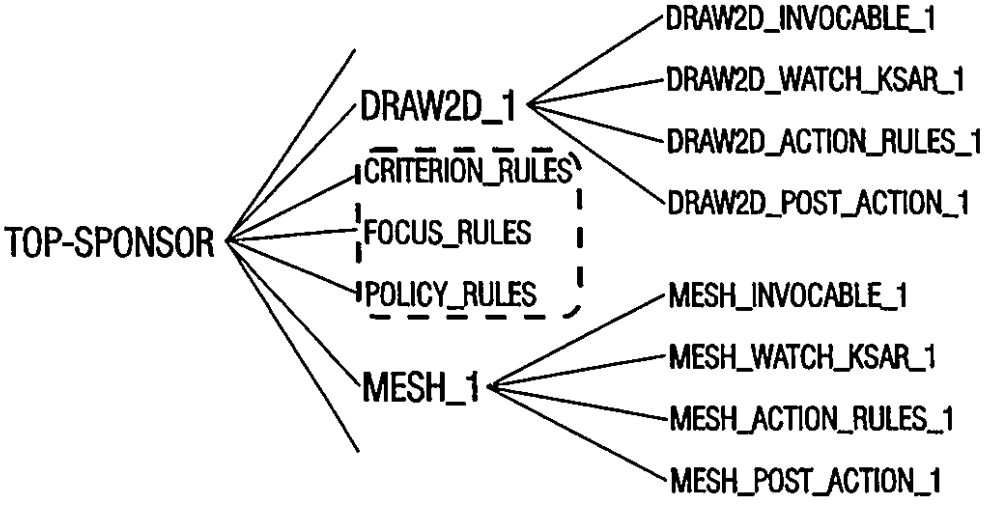

Figure 3.16: Levels of Abstraction using MagNet modules

happen if the design application is run. When the agenda item fires, the consequent of the rule enters new assertions and values into the design space. These assertions, in turn, cause more agenda items to be created, which eventually should lead to more information about, for example, the C-core design problem. BBCAD fires the agenda items, i.e. KSARs, according to the ordering of the agenda. The priority of an agenda item is determined by the priority of its rule (Figure 3.14(b) ①). This feature is user defined (the

action slot in Figure 3.6), the default is a first-in first-out (FIFO) ordering, i.e., breadth-first strategy.

3.4.2.2 Breadth-First Search

In BBCAD, at the design module level, the default forward agendas are ordered in a breadth-first manner, i.e., a prioritized queue. In this ordering, when ksars are added to the agenda, they are placed at the end of all other items sharing the same priority (established by the priority of the rule in the agenda item Figure 3.14(b) ① and the ordering of the agenda ②). The rule's priority is assigned by the designer when defining the module. During the design process, these values can be modified by the designer using other rules to place the corresponding rules in different location on the agenda. In a forward chaining strategy, this feature allows the designer to place more important decision rules on the top of the agenda. Individual rules and design tools can be controlled using *sponsors*, i.e., a locking mechanism. Sponsors are used to control the resources allocated to the firing of BBCAD forward chaining rules and design modules or to prevent specified rules from firing altogether. Each sponsor has a single agenda that contains forward agenda items, i.e. forward rules, and a state, which determines if the sponsor is active. A sponsor is designated, at run-time, for each design module and the rules pertaining to the design module are clustered under the appropriate sponsor. Rules are also created to control the sponsors, i.e., by enabling or disabling them. When a sponsor is disabled, i.e. locked, the rules in its agenda cannot be fired.

A form of hierarchical locking, i.e. a sponsor, is used to control individual design module and rules of the BBCAD tree-structured frame system. The BBCAD sponsors are organized in different levels of abstraction with one top-level sponsor called the "*top-sponsor*", shown in Figure 3.15. In order to reduce the chances of interference between the design modules, the design module selected for execution enables its sponsor, and

disables the rest, i.e. locking the rest of the design modules, thus narrowing the search space by firing all of the agenda items of the selected module. The various levels of knowledge locking using the Magnet modules [MagNet 85] in designing the C-core device are depicted in Figure 3.16. It can be seen as a two dimensional abstraction and this abstraction is extended from deep to shallow knowledge, i.e. from frames to rules. Rules of individual design modules are also stored in a hierarchical tree structure (Figure 3.16), and grouped in subtrees, i.e. knowledge partitioning, where only a certain subtree or subtrees of rules are applicable in a current situation. This is useful when heuristics controlling the focus of attention are utilized to achieve more efficient execution space search and speed.

3.4.3 Procedural Representation

BBCAD represents procedurally, all the knowledge which cannot be represented easily by rules, as user-defined functions evoked by the antecedent or the consequent in rules or daemons attached to frame slots. BBCAD has been developed to support design expertise (existing algorithms and procedures) written in both FORTRAN and C (Appendix F).

Certain procedures are programmed to solve specific problems, e.g., equations. Procedures are programs that know how to do things, or how to proceed in well-defined situations. Traditional numerical formulae usually map into procedures in a straightforward way. An example of a numerical formula is shown in equation **(3.1)**, and used to calculate the priority value in order to invoke the frame (*'ks'*) of the design module with the largest value. This procedure is attached to the **priority** slot of the *'ksar'* frame (Figure 3.8). f_i corresponds to the **efficiency** slot value, W_{f_i} is the value of the **weight** slot of an instance of the *'control_db'* frame (Figure 3.9), and C_{f_i} is the weight factor that corresponds to the attribute of the slot **competence**. The **efficiency** value of the

design module is a real number between 0.0 and 1.0, and n in f_n is the number of all possible focus control modules with efficiency less than or equal to the '*ks*' efficiency. Appendix D demonstrates how the priority values are calculated for the "draw2d" Magnet module. At each cycle of the design process, the priority value of the active design module is recalculated. BBCAD gives the user the power to reassign efficiency and weight factors to control and manage the design modules.

$$priority = \sum_{fi=f1}^{fn} W_{fi} * C_{fi} \qquad where \quad 0.0 \le f_i \le 1.0 \tag{3.1}$$

CHAPTER 4

Decision Analysis

This chapter concentrates on strategies which are utilised in the selection of the design module to use as this is the kernel of the BBCAD system. It also discusses the way the BBCAD controller makes a decision on the next step to take in the design process.

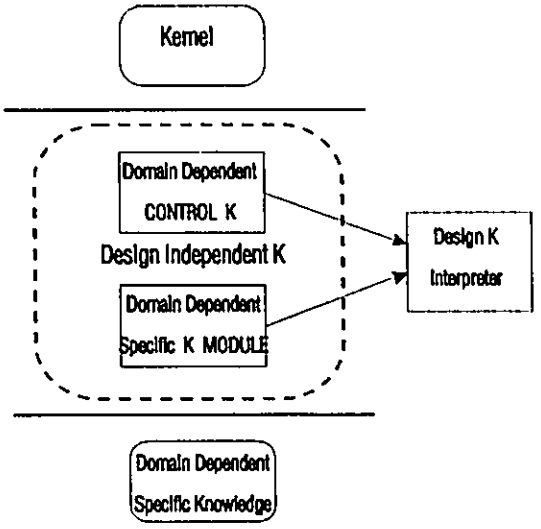

Figure 4.1: Architectural layers of BBCAD

4.1 BBCAD Organizational Levels

In the BBCAD model, the notion of separate *domain* and *control* knowledge sources has been preserved [Hayes-Roth 85(a); Prager et al. 89]. BBCAD is a knowledge-based system, containing knowledge in various representations about the problem domain and design process (Chapter 3). The problem domain consists of both problem data, i.e.

design dependent and independent knowledge (Figure 4.1), and control knowledge about how to use the problem data.

The architectural objects of BBCAD are organized into three layers, as shown in Figure 4.1. The lowest layer of BBCAD is the domain dependent level, i.e. design dependent knowledge, which contains the problem device knowledge (e.g., specifications, requirements, guidelines, etc.). The middle layer is the design module tasks level which contains the design modules and control knowledge sources, it also comprises rules of thumb and problem solvers to manipulate the design independent knowledge. The highest layer is the process control level, i.e. the kernel. It contains the control mechanism for executing and controlling the entire design process, as well as operators for initiating the tasks of the KSARs. The mechanism of control differs depending on both the structure and the steps taken. The control mechanism is implemented by the use of an agenda-based scheduler. The agenda lists all design modules that are ready to be executed, and the scheduler decides which of these modules to execute first. The scheduler calculates a priority for each waiting tool and selects the tool with the highest priority, for execution (Chapter 3).

The knowledge representation serves as a mechanism for assisting users in keeping track of their goals and for supporting cooperation among design tools by providing a central structure for knowledge that otherwise would not be easily shared.

4.2 Organization of the BBCAD Problem Solver

The approach centres around a knowledge base containing a model of representations and communications into which new design tools can be entered and classified with respect to preexisting design tools. The BBCAD *blackboard inference engine* is the system's reasoning process (Figure 4.2), i.e. the kernel. The kernel controls the

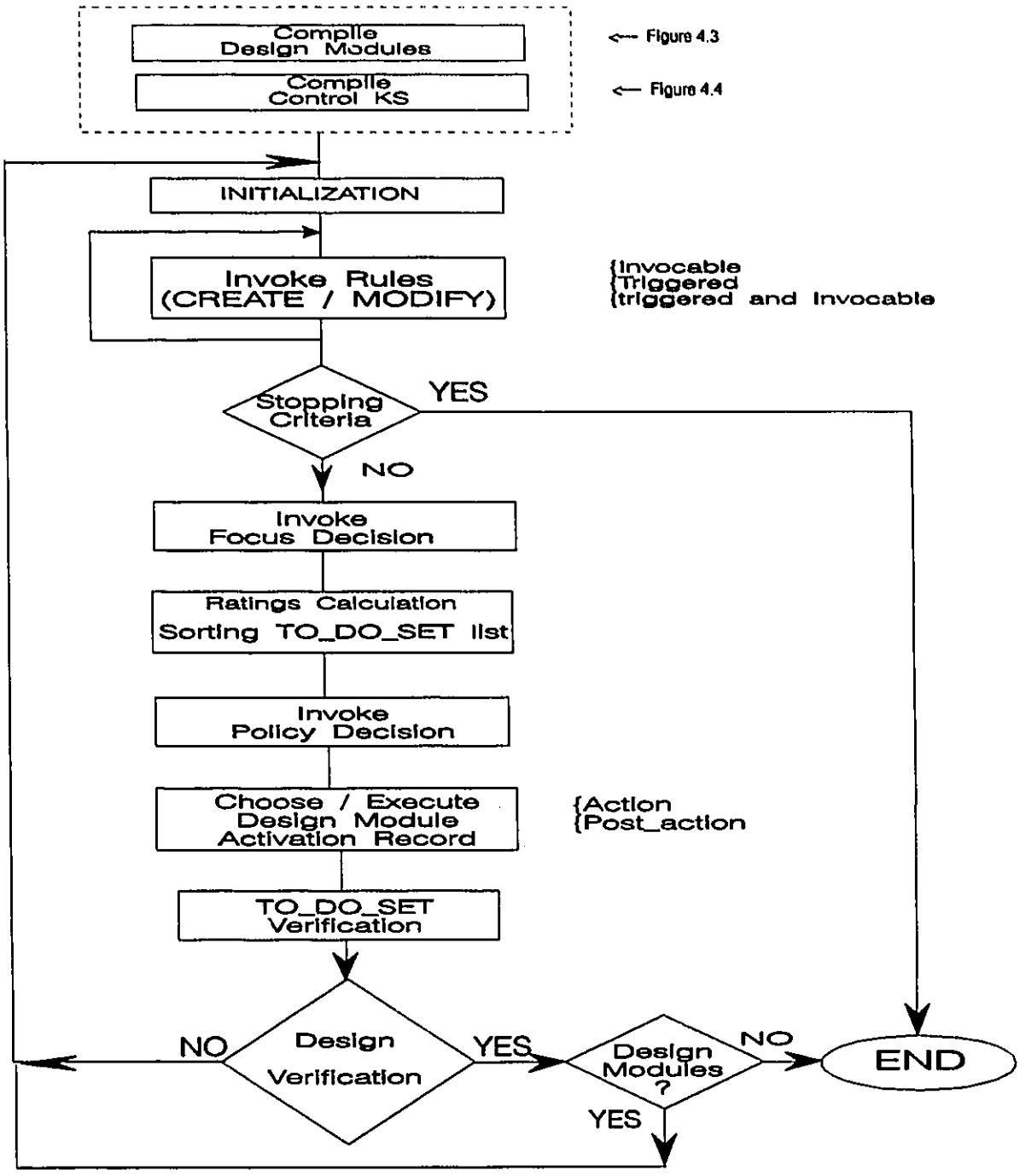

Figure 4.2: BBCAD algorithm

operation of the problem solver in the design space. The blackboard engine is a combination of LISP functions, precoded rules used to apply control decisions, and rules generated from the design module declarations which are used to verify the specified context and preconditions of these functions. It also uses external design independent knowledge modules, which contain search strategies, to guide an iterative cycle that tries to reach a final design state, i.e., an optimal design. The design solution is generated on the blackboard incrementally by applying the design knowledge modules one at a time.

The first step is to generate production rules from the appropriate slot values of the design knowledge modules and control knowledge sources (Figure 4.1). These knowledge resources are sets of elements used and modified by the kernel. They contain the clusters of knowledge required for the problem solving process. They are action task modules that contain information about when they are applicable, how their bindings are set, and what action to perform on the design space. This knowledge must be maintained in any implementation such that modifications to the knowledge can be undone and prior states can be recovered, i.e. a history is maintained.

4.2.1 BBCAD Design Module Classifications

The design modules or domain knowledge sources (*'ks'* frames), operate primarily on the domain blackboard. The domain blackboard records solution elements for the current design tools. Each design module contains knowledge about performing a particular design task in the design process, together with a data structure describing the action part, i.e. a descriptor which contains at least an identification and a condition describing the blackboard states for which that design module is defined as applicable [Velthuijsen and Braspenning 91]. Descriptors are used to quantify the ability of a particular design module to contribute to the design process; they allow the kernel unit to identify that such a design module exists. The structure implemented and discussed in

Chapter 3, allows the following information to be provided to the kernel unit:

- Design module name and a description.
- Set of triggering condition patterns.
- Set of precondition patterns.
- List of variables, parameters and relationships related to the design tool goal.
- Features and properties such as type, competence, efficiency and weighting factors.
- Pre-actions, actions, and post-actions to be taken when executing the design modules.

Table 4.1: Design modules classification

pre_cond → / SLOTS / trigger_cond ↓	Pattern	True
Pattern	**trig_and_invoc**	**triggered**
True	**invocable**	Don't Care

Since the only means of communication between the design tools is through the blackboard, a common representation scheme is implemented. To reduce the possible communications bottleneck which may occur when too many accesses to the blackboard are attempted, BBCAD decomposes these design modules into three different queues (Chapter 3), and design modules that operate on a particular level only are selected. By allowing multiple perspectives, the design knowledge modules could be structured to closely fit the design application. This decomposition allows the user to have tighter control over the execution of the design tools and to limit the search to a small portion

of the design space. Thus, by appropriately ordering the activation of the design modules, some of the excessive searches of the design space may be avoided, i.e., the system will converge faster to a solution state.

The BBCAD design module does not follow the traditional blackboard definitions, discussed in Chapter 3, nor the control strategies. The traditional control cycle can be summarized as follows [Hayes-Roth 85(a)]:

- When the trigger of a design module is matched, a KSAR is created and it is defined as being "triggered".
- The KSAR is "invocable" if all the precondition patterns of the triggered KSAR are checked to be true.
- The control mechanism decides on which invocable KSARs to execute.

The three different types of module constructs, i.e. queues, shown in Table 4.1, primarily depend on the precondition (**pre_cond**) and trigger condition (**trigger_cond**) patterns of the frame *'ks'*, discussed in Chapter 3. A pattern is a list of one or more terms, where a term can be either a value, an object, a variable, or Lisp relational functions, i.e., predicates. These patterns are also used as predicates in rules. The BBCAD control strategy cycle is summarised as follows:

- The KSARs of the design modules are created at run-time, and are defined as follows:
  1. The design module is of type **invocable** when it contains precondition patterns to satisfy, but not when the patterns are satisfied.
  2. When the trigger of a design module contains patterns to satisfy, the module is defined as being **triggered.**
  3. When the design module holds precondition and trigger patterns to satisfy, it is defined as being **trig_and_invoc.**
  4. "Don't Care": these types of modules can be used as general design tasks for control

and verification of design activities. The rules generated from these modules are fired whenever the antecedent part is satisfied.

- The above KSARs are placed on their appropriate lists. These lists are elements of the *'to_do_set'* frame structure, which is part of the general purpose control knowledge (Figure 3.12Ⓑ of Chapter 3), and they are: the **invocable_list** which is matched first, then the **triggered_list**, and last the **trig_and_invoc_list**.
- The above mentioned KSARs are "applicable" when all their patterns are checked to be true, i.e., matched and satisfied. All the applicable KSARs are then posted on the **execution_list** of the *'to_do_set'* frame, and selected for execution in the following sequence: invocable first, then triggered, and last triggered and invocable. While a KSAR is being executed, the pattern of another applicable KSAR may become false, i.e. changing its state, thus forcing it out of the executable list. However, only the KSARs on the executable list are scheduled for execution, i.e., placed on the **chosen_action** slot of the *'to_do_set'* frame.
- The scheduler selects and executes the KSAR based on the amount of data added to the knowledge space, leading to a solution state.

The above formation makes it possible to control where certain design actions direct their attention, i.e. focus of attention, that is, on which part of the design space they operate to produce a solution. The scheduling is dynamic. With each firing the design space changes and other design modules may become eligible. A design module may also cease to be eligible once another event ahead of it in the schedule has been activated. By decomposing the design modules into three categories, the degree of freedom in each design cycle is limited to only one category. Spatial partitioning is implemented by ignoring all satisfied design modules that are not part of a particular design tool. A design tool can have more than one design module (e.g., the MagNet finite element tool consists of preprocessing, solving, and postprocessing modules) and these modules could be in

different module constructs. At any one cycle of the design process, several design modules may be ready to execute, only the modules of the scheduled tool are executed if the goal is not reached. This strategy is used to prevent conflict between modules of different design tools.

4.2.2 Rule Generation

As discussed in Chapter 3, BBCAD supports a production rule knowledge representation scheme which can generate hypotheses, examine and modify design modules, deduce conclusions, execute external functions and programs, and dynamically alter control strategies. The procedure which is called to automatically generate rules, depends on the knowledge domain under consideration. A unique name is assigned to each defined rule. Each rule created has optional keywords, discussed in Appendix E, to assign attributes such as documentation string, priority of the rule, rule direction, a sponsor to which the rule is assigned, i.e. the locking mechanism discussed in Chapter 3, and a string used to tailor the explanations that are generated when asked to explain why something is true. In order to reduce the chance of interference between the design modules, the control mechanism locks onto one design module rather than the entire space, i.e. operates on the generated rules of the design module. Since the blackboard frame system is tree structured and provides an inheritance mechanism, a form of hierarchical locking is used to lock all ancestors of a design module. There are two types of rules which are being generated, and they are design knowledge modules and control knowledge modules rules. Each of which is discussed in more detail below. The attributes of a general rule construct and an example of a rule generated from a design module can be found in Appendix E.

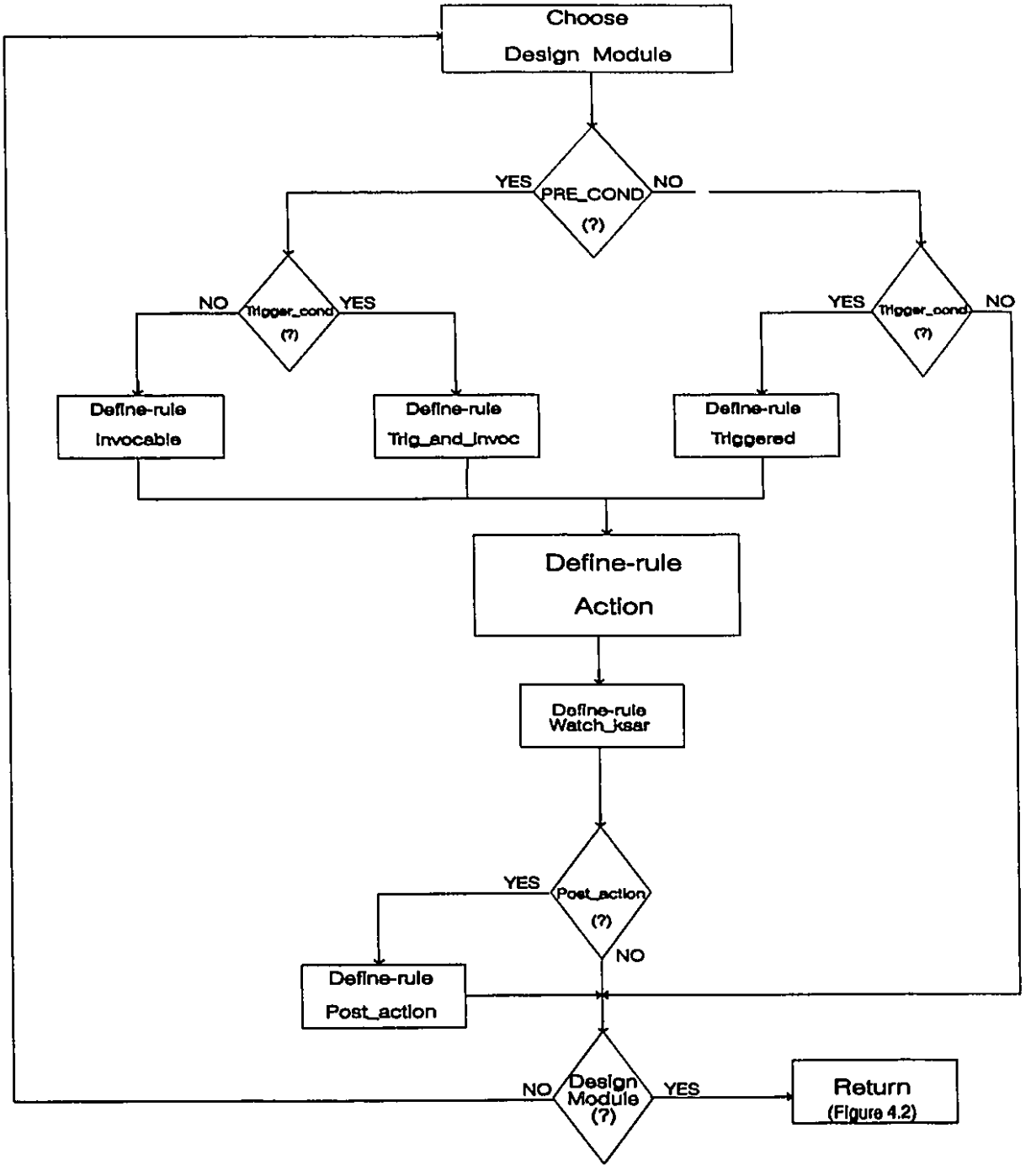

Figure 4.3: Flow diagram of BBCAD design modules classification

4.2.2.1 Design Knowledge Module Rules

The flow diagram of the rules which are being generated for the design modules, is shown in Figure 4.3. These rules get generated at startup and when design modules are included. Rules are also stored in a hierarchical tree structure allowing them to be grouped into subtrees for design module partitioning. This is useful when heuristics controlling the focus of attention are utilized to achieve efficient execution speed. The **compile_ks** procedure generates design module rules depending on the slot attributes of the *'ks'* frames. The antecedent part of these rules consists of the **pre_cond** and/or **trigger_cond** patterns (Figure 3.5 of Chapter 3). The patterns consist of calling functions, assigning local variables, comparing slot variables or local variables, and checking for facts. The consequent part of these rules calls functions, invoke simulation tools, assigns variables, asserts and deletes facts, and changes the BBCAD control states, etc. Each of the design modules contains the following categories of rules:

$$\textit{((rule.1) (rule.2) ... (rule.n))} \qquad \text{(a)}$$

$$\textit{rule.n} = \textit{((rule.n}_{\textit{name}}\textit{ (antecedent.n) (consequent.n))} \qquad \text{(b)}$$

Figure 4.4: Pre_action rules expression form

1. KSAR rule generation: this type of rule produces the KSARs of the design modules, and records are kept to indicate the invocation and modification steps that are taken at each cycle of the design process, for later use. General purpose control frames are tallied whenever KSARs are created, changed or modified, i.e. a history is maintained for the policy rules.
2. Pre_action rule generation: these rules give more control over the design module, i.e. actions to complete before invoking the design module, they resemble the precondition

of the traditional blackboard. The structure of the slot values of the **pre_action** component, shown in Figure 4.4, is a list of one or more patterns, where each pattern is a list of three elements which are the rule name, the antecedent part, and the consequent part.

3. Action rule generation: these rules deal with the patterns of the **action** slot of the design module and represent the core knowledge, i.e. strategy knowledge. These production rules contain information on what is needed to invoke and run the various simulation tools, and general rules of thumb on how to modify a design when the constraints are not satisfied. The search in the design space is guided by these heuristic rules. The sequence of rule firing is contained in the choice of the weight value, i.e. priority, given to different rules in the action slot. The structure of the slot value is in the form of lists of rules as previously shown in Figure 3.7 of Chapter 3. The slot action can have one or several rules.

4. Post_action rule generation: this type of rule has the same structure as the pre_action rules discussed above. The **post_action** slot values are the meta-knowledge of the tool. These rules will assist in the decision making of the next step, once all of the current action rules are fired.

5. Display rule generation: this type of rule is used in assessing the execution of the design modules. These rules help the user to develop new insights that facilitate understanding and help in solving the design problem, by giving explanations of the process to date, they are a history recap.

For each of the conditions there must be at least a rule in the module with that condition as a conclusion. Such a rule may be translated: "If all the input variables and all the constraints are present in the right sequence, and the script-file is executable, and the computer program is executed, and its output variables are paired with the computed values, and the computed values are asserted into the design space, then it is true that

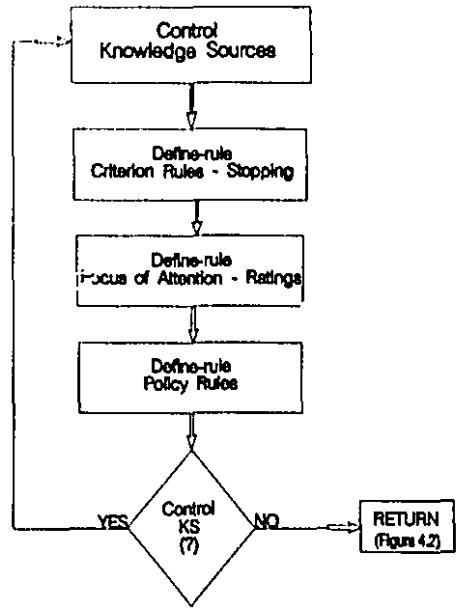

Figure 4.5: BBCAD control rule generation flow chart

results have been computed for the module." Since the rule for running the program requires that all required input values be present, the program of the module may fail to execute on the first attempt.

4.2.2.2 Control Knowledge Module Rules

Control knowledge module rules are used to sequence the execution of design modules for efficient design optimization under the control of the kernel. They operate primarily on the BBCAD spaces of design knowledge, and are predefined heuristic control modules. All solution elements for the control of the design process are recorded on the BBCAD blackboard. The automatic generation of the BBCAD control knowledge modules rules is depicted in Figure 4.5, and these rules are generated at the initialization stage.

The frame structure of these control modules is defined in Figures 3.9 and 3.10 of

Chapter 3. The generated rules use the dependency option and are specified as dependent rules, i.e. logical (Appendix E). They are used to support retraction, so that if the values and assertions that cause the rule to fire are retracted, the values and assertions generated from that rule are also retracted (if they have no other logically supporting values and assertions), i.e. an entire branch of the tree is removed. Since the rules are dependent and they run in a forward direction, they create a justification structure for new assertions and the values they assert. The justification contains the rule and the list of assertions and values used in matching the antecedent portion of the rule. There are three types of generated rules which provide decision strategies of what needs to be done to solve a design problem and they are:

1. Focus decision rules: the focus rules are generated by coding the patterns of the goal slot of all the control modules with "focus" as decision_type (Figure 3.9). These rules are a set of search strategies used by the scheduler to determine which of the executable KSARs is desirable at that cycle, i.e. are used to calculate a priority for each of the design modules.

2. Criterion rules: these rules reflect the slot value of the slot **criterion.** This criterion causes the appropriate control decision to be selected in the decision making process of rating the design modules. Processing will be halted when the design process has exhausted all of its resources without finding an adequate design solution, i.e. the design does not converge to a final solution, and the number of design cycles exceeds the required predefined cycle number (***number_of_ks***).

3. Policy decision rules: the policy rule is created when the slot **decision_type** is of type policy. These rules influence the scheduling decision and only operate on design modules with particular attributes and values. This collection of executable rules is called the "conflict set", and provides the user with several choices of tailoring the conflict resolution strategy, thereby emphasizing the focus of attention.

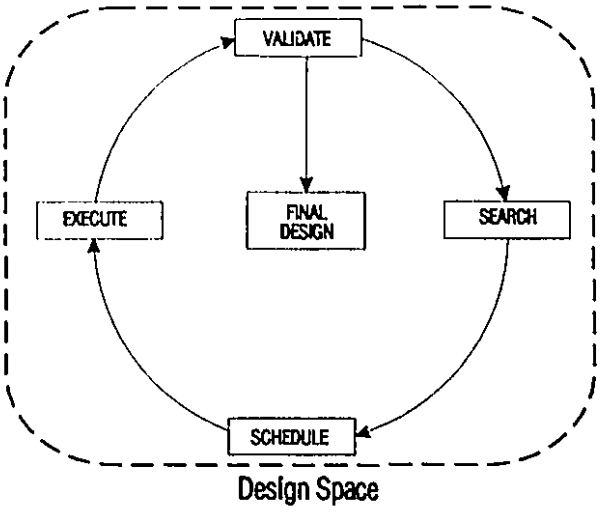

Figure 4.6: BBCAD kernel

4.3 BBCAD Inference in the Design Process

The BBCAD is capable of reasoning opportunistically because the order of executable design modules is dynamically determined based on the applicability of design modules and the latest blackboard state. A design problem is posted on the design space and the various design tools work in an opportunistic fashion to solve the problem. When the design process has proceeded to a point where a certain tool has enough information to make a contribution to the design space, that tool activates, and adds the appropriate solution to the design space for other tools to work on. At each cycle of the blackboard engine, a set of control decisions is used to select a design module for execution. The kernel cycle iterates through a basic set of four steps as shown in Figure 4.6:

- Update, validate, and examine the current blackboard state to see if a final design solution has been attained. At the beginning of every cycle, the kernel validates and examines the general purpose control knowledge modules. The following tasks are performed:

1. Cycle number update.
2. Check whether or not a final design solution has been reached.
3. Decide if there are any design modules to execute.
4. Remove all KSARs for which no goals are applicable in the design space.
5. Change the state of KSARs.

- Search for a new design module to work with: this is determined by satisfying, instantiating, and interpreting the design knowledge modules. One design module in the design space is consulted at every cycle of the design process, and the limit on design cycles is a function of the design modules and their execution. A design module could be executed several times, and its execution is determined by the following stages:
  - Instantiating: a KSAR is created for each design knowledge module on the blackboard spaces of design.
  - Checking and satisfying: after the instantiating process, every KSAR precondition and trigger condition are checked. Satisfied design modules are placed on the execution agenda list of BBCAD, i.e. agenda entries (Chapter 3 - Rule Inferencing).
- Scheduling the new design knowledge modules: a more detailed scheduling problem is shown in Figure 4.7. The BBCAD scheduler is a sophisticated mechanism that uses a variety of criteria (e.g., design module competence, triggering information reliability, design information credibility) to choose a KSAR for firing. Whilst the invocable KSARs are invoked first, then the triggered KSARs and last the triggered and invocable, the order of execution of each one of these classes is determined by the control knowledge modules (e.g., focus and policy). The focus decision rules set the ratings of the design modules in each class depending on their efficiency factor and their competence level. The policy decision rules act only on the triggered and invocable rules. Every KSAR is rated, and the highest-rated KSAR is recommended

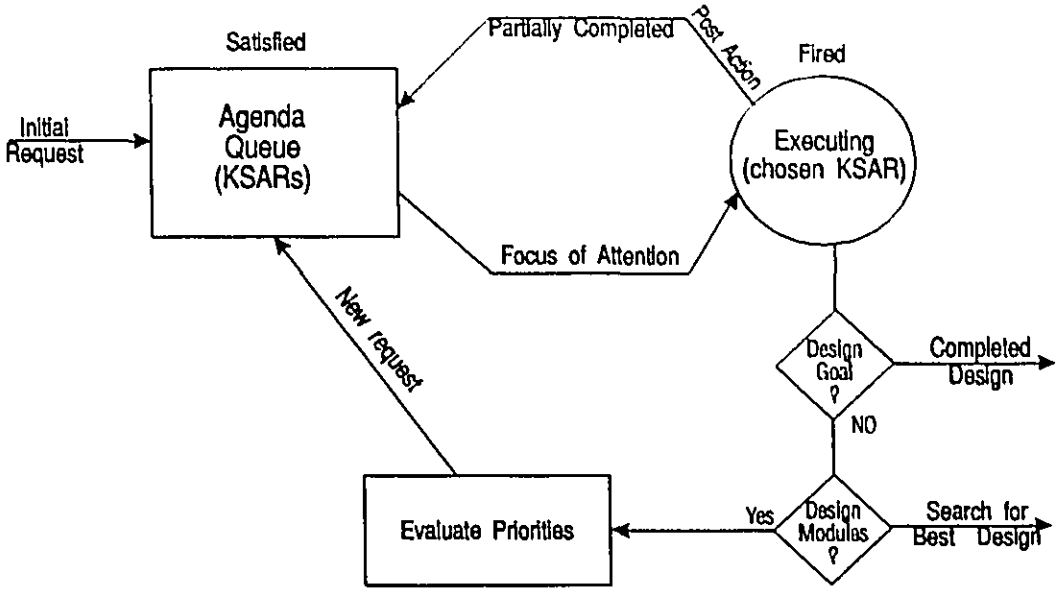

Figure 4.7: More detailed scheduling Kernel

for firing. Ratings and priority factors are assigned to each design module and are used for resolving deadlocks among competing design modules. Ratings are numerical values and a weighted sum of ratings is used to calculate a priority for each tool. An example explaining how these ratings are calculated is discussed in Appendix D. All the KSAR specific requirements for firing are tested again to verify that they remain valid.

- Firing the design module: firing the chosen KSAR is in essence forward-chaining the generated rules previously discussed, and implementing the locking mechanism, i.e. the sponsor discussed in Chapter 3, to limit the search of the design space. A more sophisticated mechanism, i.e. assigning a different priority value for each different rule construct, is utilized to control the invocation of the design tool rules and to resolve conflict among competing rules. Rules with higher priorities will fire before rules with lower priorities and the result of firing the KSAR is posted back to the blackboard. The above process repeats itself until the design problem is solved or a predefined termination condition is reached.

Conflict among the BBCAD design modules may arise when, after a goal is posted on the blackboard, more than one design module submits an estimate for the solution of the device design. The focus of attention acts as a manager resolving these conflicts, and also decides what to do in the event of a failure. If more than one design module is competing for execution, a priority value, based on the user experience and heuristics, is used for resolving deadlocks among competing modules; the default is a FIFO. If, after firing the tool, no new design solutions or events have occurred on the blackboard space, this tool will be deactivated and the next tool on the executable list will be fired. The **post_action** slot plays a salient role in the decision making before leaving the design tool if it is confronted with problems.

The scheduling mechanism in BBCAD is an example of planning. The scheduler is able to decide, in a non-deterministic fashion, which design module should be executed next in order to most effectively complete the design task. In addition, the scheduler is able to resolve conflicts between competing design tools, i.e. selects the design module that converges most rapidly to a solution, and initiate the necessary corrective action when a particular design module was supposed to be fired, but could not due to a missing program. The action slot contains rules to verify whether or not the simulation programs and their appropriate data files exist. If the program does not exist, a message is displayed pointing out the proper actions to take.

BBCAD control cycle is summarized as follows:

- At run-time, for each design module that exists, a KSAR is created, and called "instantiated", where the attributes and their values are largely inherited from the parent design module.
- The KSAR is "satisfied" if all the precondition and/or trigger condition patterns of the instantiated KSAR are checked to be true.
- The scheduler decides on which list type of satisfied KSARs to execute.

CHAPTER 5
Application

In order to illustrate the ideas already presented in this thesis, the simple problem of designing a C-core magnet device is implemented using the BBCAD hybrid system. This example demonstrates the integration of a finite element analysis tool, i.e. MagNet, in the field of electromagnetic device design with other analysis design tools, i.e. OPTDES. An optimization analysis tool is to be integrated to assist the designer in the refinement process of the final design, that is if a change is made to the C-core design, all relevant tools are re-run. The design optimization process is also discussed.

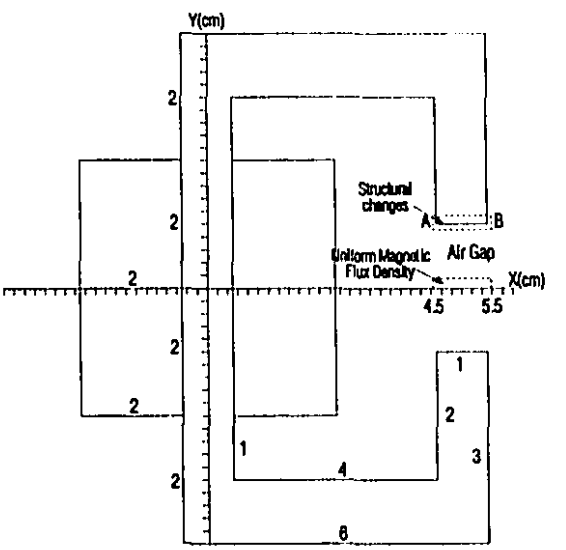

Figure 5.1: C-core Magnet

5.1 An Implementation Definition

The goal of the C-core magnet design, depicted in Figure 5.1 [Fitzgerald et al. 83], is to find a final design fulfilling a given requirement. Moreover, the last design should be optimal in some respect, for example, the physical core layout must meet the specifications and the flux density in the C-core air gap should be uniform. These may well be conflicting criteria, and therefore loosening constraints is common in design tasks, i.e. when one criterion is improved some other must be relaxed. Design knowledge can be used concurrently with optimization techniques to generate new designs to be evaluated and revised by the user, thus speeding up the design process and increasing the quality of the design.

At present, packages and conventional programs for the computational analysis and design of electromagnetic devices are in a noticeable state of growth. There are numerous well-validated computer programs on the market for static, steady state, and transient analysis. These programs are modular, with modules able to handle realistic geometries, to model materials, to provide valuable solution evaluation and pre- and post-processing capabilities, such as MagNet, PE2D, and several others [Lowther and Silvester 86]. For example, the conventional architecture of MagNet for the computer-aided design of electromagnetic devices, consists of a set of relatively large scale modules, each of which requires significant skill for its use. At each step in a design simulation using a finite element package, the user must know how to set boundary conditions, generate an appropriate mesh, select the best solver, and manipulate the field solutions generated by the solver to calculate desired quantities such as impedances, inductances, and losses, using the post-processing module included in the package. However, depending on the overall level of mesh refinement chosen by the user, an inaccurate solution may result. Should this occur, there would be a need to change the refinement and rerun the appropriate modules.

The design of electromagnetic devices has developed over the last two decades based on experience and analysis techniques [Lowther et al. 85; Rychener 88; chapter 10 of Dym and Levitt 91]. Each development in analysis techniques has allowed the designer to simulate reality more closely in testing a design. The intention in device design is to attempt to unify the experienced designer's role and the mathematical model used for analysis, i.e. design by analysis. Whilst, traditional design is based on heuristics and design formulae, the final step in the design process, having narrowed down the design space is to close in on a final design by examining a numerical model. Analysis systems provide feedback of information from the analysis to compare with the initial specifications. The results are examined in terms of these specifications and used in a feedforward fashion to re-analyze the design if sufficient agreement is not reached. Based on the results of the analysis, the designer then makes a judgement as to the acceptability of the design. Thus analysis is an assistant to design and not a replacement. It enables the designer to predict the performance of a proposed design. Design is the process of finding a solution based on the values of the analysis results.

A finite element analysis tool such as MagNet, provides information on the performance of the C-core device and operates in a space which consists of all the variables which describe the design, i.e. the analysis space. The designer seeks to optimize the C-core design by altering the shape of the core, which is a function of the analysis variables, and define a space within which an optimization is desired, i.e. the design space. Design is a search for an optimum in this space, subject to constraints which are placed on certain device parameters. An optimization system, such as OPTDES, uses the analysis package MagNet the same way the designer does: to obtain feedback on a proposed design and to explore the design space by analytical techniques.

To build an effective knowledge-based system for a C-core device design problem, one must first identify the device and define its domain. The designer of the device must

```
(DEFINE-FRAME  device  (:print-name "device"
                       :doc-string "Device specification..."
                       :is data_structure)
 (device_name    :multivalued t
                 :explanation-string "Device explanation string" )
 (design         :constraints (:one-of (none new old))
                 :explanation-string "Design state" )
 (bh_material    :constraints (:one-of (soft hard))
                 :when-modified (bh_material_modified)
                 :explanation-string "B/H Material: SOFT or HARD" )
 (bh_curve_exist :constraints (:one-of (yes no))
                 :explanation-string "B/H curve existence in MACLIB file" )
 (material_prop  :constraints (:one-of (none air copper steel))
                 :when-modified (material_prop_modified)
                 :explanation-string "Material Property" )
 (excitations    :constraints (:lisp-type integer)
                 :when-modified (excitations_modified)
                 :explanation-string "Excitations in Amperes" )
 (bound_cond     :constraints (:one-of (yes no))
                 :when-modified (bound_cond_modified)
                 :explanation-string "Boundary Conditions" )
 (h_frequency    :constraints (:one-of (yes no))
                 :when-modified (high_frequency_modified)
                 :explanation-string "High Frequency application problem (Y or N)" )
 (high_order     :constraints (:one-of (yes no))
                 :when-modified (high_order_modified)
                 :explanation-string "High Order problem (Y or N)" )
 (poly_order     :constraints (:lisp-type integer)
                 :when-modified (poly_order_modified)
                 :explanation-string "Polynomial Order of the problem" )
 (vector_scalar  :constraints (:one-of (none vector scalar))
                 :when-modified (vector_scalar_modified)
                 :explanation-string "VECTor or SCALar problem" )
 (linearity      :constraints (:one-of (none linear nonlinear))
                 :when-modified (linearity_modified)
                 :explanation-string "NONLinear or LINEar" )
 (mode           :constraints (:one-of (none real complex))
                 :when-modified (mode_modified)
                 :explanation-string "REAL or COMPlex mode" )
 (axis           :constraints (:one-of (none cartesian cart_axisym axisymmetric axiperiodic))
                 :when-modified (axis_modified)
                 :explanation-string "Coordinate system: CARTesian or AXISymmetric" )
 (field          :constraints (:one-of (none magnetic electric))
                 :when-modified (field_modified)
                 :explanation-string "MAGNetic or ELECTric field" )
 (type           :constraints (:one-of (none static harmonic transient non-transient))
                 :when-modified (status_modified)
                 :explanation-string "Status of the problem: STATic HARMonic TRANsient" )
 (data_file      :constraints (:one-of (none exists modified nochange))
                 :explanation-string "Data File status" )
 (state_device   :multivalued t                  :default-values ((0 none)) ) ;; (cycle# module)
 (rerun_device   :constraints (:one-of (yes no))  :default-values (no)
                 :explanation-string "Whether need to rerun the application" )) ;define-frame
```

Figure 5.2: Magnetic device frame structure

know how to solve it and be able to clearly express the data pertaining to the problem and its solution. The user's information can be thought of as passive and active knowledge. A fact a user knows to be true is considered passive knowledge, while a method a user uses to make deductions or inferences about the design is considered active knowledge.

Design knowledge of the C-core magnet (Figure 5.1), is classified into two groups:

① The first is the device knowledge, which refers to the full set of parameters which describe the structure and operation of the magnetic device. The C-core device frame, shown in Figure 5.2, possesses a specific set of properties based on the analysis tool MagNet that the designer has to specify. To add a new tool, e.g. PE2D, this structure could be incremented, or a new structure devised by the designer and the variable input parameters of the new tool defined. Once the frame is loaded, a user interface can assist in adding new slots to the frame structure or deleting old slots from it.

② The second, the design knowledge module (e.g., the module "draw2d" of MagNet, Figure 3.6 of Chapter 3), combines design rules, and procedural knowledge about the domain module, i.e., the "function" type of knowledge in conventional programming. A good understanding of the design tool is required to be able to supply all the knowledge needed.

In order to effectively integrate the existing tools and to provide the kind of coupling between the various analyses, the designer carefully defines the input and output requirements of the tool, and their relationships with other tools, and gives it the appropriate ratings and weights. In which case, the tools are sequenced to completely analyze and optimize the design.

5.2 Design Example

The simple problem of designing the C-core magnet, is considered to illustrate the concepts of integrating design tools to solve a problem in the design of electromagnetic devices. In this example, the problem is to redefine the physical structure, i.e. the shape, of the C-core at the air gap, such that the magnetic flux density in the air gap is uniform, i.e. $B_{max} = B_{min}$ (tesla or weber/m^2), shown in Figure 5.1. A set of optimization analysis tools are integrated into the process to explore the design space in a controlled fashion. These optimizers are relatively conventional design tools allowing multi-dimensional analysis, where the results of the analysis tools can then be used to direct the optimization procedure. One concern is that the optimization techniques involve primarily numerical manipulation, and the computational procedure always starts from a point, and with knowledge, specified by the designer. Thus, the greater the expertise of the designer, the faster the design converges to a final solution, i.e. the process is to use design rules to get near to an optimum then optimize. The feedback loop of these optimisation analysis tools can be improved by providing more accurate sets of initial design rules, thus a good design can be achieved with minimal analysis effort. The initial design can be appropriately modeled by the simple logical relationships of the rule-based system, where the rule is used as a starting point in the design search, providing those component changes most likely to improve a design. The use of heuristics in design optimization helps to decrease the size of the design space searched and consequently leads to a decrease in the overall time required to solve the design problem.

5.2.1 Design Dependent Knowledge

The frame '*device*' (Figure 5.2), depicts the domain-specific knowledge relevant to the C-core device (e.g., geometry and material). It holds the principle design components of the magnet relevant to the finite element analysis tool MagNet, and these components

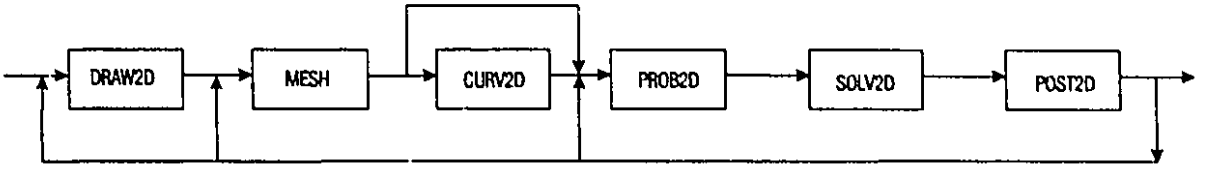

Figure 5.3: MagNet Modules

contain information about variable inputs and outputs. Variable inputs are those which can be changed to improve the design. Daemons (e.g., **:when-modified**, shown in Figure 5.2) are widely used in the device model to track down changes to the components. Whenever a value is changed (asserted, retracted, or modified), the lisp function attached to the facet is evaluated. The slot, **data_file** of Figure 5.2, alludes to the data file that contains the physical dimensions, e.g. nodal points, of the C-core. Limits on these nodal dimensions are kept in a separate file which is accessed by the optimization tool.

5.2.2 The Design Knowledge Module

MagNet [MagNet 85] is a finite element based system which provides considerable pre- and post-processing capabilities as well as analysis functions. Each module of the MagNet finite element package, shown in Figure 5.3, is stored in a separate design module. These different design modules together allow BBCAD to design a C-core device. An instantiation of the frame-based design knowledge module defines the various modules of MagNet (Chapter 3).

The MagNet package which consists of pre and post-processing modules for setting up and analyzing designs, contains the following modules:

- Draw2D: defines the basic geometry of the C-core magnet (e.g., points, lines and arcs), elements and constraints. To speed up the process, the geometry is provided using an

```
(rule_xypmh  300
        ;IF      ((instance ?ksar_solver is ksar
                                with ks_name solv2d)
                 (OR     ; VECT REAL NONL CART VERT HARD
                         (instance ?device is device
                                        with device_name ?device
                                        with vector_scalar vector
                                        with mode real
                                        with linearity nonlinear
                                        with axis cartesian
                                        with direction vertical
                                        with bh_material hard
                                        with data_file exists)
                         ; VECT REAL LINE CART VERT HARD
                         (instance ?device is device ...)
                         ; SCAL REAL NONL CART VERT HARD
                         (instance ?device is device ...)
                         ; SCAL REAL LINE CART VERT HARD
                         (instance ?device is device ...) )     ;OR
                 (equal ?device *inst_device_name*)     )        ;IF
  ;THEN          ((setf          (slot-value 'out_window 'display)
                                 (format nil "~%Running xypmh") )
                 (format "file_out" "~&Running xypmh~&")
                 (run_solv2d 'xypmh)
                 (instance ?ksar_solver is ksar
                                 with ks_name solv2d
                                 with state fired)    )          ;THEN  ) ;rule_xypmh
```

Figure 5.4: SOLV2D - an action rule

input file, elements and constraints were defined using pre-defined verbs in MagNet, i.e., commands. Drawings are stored in a text file (.SKT file) which can exist independently of Draw2D and can be edited. With the help of other script files, the automation of most of these design tools was accomplished. Upon completion of this task, the user is asked to verify all the device parameters (Figure 5.2), and whether any changes are requested. Figure 3.6 of Chapter 3 shows the specific Draw2D design module with all the slot values defined.

- Mesh: generates the finite element mesh of the problem specified in Draw2D. For this module to be satisfied, the device that corresponds to the data file (.SKT file) obtained from the previous module has to exist and the design module Draw2D has been fired.

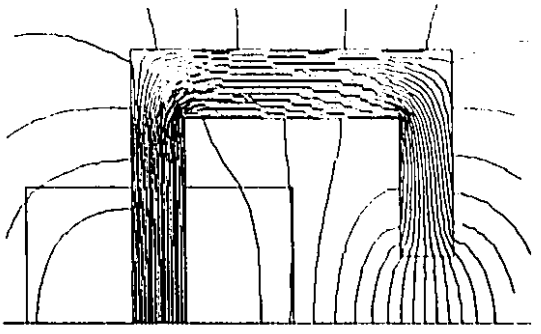

Figure 5.5: Magnetic field distribution (flux lines) of the C-core magnet

- Curv2D: builds a library of material properties that may be applied to the geometry specified by Mesh. If the material type is not in the library (e.g., table look-up), the Curv2D module will be invoked to supply the magnetic characteristics of the material. In this design problem, Curv2D was not implemented, even though it was included, because the TR66 steel magnetic characteristic, i.e. B-H curve, is defined in the library.
- Prob2D: builds a problem file ready for solution. The geometry is the same for the design, but may have differing boundary conditions, excitations, and materials in which case Mesh and probably Curv2D have to re-run. The design module contains rules related to the above conditions.
- Solv2D: solves the pre-defined problem of the C-core magnet. The solver chosen depends on the coordinate system and problem type criteria. Figure 5.4 shows an example of a rule in the action slot of the Solv2D design knowledge module. Depending on the problem type and device characteristics a solver is chosen to produce solutions.
- Post2D: derives, extracts and analyzes useful results from the solutions obtained in the previous module. This post-processing module contains rules to verify if the goals of the design are reached. The task of checking and verifying goals can be easily

automated using the MagNet verb definition facility to verify points of interest (e.g., magnetic flux density in the air region of the core). The flux plot for the simple C-core inductor is shown in Figure 5.5, with a total current of 400 amperes (for half of the problem) injected into the copper coils.

5.2.3 The Design Process

BBCAD is a hybrid system and is used to help automate the design process. To represent a specific object module or an object device, instances of the design module frame and device frame are defined. The user interface, illustrated in Appendix F, provides communications to support and to fulfil the designer's request for processing, and to fill in the appropriate information of the device. Using the front-end interface, the user loads the device design and the appropriate knowledge design module into BBCAD. At the initialization level, the user interacts with the BBCAD system to provide the necessary information to create the C-core device parameters, and this information is significant in invoking MagNet modules. This information can also be read from a file that the designer provides.

The kernel contains facilities that cause the BBCAD system to make inferences about the data. The inference engine applies the deduced rules of the design modules to the factual data in the design space when searching for a solution. It uses a *pattern-matching* facility [GoldWorks 87] and matches patterns in the antecedents of rules to patterns representing facts in the BBCAD blackboard space, and to objects that represent instances in the design space. A forward chaining, breadth-first (first-in, first-out) strategy is used to infer solutions from knowledge about the C-core device design that exists in the spaces of BBCAD blackboard. Forward chaining is initiated when the antecedent, or "if," portion of a forward rule matches a set of objects in the blackboard. The inference process is controlled so that it evaluates some rules before others, and this is qualified by

assigning certainties and priorities to rules. The certainty of an assertion is derived from the certainty of the rule that created it and the certainties of the assertions that caused the rule to fire.

Internal control rules of the design modules are generated for each of the MagNet modules (Appendix E). Draw2D is scheduled to fire first, the geometry of the C-core is accessed from an input file. Upon completing the device specifications, BBCAD will go about scheduling the various knowledge modules in order to solve the C-core design. Scheduling these design knowledge modules depends on the "**pre_cond**" and "**trigger_cond**" slots, as well as on the "**action_level**" and "**efficiency**" slots, discussed in Chapter 3. The latter two slots determine the priority values of the module which are relevant in determining the order of modules. In every cycle of the BBCAD design process, the module with the appropriate conditions, i.e. the highest priority value, will be chosen and all the internal control rules associated with this particular module will fire, thus changing and augmenting the design space of the C-core device, i.e. automating the design process. The design continues carrying out the rules of each module; once the solving phase of MagNet has been completed, the post-processing module, Post2D, is invoked to analyze the results. The solver produces a solution for the C-core problem defined in Prob2D module. These solutions consist only of the value of the magnetic or electric potential at each node of the C-core model. Post2D is used to derive the magnetic flux density in the air gap. The C-core parameters, i.e. geometry at the air gap (segment AB of Figure 5.1, A=(4.5,1.0) cm, B=(5.5,1.0) cm and the air gap is 2.0 cm), are modified and the design process is resumed till the final design goal is reached. Because of the symmetry about the X axis (Figure 5.1), it is sufficient to model only half of the C-core geometry.

The numerical analysis computations are performed through a call to the Unix shell. This call creates a process executing the named script. Standard input and output for the programs are temporarily assigned, so that the program reads its values from files

which were written using Lisp routines. Program output values are read analogously. In the device design, neutral input files were defined where data were read onto the blackboard, and changes at the end of every cycle were re-written to the file. The interfaces used allowed BBCAD to load code that is written in computer languages other than Lisp. Script files as separate processes were also used to rearrange and invoke most of MagNet modules. Appendix F discusses in more details the implementation and languages used, and the user interface.

5.3 An approach for Design Optimization

Design optimization involves the searching of design space by analytical techniques which allow a systematic search based on the variation of performance with the design parameters [Balling et al. 85]. The optimal design process can benefit from the use of heuristics, which are used to control some features of a design (e.g., complexity), based on knowledge captured during the iterative process [Arora and Baezinger 86]. While analysis software enables the designer to predict the performance of a design, optimization algorithms tends to show him how to change the variables to improve it.

Many optimization algorithms have been developed to solve general engineering design problems [Vanderplaats 84]. Careful implementation of the methods is needed, however. Each method can be defined as a separate design module with its own peculiarities and rules. Thus, heuristics can be used to accomplish the robust implementation of algorithms, allowing different algorithms to be used at various stages of the search to accelerate convergence. Knowledge of the design space and the analysis functions involved can be integrated into the design process to narrow the search space, i.e. find the best algorithm for the problem solution. Two different optimization algorithms are used to investigate the integration methodology presented in BBCAD: OPTDES and OPTMAG. Each of those will be looked at in more detail below.

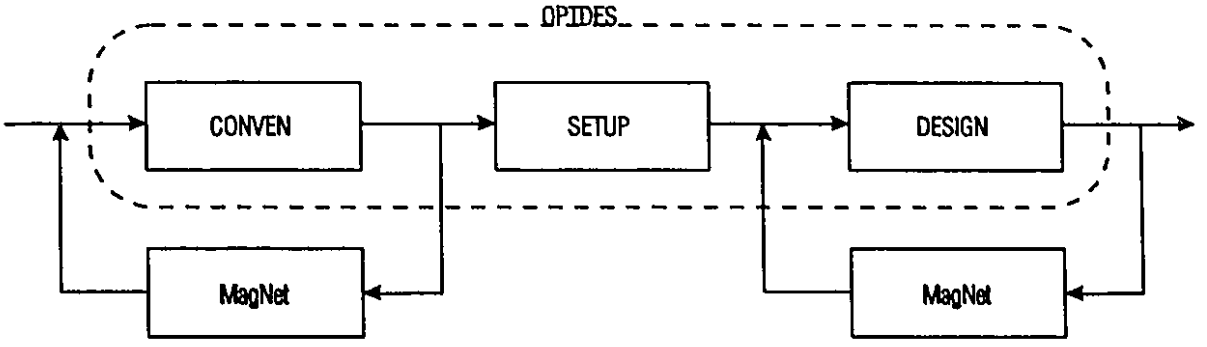

Figure 5.6: OPTDES design modules - MagNet is shown in Figure 5.3

5.3.1 OPTDES

The OPTDES [Balling et al. 85] optimization tool is a three module optimization program written in Fortran, shown in Figure 5.6. The three modules of OPTDES are represented in BBCAD as three separate design knowledge modules, i.e. instances of the device *'ks'*, and these instances are loaded at the initialization level. The first module, i.e. CONVEN, is used to interface to the user's analysis tool, i.e. MagNet tool. CONVEN is is part of the OPTDES tool and has the highest efficiency factor. It is defined as being invocable, and the precondition patterns determine when to invoke it. The three modules of OPTDES are similar to the three phases of MagNet. MagNet tool is written as a routine which OPTDES can repeatedly call. This routine contains in its argument list a vector of the analysis variables, i.e. points along the AB segment of the core (Figure 5.1) which are supplied to the pre-processor, and a vector of analysis functions, i.e. the max and min flux density in the air gap which are returned by the post-processor. This routine will load the various design modules of the tool and invoke the appropriate ones. The values of the local variables are read in when execution of the module begins. Also the analysis functions are written to a file to be processed by the optimization algorithm. The flux density of the original C-core is shown in Figure 5.5.

The second module, i.e. SETUP, defines the optimization problem of the C-core

by specifying the mapping between the design space and analysis space, i.e. the selection of design variables, analysis functions constraints, and objectives. This module needs the input and output information of the previous module. It is invoked after the CONVEN module, and is defined as being a triggerable module. In the example of the C-core magnet, the optimization problem can be stated as:

- Define the constraints on the geometry of the C-core along the segment AB of Figure 5.1, i.e. defining the degree of freedom where the point along the segment AB can move, such that the magnetic flux density is uniform in the air gap (Figure 5.1).

  This module defines the flux as a function of the geometry.

The third module, i.e. DESIGN, is used to optimize and explore the design space. This module needs the results of the previous modules. It is last for execution on the OPTDES agenda items, and is being defined as triggered and invocable module. This module repeatedly calls MagNet to reach a final design, i.e. locate an optimum. The algorithm uses a variety of information, including derivatives, to determine how the design variables should be adjusted. Once the variables have been changed, DESIGN calls MagNet to determine the new values of the magnetic flux density. Over the course of the optimization, DESIGN may need to call MagNet a hundred of times or more, depending on the number of degrees of freedom, i.e. the number of points and their constraints along the AB segment (Figure 5.1). The algorithm used for the analysis of the C-core magnet is the modified generalized reduced gradient (GRG) [Balling et al. 85; Press et al. 88; Chapter V of Gill and Murray 74], which is a nonlinearly constrained programming algorithm. The gradient is a vector of first partial derivatives. The GRG algorithms solve the original problem by solving (perhaps only partially) a sequence of reduced problems. These are usually solved by a method which uses the reduced gradient. The GRG methods use the active constraints of a problem to express certain analysis variables in terms of design variables (they are comprised of a subset of the analysis variables). The

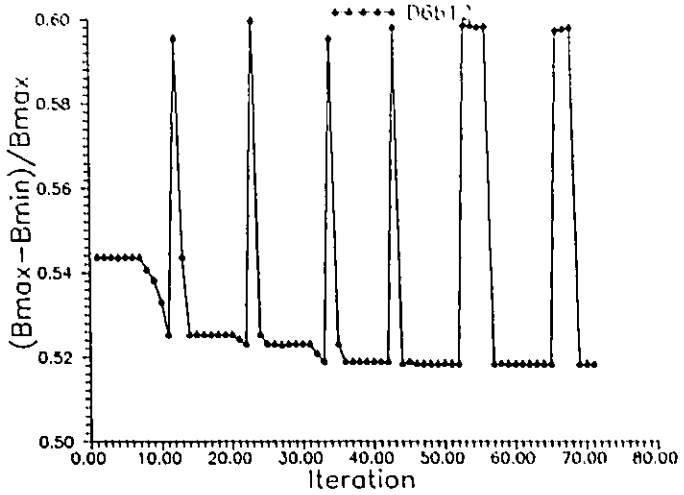

Figure 5.7: Using OPTDES optimizer in redefining the structure of the C-core

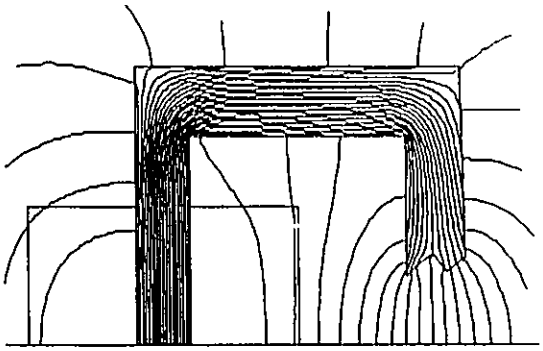

Figure 5.8: Magnetic field (flux lines) of the C-core device - GRG Method

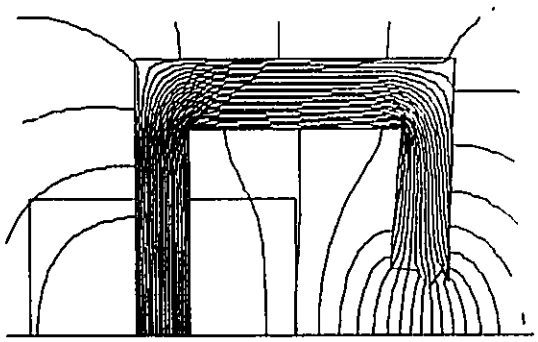

Figure 5.9: Flux distribution starting at 4.25 cm instead of 4.5 - GRG Method

function of the design variables alone becomes the reduced function, its gradient is the generalized reduced gradient. This gradient is used to determine a search direction for the design variables, and a line search determines the step along this direction. During the line search a design variable may equal or violate its constraints. If so, a new partitioning of the variables is defined, and the process is repeated on the new reduced function. If the algorithm is successful the reduced gradient will converge to minimum.

Once the analysis variables, i.e. points on the segment AB, have been changed, the optimization algorithm calls MagNet to determine the new values of the analysis functions, i.e. B_{max} and B_{min}. Figure 5.7 shows the error of the magnetic flux density in the air gap relative to the nodal variation of the segment AB. Using the generalized reduced gradient method, the final design of the C-core device and its flux distribution is shown in Figure 5.8.

Under the same conditions but different starting point, i.e. point A (Figure 5.1) starts at 4.25 cm instead of 4.5 cm as shown the previous example, there is a variation of the final design which is shown in Figure 5.9. The segment AB (4.5 cm to 5.5 cm) is divided into 4 equal parts.

5.3.2 OPTMAG

OPTMAG, shown in Figure 5.10, is another example of optimization technique written in C. It is a separate design knowledge module with its own condition and action characteristics, and is integrated into the BBCAD to solve the above C-core device design problem. OPTMAG is being defined as an invocable module with a high efficiency value and is chosen first for execution. It repeatedly calls MagNet modules which are used to run the analysis and determine the flux density. OPTMAG uses the Simplex algorithm of linear programming [Press et al. 88] to find the optimum values for the elements of the

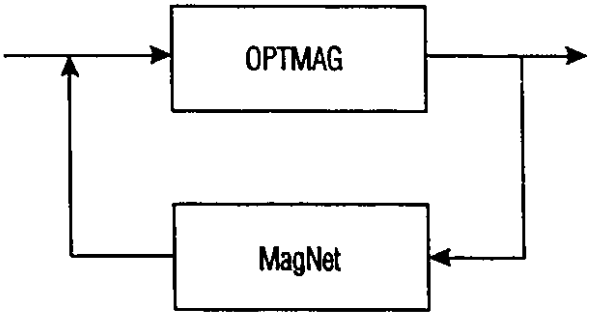

Figure 5.10: OPTMAG design module

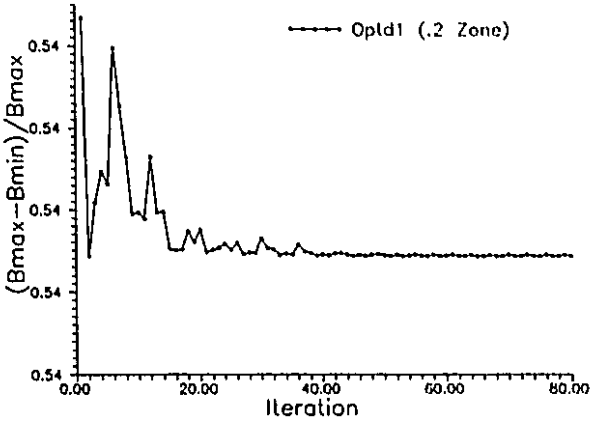

Figure 5.11: Using OPTMAG in optimizing the structure of the C-core

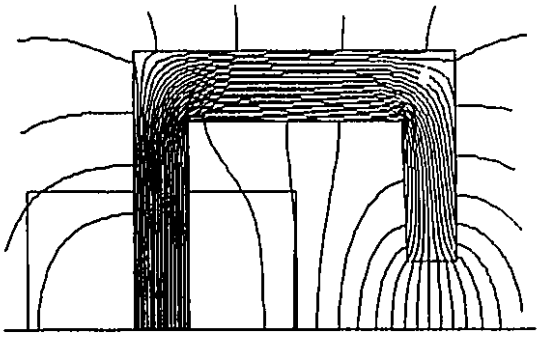

Figure 5.12: Flux lines using the Simplex Method

analysis variables, i.e. 5 nodes equally divided on the AB segment (Figure 5.1), to within a tolerance 0.0001. It is assumed that the elements of the analysis variables are scaled such that an initial step of 0.1 is reasonable. A simplex is the geometrical figure consisting, in N dimensions, of N+1 points (or vertices) and all their interconnecting line segments, polygonal faces, etc. The simplex method starts with N+1 points defining an initial simplex and then calls MagNet to evaluate the flux density at the vertices of the initial simplex (the x and y coordinates of the 5 nodes). The next step the algorithm takes is moving the variable (point) of the simplex where the function is largest ("highest point") through the opposite face of the simplex to a lower point, i.e. the reflection. This step is repeated for each new simplex. When the newly constructed step conserves the volume of the simplex, i.e., vertex persists for few iterations, the method expands the simplex in one or another direction to take larger steps. When it reaches a local minimum, the method contracts itself in the transverse direction and tries to roll down the minimum. The method terminates when the vector distance moved in that step is fractionally smaller in magnitude than some defined tolerance. In which case the simplex contracts itself in all directions and pulls itself in around its lowest (best) point.

A data translator program was used to integrate the neutral data file to the blackboard space, where all the constraints on the shape of the C-core are defined. In the optimization search the direction of the search is constrained by the limits of the points A and B of Figure 5.1. The point A is limited to ± 0.25 cm along the X direction, and B to ± 0.50 cm along the Y axis (1/4 the distance of the air gap). The changes in the AB segment are depicted in the error graph of Figure 5.11.

Figure 5.14 depicts the magnetic flux density of the final design of the C-core using the simplex linear programming method, and shows the variation of the segment AB, where the dash line is the original structure. The other points can move in both directions in steps of ± 0.25 cm. The flux plot of the simplex method is shown in Figure 5.12.

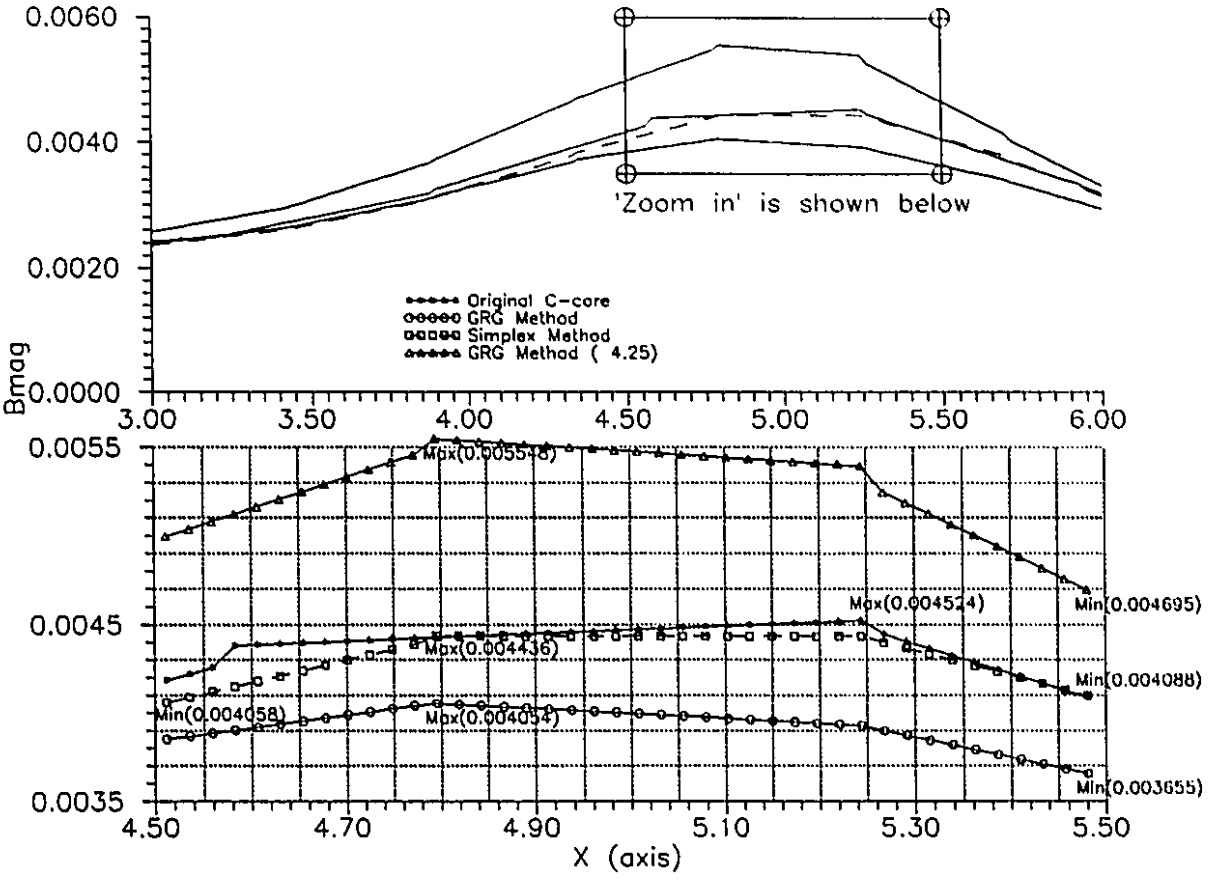

Figure 5.13: Flux distribution using different optimizing techniques

Table 5.1: Variations in optimization methods

	Max B field	Min B field	%Error	%Improvement
C-core	0.004524	0.004088	9.64	-
Simplex Method	0.004436	0.004058	8.52	11.62
GRG Method	0.004054	0.003655	9.84	-0.41
GRG Method (4.25)	0.005548	0.004695	15.37	-59.44

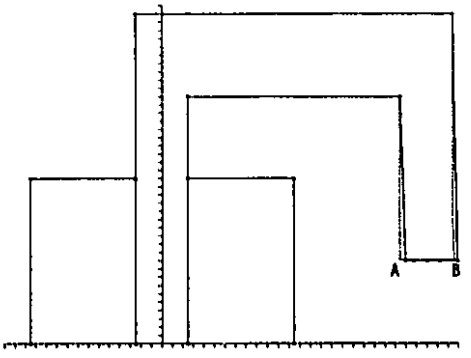

Figure 5.14: Half C-core Magnetic using OPTMAG analysis - Simplex method

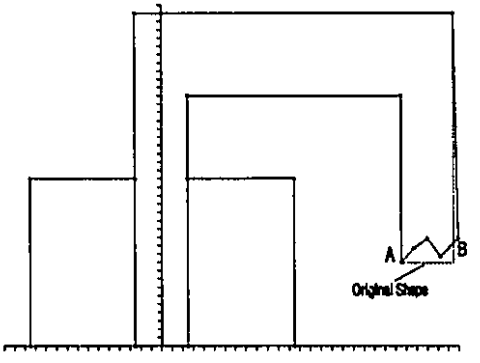

Figure 5.15: Half C-core Magnetic using OPTDES analysis - GRG method

5.4 Variations

Two different optimization approaches are used in the solution of the C-core device. Figure 5.13 shows the flux density in the centre of the air gap and a blown up of the result between 4.50 cm and 5.50 cm, i.e. segment AB of Figure 5.1. Table 5.1 presents the percent error, i.e. $[(B_{max} - B_{min})/B_{max}]$, between maximum and minimum flux density and the percent improvement relative to the original starting C-core. The Simplex method

shows a 11.62% improvement on the starting design, and the final result is shown in Figure 5.14. The GRG resulted in two different solutions depending on the starting position of the node A. The solution showed a 0.4% decrease, and gets worse when moving the node A closer to the windings. The Figure 5.15 reflects the result of the GRG method. The reason for the bad results in the GRG method is because we did not let the optimizer go far enough in the design process.

In terms of the C-core design problem, we need to point out two things:

a) The field is already fairly uniform when the design was started.

b) In real magnets, uniform fields are thought of in terms of one part in 10^5, and we have not gone anywhere near that in the test we have run. The tests are illustration of the method - not a full design.

Many researchers [Preis and Ziegler 90; Kasper 92] have reported that there is no unique "optimal" solution for the design of electromagnetic devices, and it is dependent on the method, the starting point, the degree of freedom, and the position of the nodal points. The results, presented above, showed that the determination of the shapes, sizes, and positions of the nodal points is not an easy task due to the non-uniqueness of the solution and to the method used.

CHAPTER 6
Summary and Conclusions

This research is focused on the design and implementation of a hybrid knowledge-based system founded on a blackboard architecture that is capable of providing a loose coupling and coordination between available design tools as well as the absorption of new tools in the process of designing electromagnetic devices. This chapter summarizes the main thesis results achieved and presents suggestions for future research work.

6.1 Thesis Summary

Solving the design problem of electrical machines is a complex activity involving a number of analysis design tools and a number of choice methods potentially available for each tool. BBCAD is a hybrid system that utilizes knowledge-based system (KBS) technology for engineering design to integrate various design tools and to increase robustness and integrity of the design process of magnetic devices. "Hybrid" is a concept that is achieved in several dimensions: representation of the design as well as the design space, modeling knowledge of design and design activities, applying analysis tools to assist in the design process, integration of several design tools as well as the results of these various tools, implementation languages, problem solving strategies, i.e., the scheduling processes, etc. The design of a C-core magnet was used to demonstrate the integration problem of electromagnetic device design.

Among the significant results of this thesis work are the models of knowledge brought to bear on the structuring of the design modules, in the example used these were the MagNet modules. A flexible structure for coupled systems is presented, based on the

combination of the concept of frame-based systems and blackboard architecture. The frame-based structure is used as the representational approach for the design-dependent and independent knowledge modules. BBCAD employs the blackboard as a global database through which cooperating design modules communicate with each other. In BBCAD, the various goals and intermediate results are stored on the blackboard (e.g., domain and control), and implements a control mechanism and a scheduler which prompt the firing of the design tool, the generation of new designs, and modification of knowledge in the design space. The structure of BBCAD allows for an incremental implementation of a coupled system, starting out with a shallow coupled prototype which can be generated rapidly. As more design tools, i.e. design modules, are added the depth of the coupling increases. The BBCAD structure provides for reliable rapid prototyping, flexibility concerning changes, as well as extensions, and increase of coupling. The transparency of the BBCAD system is improved by providing explanations and debugging facilities to understand the design process. Thus, the use of a flexible design representation has given rise to a much better way to manage and manipulate the design space.

The core of the BBCAD architecture is the kernel, i.e. problem solver, whose main functions are:

a. To provide a scheduling mechanism for firing and deletion of processes, i.e. design modules. The scheduler's task is to examine all executable design tool operations and decide in which order they should fire. The scheduler uses many criteria to arbitrate between design tools; it includes process priority, whether or not the expected execution resources facilities are available, and whether the design module activation record would satisfy a pending goal.

b. To provide synchronization among the design tools, i.e. a conflict resolver.

c. To provide communication tools, so that design modules can communicate with each other.

d. To develop design solutions through a generate-validate-modify paradigm, i.e. design by analysis.

The BBCAD open-ended architecture utilized facilitates the development of the design knowledge modules and their integration through the blackboard space. It implements a way of ordering rules within the design knowledge modules, i.e. by assigning rule priority, and provides a locking mechanism which makes it easy to search the design space for design solutions. The BBCAD user interface developed provides the designer with the facility to change data and re-run.

The structure of the problem space for electromagnetic device design is decomposed into three different levels, thus allowing the designer to have control over the execution of the design modules and to limit the search to a small portion of the design space.

Solving the design of a physically simple electromagnetic device, such as the C-core inductor, seems to be relatively straightforward. However, its design parameters are numerous, highly interrelated, and diverse, thus preventing a closed-form solution to the design problem. For example, dimensions and shape of the core influence the flux density in the air gap. Choosing an optimization algorithm to determine the shapes, sizes, positions of the core, permanent magnet and windings which produce a uniform flux distribution in the centre of the air gap is not a simple task, i.e., the problem of finding a unique "optimal" solution. The result is dependent on the approaches used in the solution.

The problem-solving of the C-core magnetic device integrates symbolic and numerical computing and heuristics to solve the design task. Symbolic computing techniques are used in design problem formulation and in the interpretation of results but the actual design tools are written using numerical procedures, and heuristics are used as a way to define performance criteria of the design. BBCAD demonstrates that the

blackboard is an effective means for automatic execution of design tools to meet a specified set of design requirements, i.e. automating and optimizing the design problem of a C-core magnet. BBCAD is an implementation of an example of a hybrid environment.

A blackboard system is similar to an operating system. The basic resources of a blackboard system are provided by its control mechanism, knowledge module, and a global database. The blackboard architecture provides the means for the proper use of these resources in order to solve a particular kind of design problem. It can be noted that blackboard architectures have many features and constructs which parallel the *operating system* paradigm [Peterson and Silberschatz 85]. The core of the blackboard architecture is the control mechanism which mediates between the subprocesses (global database and design knowledge modules) competing for processing resources, and controls their execution. It is similar to the *kernel* whose main function is to provide file systems, scheduling, synchronization, and communication mechanisms for the various processes. Similar to "interrupt-driven" mechanisms in operating systems, the knowledge sources respond opportunistically to the changes on the global database in a blackboard system.

Designing is an opportunistic process and is accomplished at various levels of abstraction. BBCAD is an example of exploring the utility of the blackboard model in the area of electrical machines design. Design problems have large solution spaces and require many, diverse cooperating design tools. The major difficulty with these design problems might lie in the designer's limited understanding of how to represent and reason about spatial relationships. If the initial organization of the knowledge spaces is wrong, then modifications result in a rapid deterioration of the design structure. Like any architecture, the blackboard model is not without faults. For certain types of design problems with closely coupled relationships, the "advantage" of modularity becomes a drawback. The separation of components encouraged by modularity can tend to mask these relationships. Also, the blackboard control architecture may probably be

inappropriate to build high-performance application systems, because its decision making speed, which may be too slow.

BBCAD runs on a single computer, but it is conceivable that the integration of several processors could be a very important factor in improving the performance of knowledge-based systems in electrical machine design. The BBCAD structure lends itself to a certain amount of parallelism and can greatly gain from the presence of several distributed processors.

6.2 Suggestions for Future Research

As future work in the field of integration of design problem-solving, the following are suggested:

- Create and integrate more design knowledge modules to fully compete and test the conflict resolution strategy and the scheduling mechanism adopted in BBCAD.
- Investigate ways to model organizational structures for distributed design problem-solving [Smith and Davis 81; Yang et al. 85], e.g., parallel processing [Fennell and Lesser 77].
- Develop an inferential generator to manipulate the use of heuristics based on knowledge captured during the design iterative process, i.e., a model for learning.
- Find a formulation for mapping design applications onto the blackboard architecture. It is necessary to be able to decide whether the blackboard architecture is appropriate for a specific design problem.
- Formulate a formal description, i.e. a methodology, for the blackboard architecture based on its similarities and equivalence with operating systems.

APPENDIX A
Knowledge-Based Systems

This appendix attempts to provide a background on knowledge-based systems (KBS) in engineering design.

A.1 Basic Architecture

Knowledge-based systems are computer programs in which an attempt is made to capture and render operable human knowledge about some domain [Buchanan and Shortliffe 84]. The goal is to represent knowledge in such a way that it is comprehensible to human and machine. A typical structure of a KBS consists of an inference engine, a knowledge base, and a workspace , as shown in Figure A.1.

The *knowledge base* contains the basic (declarative) knowledge of the specific domain, including facts, beliefs and heuristics, i.e. rules of thumb. The *workspace*, or working memory, contains the specific problem data (supplied by the user or inferred from the knowledge base during a consultation) that reflect the current state of the problem solution. The knowledge base builds up dynamically during the solution process of a particular problem. The use of that knowledge is governed by a control strategy stored in a separate *inference engine*. The inference engine incorporates reasoning methods, and acts upon the input data and the information in the knowledge base to solve the problem. Thus, the inference engine mechanism acts as the executive that runs the design system. It performs actions that lead to a solution of the design problem, and at the same time, may change the knowledge base by adding to or modifying the information contained

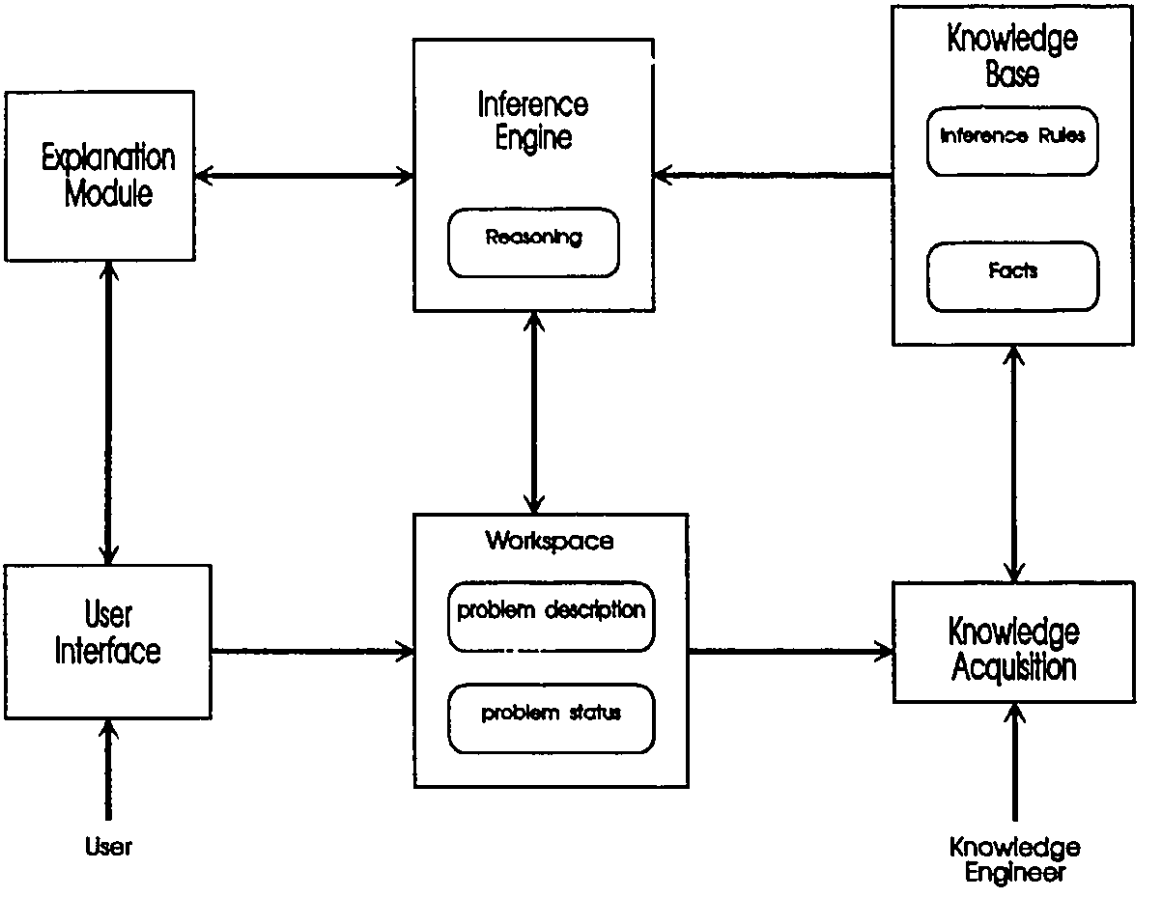

Figure A.1: A Typical KBS Schematic

therein. The inference mechanism often uses rules to infer new information about the current state of the design [Dym and Levitt 91]. An *explanation module* produces explanations of the inferences used by the knowledge-based system, such as why a certain fact is requested, or how a conclusion was reached. A *knowledge acquisition* facility allows the system to acquire more knowledge about the problem domain from knowledge engineers - experts. The *user interface* (I/O facilities) allows the user to communicate with the system. It usually provides a command language for directing the execution of the system. It is responsible for translating the input as specified by the user into the form used by the knowledge base.

Knowledge-based systems are problem-solving programs whose performance depends strongly on the use of facts that express valid propositions, heuristics that express rules of good judgement in situations where valid algorithms generally do not exist - normally performed by an expert, and beliefs that express plausible propositions [Mittal and Araya 86]. KBSs reason to solve design problems by generating and testing all possibilities until a solution is found. Usually, solutions involve applying heuristic rules to given data in order to deduce logical or probable consequences and prove that these consequences satisfy the goal. Some of them have been specifically developed to deal with ill-structured problems (uncertain, incomplete, inexact and unformalized problems). In contrast to algorithmic programs which follow step-by-step procedures, KBS are free to search through and reason about the knowledge in order to reach a goal.

A.1.1 Advantages

Building a KBS to aid the design process can be approached by building individual modules to handle problems posed in the various components of the design. The resulting modules are integrated so that results can be communicated and the overall process can be repeated as designs are refined and improved. The following is a list of advantages when using KBS for performing symbolic manipulation procedures in BBCAD:

- Ease of development and maintenance: the modular nature of the language used [GoldWorks 87; Winston and Horn 84] allows flexibility for exploring variations, experimenting, etc., whilst the symbolic, interactive, and high-level features of BBCAD allow the designer to expand the design after the prototype stage.
- Friendly environment: due to the symbolic nature of the expert system shell used [GoldWorks 87], implementation of an intelligent, friendly, and personalized user interface for BBCAD was made possible and easier.

- Representational advantages: knowledge is programmed explicitly. Self-understanding and explanation are made possible to follow, e.g. rule justification. Meta-knowledge is subject to easy change and modification. When improving the design process only the knowledge is modified, not the inference mechanism.
- Rapid checking of design concepts allows more alternatives to be considered in a short time period and permits easier incremental improvement to the design.

KBSs are also typified by a collection of other properties, many of which are detailed in [Rychener 88]. The general techniques which have been developed in artificial intelligence research [Barr and Feigenbaum 81, V.1] provide many of the basic tools [Winston and Horn 84; Steele 84] necessary for KBSs in engineering design.

APPENDIX B

Blackboard Systems

This appendix attempts to provide a background on the blackboard architecture and reflects on the use of this architecture in engineering design. It also presents an overview of variations derived from the formal descriptions of a number of influential blackboard architectures. Figure 1 of [Nii 86(a)], August 1986, shows a general chronology and intellectual lineage of the various applications and skeletal systems. It also includes some of the "better-known" and documented systems.

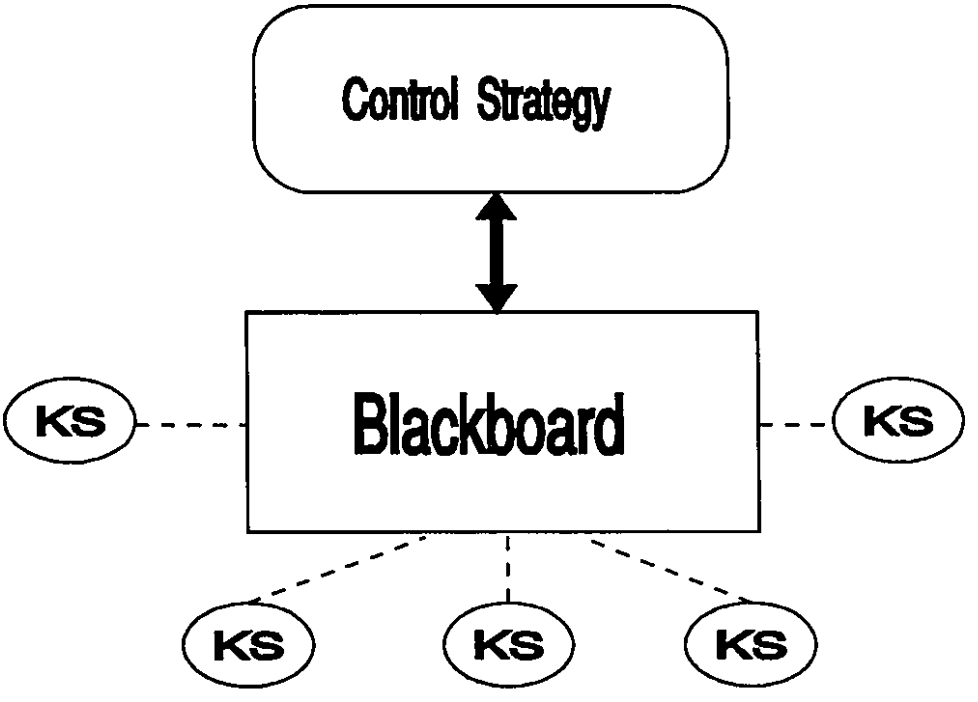

Figure B.1: A Typical Blackboard Model

B.1 Basic Architecture

The blackboard model [Lesser et al. 75; Nii and Aiello 79; Nii 86(a)] consists of three major components, as shown in Figure B.1:

1. **The Blackboard** - knowledge storage and communication: the blackboard is the source of all the data structures on which design knowledge modules, i.e. knowledge sources, operate, and the destination for all conclusions from knowledge modules, thus it represents the task domain. The blackboard data structure is often referred to simply as *the blackboard*. It represents the channel through which the design modules communicate their findings to each other. The data structures used for the blackboard is influenced by the design problem.

2. **Knowledge Sources** (KS) - specialist: the KSs represent expertise about some aspect of the design problem. They contain the knowledge of the task domain pertaining to the problem being solved. Each KS contributes information that leads to the current state of the design. Whatever form of representation (e.g., frame, rules, logical relationships, etc.) is used within a KS, however, the knowledge reflects an action, i.e. a change to the blackboard, under appropriate circumstances. By definition, KSs only modify the blackboard data structure and cause state transitions.

3. **Control** - problem-solving strategy: the control component contains the mechanism that influences the selection and execution, i.e. firing, of the KSs from among those qualified, and allows the specialist to place numerous data on the blackboard. It guides the design process by choosing and then activating appropriate KSs. Control is based on opportunistic reasoning that can apply KSs in either a forward or backward direction, or a combination of both.

The blackboard model provides a way in connecting individual separate pieces of design tools into a large intelligent program. These design tools are stored in independent

modules, each of which monitors a small region of the design space and is activated only when the tool is needed in the process. The blackboard model serves as an excellent integration framework for combining diverse expertise (dissimilar pieces of code) and problem-solving techniques in the design process. The model is analogous to a committee of designers standing around a blackboard; each is able to read everything that is on it, and to decide when he or she has something worthwhile to add to the design space.

The blackboard paradigm is a powerful technique for implementing applications requiring multilevel reasoning, flexible control, or the integration of diverse problem solving expertise into a common framework [Nii 86(a)]. Thus, it is noted that some of the blackboard architectures are used as blackboard *frameworks* (BB1 [Hayes-Roth 85(a)], GBB [Corkill et al. 86], and ATOME [Laâsri and Maître 89]), while others occur as blackboard *applications* (HEARSAY [Lesser et al. 75; Lesser and Erman 77]; DECADE [Bañares-Alcántara et al. 87; 88(b)]. Typically blackboard frameworks allow more variation of their components than blackboard applications, where for most aspects a choice has been made for a specific option.

B.2 Options and Variations in Blackboard Systems

Although a blackboard concept was documented in Artificial Intelligence (AI) literature as early as 1962 by Newell [Newell 62], it was implemented as a system a decade later by people working on the HEARSAY speech-understanding project. This section presents an overview of options and variations derived from the formal descriptions of existing blackboard architectures. In [Velthuijsen and Braspenning 91], the authors developed a formal description of blackboard architectures and used this formalisation to describe a number of rather influential blackboard architectures. The list of examined blackboard architectures is not complete. A complete analysis of all existing blackboard architecture is obviously beyond the scope of this thesis.

B.2.1 Blackboard

In early blackboard systems, the structure and organisation of blackboard contents were subdivided into different levels, i.e. dimensions, or ordered sets of levels, i.e. panels. HEARSAY [Lesser and Erman 77] levels are ordered in a loose hierarchy (problem-specific), and are divided by time, representing the temporal order of the utterance being examined [Erman et al. 80], and DVMT [Lesser and Corkill 83] uses an abstraction hierarchy between the different panels. At a later implementation, this kind of organisation was extended towards further specialization in the form of named structures (such as *defstructs* [Nii and Aeillo 79]), classes, i.e. frames/objects, in a class hierarchy (e.g., GBB [Corkill et al. 86], ATOME [Laâsri and Maître 89], CAGE [Aiello 86], POLIGON [Nii 86(b); Engelmore and Morgan 88, Chapter 25; Jagannathan et al. 89, Chapters 7 and 18]), and even a relational database (HEARSAY-III [Erman et al. 81]). The basic units, i.e. elements, containing the information on the blackboard are either *data types* (e.g., HASP/SIAP [Nii et al. 82]), sets of *attribute-value pairs* (e.g., AGE [Nii and Aiello 79]), or hierarchically ordered elements (GBB units are ordered according to dimensions according to their attributes). Certain blackboard architectures provide facilities for representing links as attributes of units, i.e. semantic nets. Explicit representation of *directed* (AGE, CAGE, and HASP/SIAP) or *labelled* (GBB, ATOME) links facilitate their use in consistency maintenance and in event specifications.

B.2.2 Knowledge Sources (KS)

This section addresses the variation of KSs with respect to the following components: KS descriptors, instantiations, condition parts, and action parts.

The most simple information kept about a KS in a KS descriptor is a name (e.g. BBCAD uses "draw2d" as a design knowledge module descriptor). However, in most

blackboard architectures more information about a KS is represented. This information can be subdivided into knowledge about how the binding of a KS with the context in which it became applicable occurs, and knowledge that can be used for control. Binding occurs upon instantiation and execution of immediate code, e.g. HEARSAY-III, or by the precondition of a KS, e.g BB1. A precondition is that part of a KS that determines applicability of the KS by using blackboard state knowledge solely. The results are stored in special data structures representing an instantiation (stimulus frames and response frames in HEARSAY-II, or KS activation records (KSAR) in BB1).

A subdivision of the condition part is made into a trigger (specifying *events*, lists of tokens in the case of AGE and CAGE) and a precondition (specifying *states*, rules or procedures as in BBCAD). Most blackboard architectures allow either rules or procedures as the action part of KSs. The extensions here are BBCAD and HASP/SIAP where rules, procedures and frames can be used for the action parts. In AGE, CAGE, and BB1 the way in which rules are fired can be specified by the user. CAGE went one step further by allowing the user to specify when rules, clauses in rules, and statements within clauses can be evaluated in parallel.

B.2.3 Control Mechanism

The variation of the control mechanism addresses the following aspects: basic cycle (control vs. KS), concurrency, matching, selection, and stopping criterion.

- **Basic cycle**: the **control loop** is formed by a cyclic recurrence of the following steps: examination of the state of the blackboard, determination of applicability of KSs in view of the current state, and execution of the applicable (if any) KSs. BBCAD, BB1, AGE, HASP/SIAP, and GBB use a control cycle mechanism. The **KS loop** is formed when the design modules perform a cycle of alternately monitoring the blackboard for

opportunities specified by activation-patterns, and contributing to the design process when a situation has occurred. When such an activation-pattern occurs the associated function is executed immediately. AGORA [Bisiani et al. 87] and POLIGON [Nii 86(b)], where control is not an issue, are more reminiscent of the original blackboard metaphor than many blackboard architectures in the sense that KS instantiations monitor the blackboard for opportunities.

- **Concurrency**: the control becomes more complicated when dealing with concurrent blackboard systems. The distinction is made here between *parallel* and *distributed* blackboard architectures. Parallel blackboard architectures are characterized by a shared-memory blackboard and concurrently executed KSs (e.g. CAGE and POLIGON). In CAGE the execution of KSs can be synchronized. In a distributed blackboard architecture, each node has a separate local blackboard containing objects created locally as well as copies of objects received from other processing nodes (Chapter 6 of [Jagannathan et al. 89]). Because of the independence of knowledge modules, parallel processing can be achieved by having several triggered knowledge tasks all executing in parallel on separate processors. Although many of the same problems which haunt other parallel and distributed applications are of concern here as well, the overall architecture of the blackboard model is at least supportive of a parallel processing approach [Corkill and Lesser 83].

- **Matching**: a distinction is made between blackboard architectures which take events and try to match these with the KSs in the system (*event-based* matching, e.g. BB1, GBB, AGE) and blackboard architectures which consider KSs and search for events that match (periodically) the KSs (*KS-based* matching, case of CAGE) for determining the applicability of the KSs.

- **Selection**: the KS selection is a mapping determining a subset from an existing set of KS instantiations. This is associated with the kind and amount of knowledge used for

selecting activities for execution. BBCAD uses a priority calculation for all KSARs based on the ratings (*strategy*, *focus*, and *policy* levels [Hayes-Roth 85(a)]) and a rule for integrating these ratings. Examples of selection mapping with restricted domains are: (i) select that KSAR whose corresponding KS has the highest priority, i.e. a simple best-first strategy, (ii) select that KSAR most recently generated, i.e. a depth-first strategy, and (iii) a select that KSAR generated earlier during the problem-solving process, i.e. breadth-first strategy. HASP/SIAP, AGE, and CAGE employed a *blackboard-based* selection of activities, i.e. event-driven, where events are changes to blackboard levels. Blackboard-based selection uses mainly knowledge currently present on the blackboard.

- **Stopping criterion**: most blackboard architectures halt automatically when there are no more applicable KSs and no KS is being executed. BBCAD includes a procedure in the control cycle for deciding whether a certain stop criterion has been met. In AGE and CAGE, this stop criterion is formulated in terms of the contents of the blackboard. HEARSAY-II has a special KS that can discourage pending activities and is invoked when an interpretation on the highest level emerges, effectively clearing the queues (KS actions) and thus halting the problem-solving process.

APPENDIX C
Language Review

There are many possible implementation procedures available inside the knowledge-based system domain (also refer to [Bañares-Alcántara et al. 87]). These may be grouped into:

C.1 General Purpose Programming Languages:

A knowledge-based system may be built using a general purpose programming language. Although in principle any language can be used, only few are convenient because of their flexibility, appropriateness, availability, and portability:

- LISP and dialects [Winston and Horn 84] (e.g. FRANZ LISP, INTERLISP, COMMONLISP, SUNLISP, etc.).
- PROLOG [Clocksin and Mellish 85].
- C++ [Stroustrup 86], C [Kernighan and Ritchie 78].
- FORTRAN [Katzan 78].
- PASCAL [Jensen and Wirth 74], etc.

Lisp has been widely used for most the work in artificial intelligence. Although terms (symbolic expressions) have no direct meaning in Lisp, the Lisp program can manipulate such expressions. Most Lisp systems are interpreted and therefore provide an interactive environment for the development of Lisp code, which greatly facilitates the development of knowledge-based systems.

C.2 General Purpose Representation Languages:

A general purpose representation language is a programming language developed specifically for knowledge engineering. These languages provide constructs that aid in the development of a knowledge-based system that do not exist in conventional programming languages. Some of these are simply expert system shells (e.g. an inference mechanism with an empty knowledge-base). Usually a knowledge acquisition module is available to assist in developing the knowledge-base and to structure the context for a particular application domain Such examples as:

- GWII [GoldWorks 87]: hybrid knowledge-based expert system development tool based on rules and frames built on the top of Common Lisp.
- OPS5 [Forgy 81]: rule-based representation language built on the top of LISP to facilitate the use of production rules.
- Knowledge Craft [Knowledge Craft 85] by the Carnegie Group: frame-based knowledge representation language with procedural attachment and inheritance.
- SRL [Wright and Fox 83]: the Schema Representation Language provides specialized constructs for the representation knowledge.
- HEARSAY-III [Erman et al. 81]: programming tool designed for representing and applying diverse sources of knowledge to a problem area. It provides primitives for developing a blackboard data structure which would be accessed, and used by the knowledge sources.

C.3 Skeletal Systems:

They are knowledge-based systems which originated from previously built systems [Nii 86(a)]. One common practice is to generalize a successful knowledge-based system

and to try to make it domain-independent, using this generalization in the construction of a new system. Most knowledge-based systems were developed from domain dependent systems by stripping out the original knowledge base. In general their convenience is directly proportional to the similarity of the original design and the proposed new concept. These skeletal systems consist of the basic system module from which application systems can be built by the addition of knowledge and the specification of control (metaknowledge) [Nii 86 (a)], i.e. expert system development tools. Examples include (→ indicates building application from skeletal system):

- MYCIN [Davis et al. 77] → EMYCIN [vanMelle et al. 81] → PUFF [Feigenbaum 77]
- AGE-I [Nii and Aeillo 79] → HANNIBAL [Brown and Buckman 82]
- AGE-I [Nii and Aeillo 79] → BB1 [Hayes-Roth 85(a)]
- BB1 [Hayes-Roth 85(a)] → PROTEAN [Hayes-Roth et al. 86]
- GBB [Corkill et al. 86] → GBB1 [Corkill et al. 87]
- BB1 [Hayes-Roth 85(a)] → ATOME [Laâsri and Maître 89]

The use of an available knowledge-based skeletal system can greatly facilitate the development of new systems since many of the representation issues and control strategy decisions are already incorporated in the model. The use of a such systems, as opposed to a general programming or representation language, involves a certain loss of generality. However, it provides the advantage of building a complete system with knowledge acquisition and explanation facilities in relatively little time.

APPENDIX D

Priority Ratings

The Table D.1 below shows the various instances with the corresponding weight values and the weight factor of the competence slots. Figure D.1 shows an instance of the *'control_db'* frame, where the slot-values of the slots **weight**, **competence**, and the **goal** (f_i) are shown below in Table D.1. The *'integration_rule'* frame, shown in Figure D.2, binds the corresponding **competence** slot-value to a weighting factor that is used to calculate the priority value for a design module.

Table D.1: Weight table for priority calculation

Instance	f_i	Weight	Competence	Weight Factor
focus_1	≥ 0.25	10	class_1	1
focus_2	≥ 0.45	20	class_2	2
focus_3	≥ 0.65	30	class_3	3
focus_4	≥ 0.85	40	class_4	4
focus_action	-	100	domain_ks_focus	1
focus_control	-	10	control_ks_focus	3

To determine the priority value for the design knowledge module "draw2d" (Equation 3.1 of Chapter 3), it is first necessary to find all the instances of the frame *'control_db'* with the "focus" as the decision_type slot value, that are less than the efficiency slot value 0.90. Once these are determined, the corresponding weight and weight factor are multiplied, and then the results are added to determine the priority value for the knowledge source.

Since 0.90 is greater than all the f_i , then the end result is:

priority for "draw2d" = (10x1) + (20x2) + (30x3) + (40x4) = 300

```
(define-instance  focus_1  (:print-name "focus_1"
                           :doc-string "..."
                           :is control_db)
 (competence      class_1)
 (criterion       (instance current_cycle is cycles  with value ?cycle)
                  (= ?cycle *number_of_ks*) )
 (decision_type   focus)
 (goal            (instance ?ksar is ksar  with efficiency ?effic)
                  (>= ?effic .25) )
 (status          active)
 (weight          10) )
```

Figure D.1: Example of the decision_type focus_1 instance

```
(define-frame integration_rule (:print-name "integration_rule"
                                :doc-string "Binding Integration_rule to weight factors"
                                :is data_structure)
 (list_of_weights              :multivalued t
                               :constraints (:lisp-type list) ) )              ;end define-frame
```

Figure D.2: Integration_rule frame

APPENDIX E

Rule Constructs

```
RULE:   DRAW2D_RULE1
                Doc String:         "action rules"
                Explanation String: "Draw2D is the drafting pre-processor,
                                     where the basic geometry is entered."
                Priority:           600             ;==> highest FIFO
                direction:          :FORWARD
                dependency:         nil
                Sponsor: ACTION_RULES               ;==> FIFO
IF      (INSTANCE STEP IS STEPS WITH MODULE II.7)
        (UNKNOWN (INSTANCE DRAW2D IS KSAR))
        (INSTANCE TO_DO_SET_RULE IS TO_DO_SET WITH CHOOSEN_ACTION KSAR_DRAW2D)
        (INSTANCE KSAR_DRAW2D IS KSAR WITH-UNKNOWN STATE FIRED)
        (INSTANCE ?KSAR_DRAW2D IS KSAR WITH KS_NAME DRAW2D)
        (INSTANCE DRAW2D IS KS)
THEN    (INSTANCE RULE_KSAR_CYCLES IS VARIABLE
                WITH NAME KSAR_DRAW2D
                WITH CY_VALUE (EVALUATE (SLOT-VALUE 'CURRENT_CYCLE 'VALUE)))
        (EVALUATE (SETF    (SLOT-VALUE 'OUT_WINDOW 'DISPLAY)
                           (FORMAT NIL "~&Running Draw2d")))
        (EVALUATE (FORMAT "FILE_OUT" "~&Running Draw2D !"))
        (EVALUATE (RUN_DRAW2D))
        (INSTANCE ?KSAR_DRAW2D IS KSAR
                WITH KS_NAME DRAW2D
                WITH STATE FIRED)
```

Figure E.1: Example of a production rule (draw2d)

The rule name, DRAW2D_RULE1 of Figure E.1, is obtained by concatenating the design module's name, DRAW2D, and the appropriate symbol of the rule's type, RULE1. The following keywords are used to assign attributes:

- ":doc-string" is used to document the purpose of the rule and its relationships to other rules in the design process. This string defaults to the rule's type.
- ":priority" is an integer between -1000 and +1000, inclusive, that indicates the priority

of the rule relative to other rules in the design. In the BBCAD design system, the rules default to a priority value, except for the action rules where the priorities are set by the user. They are used to give different ranks to different rules of the same design module.

- ":direction" indicates to the inference engine what kind of chaining to do with this rule. Its default value is defined as forward chaining.
- ":dependency" creates a justification structure for the new assertions a..d values it asserts, if the rule runs in a forward direction. If the rule has dependency nil, then it is a state rule, and is not part of the justification of values and assertions asserted in the consequent.
- ":sponsor" refers to the sponsor object to which this particular rule is assigned; its name is unique and depends on the rule's type. They are used to control the execution of the rules by having different levels of abstraction.
- ":explanation-string" consists of the explanations that are generated when asking why something is true. This string is user defined. It is put into the slot **explain_action** of the frame *'ks'*, defined in Chapter 3.

The antecedent part of this rule construct has a multitude of conditions. For it to fire, all the conditions set forth in the antecedent must be satisfied, thereby making the consequent true. The generated rule is shown in Figure E.1.

APPENDIX F
Implementation and User Interface

This appendix deals with more specific details of the implementation of the BBCAD blackboard system for engineering design. A description of the computer languages and of several alternative ways of communicating these languages that were used is presented. The interaction with the user and a front-end facilities are also a subject of this appendix.

F.1 Implementation

There are many possible implementation approaches available in the knowledge-based systems domain. Their choice depends on the basic approach taken. Aside from the approaches described in Appendix C of Chapter 2, other elements affect the choice of implementation, and these elements could prove to be even more important than those aforementioned. They include the type of design problem to be solved, the desired capabilities of the knowledge-based system, and the availability of the selected tools. Three approaches are used for the implementation of BBCAD: GoldWorks, for the implementation of frames, and rules; Golden Common Lisp (GCLISP) the platform upon which GoldWorks is based; and Sun Common Lisp.

F.2 Languages used

The following sections examine in some detail the languages used in implementing the design of BBCAD, and the approaches needed to construct the design modules:

F.2.1 GoldWorks

GoldWorks [GoldWorks 87] is a hybrid expert system development tool. It runs in a windows environment and provides a set of advanced integrated software development techniques which helped in the development of BBCAD. The information in GoldWorks is structured in a network representation called a *lattice*. The lattice is composed of *frames* and *instances*. Frames are the mechanism for structuring data and are used to describe a general class of objects. Slots are used to describe the attributes of those objects. Instances of a frame represent specific objects. They have the same slots as the associated frame, but these instances hold values in the slots. The GoldWorks lattice supports *inheritance*, so that a frame inherits all slots from its parent frames and ancestors, and passes on all its slots to its child frames. In addition, it gives the capability to define *local* slots for each frame to be able to define any attributes particular to the class of objects the frame represents.

F.2.2 Golden Common Lisp

The Golden Common LISP Developer (GCLISP Developer) [GoldWorks 87], serves as the programming language platform for GoldWorks. It can be used from within GoldWorks, or function independently as a Lisp programming facility [Winston and Horn 84]. It provides the power of a full Common Lisp [Steele 84]. Common Lisp is a function, or applicable language; that is, it exchanges data by using return values rather than temporary storage. It has two salient features: a list-based representation of data and an evaluator, or interpreter, that treats some lists as programs. Lisp functions are equivalent to subroutines or procedures in other languages. In contrast to most other languages, Lisp functions can create and return arbitrary data objects as their values. These data objects can then be passed as arguments to other functions.

F.2.3 Sun Common Lisp

Sun Common Lisp [SCL 88] is a complete implementation of the Common Lisp language. It includes all the Common Lisp functions, constants, variables, macros, and special forms. In addition, Sun Common Lisp provides many functions as extensions to Common Lisp and as enhancements to the user environment. It also includes an extensive set of advanced tools and features, such as the foreign function interface, and the function **run-program**. BBCAD uses the run-program to call executable programs from Lisp. The run-program runs UNIX programs from inside the Lisp environment, and these programs start up as separate processes. Thus, communication is limited to the standard input and standard output streams, and files.

F.3 Communications among Languages

The use of an appropriate language for each design tool during the design process of a problem is important. Since more than one language is used in the BBCAD design, it is necessary to establish communication among them. This section addresses the techniques and implementation issues that have been confronted in building BBCAD. It is divided in two sub-sections: the communication among symbolic languages, and the communication between symbolic and numerical languages.

F.3.1 Couplings among Symbolic Languages

The design modules and control knowledge sources (metaknowledge) are based on the GoldWorks lattice structure. They both use the frame as the basic knowledge definition. The slot values are lists of patterns and procedures which are defined in a programming language, and that language is Common Lisp. The Common Lisp language was useful for creating *Lisp function relations*, *daemons*, and *handlers*.

```
(define-instance  low_prio
                  (:print-name "low_prio"
                   :doc-string "Control Knowledge Source"
                   :is control_db)
 (decision_type   policy)
 (goal            (bind ?pairs (sort_priority_trig_invoc '< 'priority) )
                  (bind ?ksar1 (cadr ?pairs))
                  (bind ?prio1 (car ?pairs))
                  (variable-bound-p ?prio1)
                  (instance ?ksar2 is ksar    with status trig_and_invoc
                                              with priority ?prio2)
                  (equal-instances ?ksar2 ?ksar1)
                  (= ?prio2 ?prio1) )
 (status          active)
 (weight          5)  ) ; end define-instance
```

Figure F.1: A Policy decision type Control KS

Lisp function relations are used as predicates in rules. The lisp function relation "*(equal-instances ?ksar2 ?ksar1)*" of the slot **goal** in Figure F.1, verifies whether the two variables are the same instance.

Daemons perform various operational activities when slot values change, i.e. take care of procedural details and allow the overall program to be more clearly visualized. Daemons are lists of Lisp functions attached to the *facet* **:when-modified** of a slot (Figure 3.6 of Chapter 3). Whenever a value associated with a slot in an instance is changed (e.g., asserted, retracted, or modified), the when-modified daemon is only called once and the Lisp functions are evaluated.

$$(\textit{send-msg}\ \ \textit{instance}\ \ \textit{:handler}\ \ \{\textit{arg}\}^*) \qquad \textbf{(F.1)}$$

Handlers are used to attach Lisp functions to frames for use in *object-oriented* programming (see Chapter 3). The **send-msg** function, shown in statement **(F.1)**, is used to invoke the various defined handlers.

F.3.2 Couplings between Symbolic and Numerical Languages

Most simulation tools are programs written in compiled languages (e.g., mainly Fortran and C). It is essential to communicate these programs with the BBCAD, which is written in symbolic languages (e.g., Lisp and its dialects, and GoldWorks). Both capabilities are necessary for engineering design; symbolic languages are extensively used to represent heuristic expertise, while compiled languages are widely used for numerical processing.

The problem in coupling languages consists of interfacing different pieces code. The differences may consist of a combination of any of the following factors: parameter structures, concepts, operating system, implementation language, physical location of the programs, and data transfer among programs. Since generally the tools that are to be coupled are already written, this may present a restriction in the integration. It would be desirable to make a minimum set of changes, preferably none, to achieve the linkage. Sun Common Lisp allows compiled versions of programs in other languages to be treated as callable functions.

F.4 User Interface

The user interface is the communication medium between the user and the BBCAD blackboard system during the design process stages. An interactive interface linked to a knowledge representation and reasoning system will allow the user to cooperate with a knowledge-based system in a synergistic manner that utilises both human cognitive strengths and the abilities of computers to perform computation and display. The BBCAD user interface consists of a graphical environment which contains multiple scrollable pop-up windows, pull-down menus, dialog boxes and draggable icons. The user input is through both the keyboard and the mouse. Although BBCAD has the capability

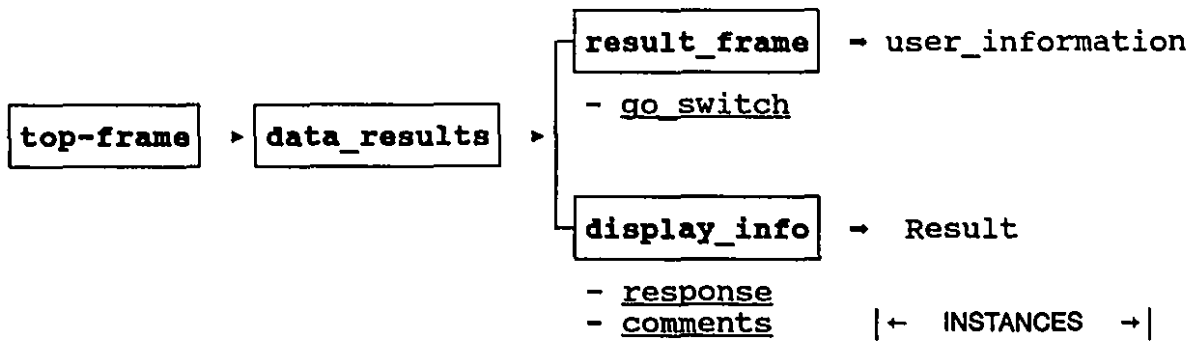

Figure F.2: Frames and instances of BBCAD user interface

of automating many phases of the design process, a user-friendly interface is very important in encouraging the user to explore new alternative solutions and in visualizing design layouts and results of analysis. The BBCAD system is implemented using the GoldWorks II shell on the SunOS 4/110 Release 4.1.1, running under X Windows environment. A graphic front-end interface is implemented using the GoldWorks graphics toolkit, to make it easier for users to load applications, to enter the information that the application requires, to view activities, to facilitate user interactions, to pass information messages between the design modules, and gives the power to represent events occurring in the application through various displays according to the changes taking place. Figure F.2 shows the top-level frame and instances of the user interface system.

F.4.1 Front-End Interface

A graphic object approach is used to define images, screen layouts, popup menus, and other elements of the BBCAD hybrid shell. The Graphics Toolkit of GoldWorks includes predefined (generic) frames that define the different kinds of graphic objects. Instances of the generic frames, i.e. frames with valid data, produce the graphic objects in the BBCAD blackboard. The Graphics Toolkit handlers [GoldWorks 87] are used to

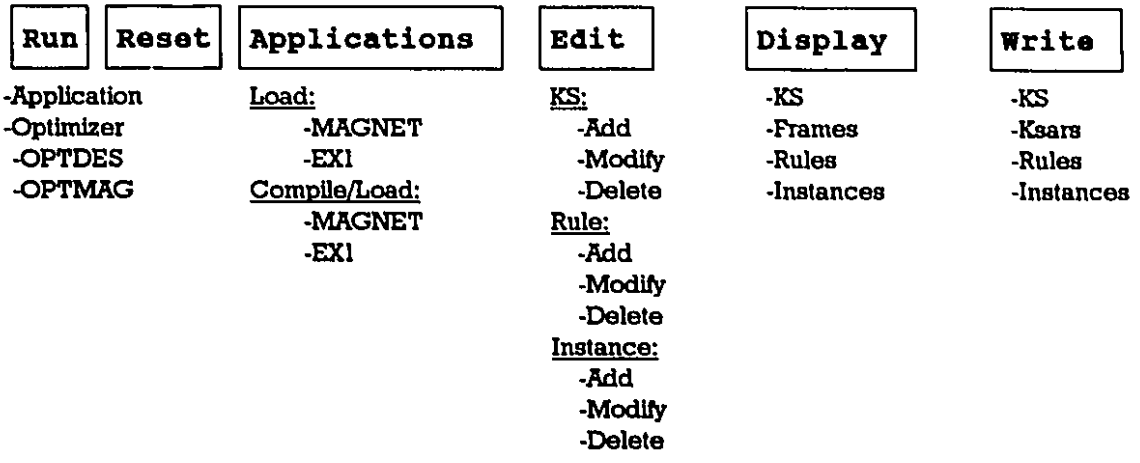

Figure F.3: BBCAD front-end pull-down menus

control the objects while the design process is running.

The interface has the capabilities of editing (e.g., adding, modifying, and deleting), viewing, saving, resetting, and clearing design modules, rules, instances, and frame structures. Figure F.3 identifies all the BBCAD front-end user interface pull-down menus, and the last section of this Appendix details the user interface frame structures of the BBCAD system.

F.4.2 User Interaction and Frame Structures

Interaction may occur directly between the user and events inside the BBCAD - design knowledge modules, and indirectly with the rest of the knowledge sources comprising the BBCAD shell. The interaction may occur in many directions: the user modifying the flow of control of the design process by means of commands, prompting for answers to questions, or popup menus from which to select choices; and the BBCAD system informing the user of important events or explaining decisions. The interaction also arises in managing the initialization of the design process, the information on internal

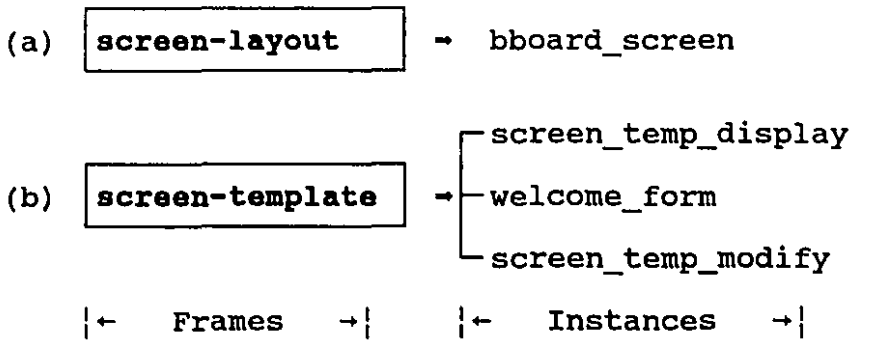

Figure F.4: Screen layout and Screen template

decisions of BBCAD, such as failure, success, and results of a design module. Design modules are able to provide explanations about their own structure and functionality. The validity of the answer is checked and verified in the frame structure of the device, i.e. through constraint facets.

The GoldWorks Graphics Toolkit which includes predefined frames that specify different kinds of graphic objects (e.g., screen layouts in Figure F.4 (a), images (e.g., screen templates in Figure F.4 (b)), canvases in Figure F.5, and popup menus in Figure F.6), is used to build BBCAD.

BBCAD utilises a *'screen layout'* window frame structure, shown in Figure F.4 (a), to display the various canvases items and to manage the user interface. The command line items of the screen layout are utilized to display the pull-down menus (Figure F.3), to call functions or change slot values when selected. Instances of the frame image *'screen-template'* shown in Figure F.4 (b), (e.g., "*Screen_temp_display*", "*screen_temp_modify*", and "*welcome_form*"), are used to arrange selectable items slot values and to display text.

To display an image on a canvas, i.e. window objects, the image must be *attached* to that canvas. Likewise, to display a canvas on a screen layout, the canvas must be attached to the screen layout. When a screen layout is opened or closed, all the canvases

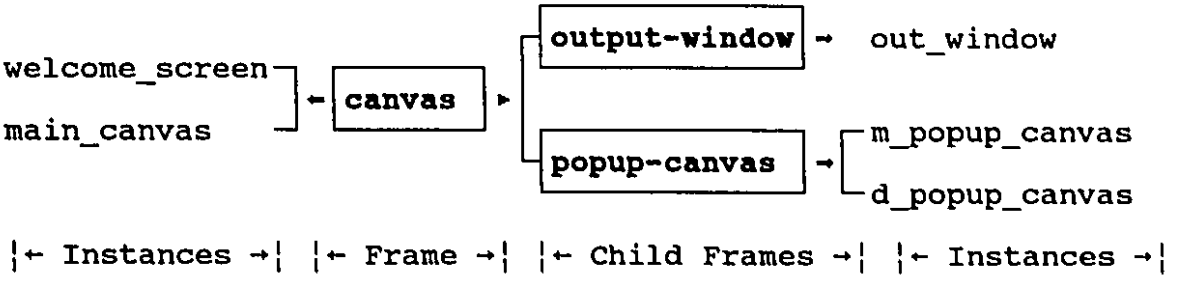

Figure F.5: Canvas frames and instances

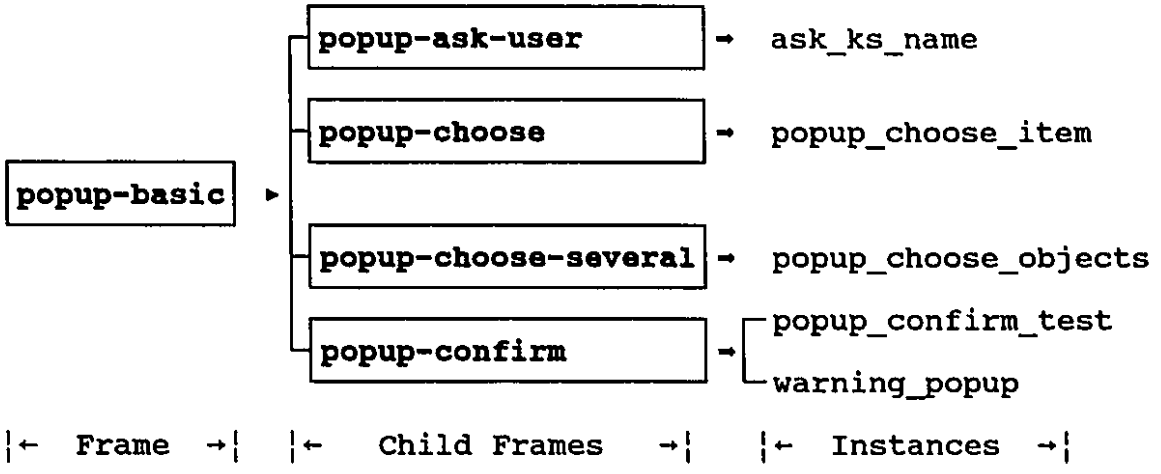

Figure F.6: Popup menus

attached to it are also opened or closed. Instances of the canvas frame, shown in Figure F.5, are used to display user information and activities in the design process.

The popup menus (Figure F.6) prompts the user for more information and instructions regarding the design knowledge modules. User inputs, i.e. values of instance "*ask_ks_name*" (Figure F.6), are checked and verified against the constraints requirement. BBCAD gives the user the choice of selecting one or more objects for processing, (e.g., "*popup_choose_item*" and "*popup_choose_objects*" shown in Figure F.6), and to confirm the answer, (e.g. *'popup-confirm'* frame of Figure F.6).

BBCAD uses GoldWorks' specially defined *handlers* as the mechanism to manipulate screen template, canvas, screen layout, and popup menu instances explained above. Handlers are the named procedure that is attached to a frame. They are used to send messages to instances of frames described. The message indicates the nature of operation to perform on the instance. BBCAD implements the GoldWorks **send-msg** function to send messages between the various deign modules. Its syntax is shown in statement **(F.1)**.

References

[Aiello 86] Aiello, N., "User-directed control of parallelism: The CAGE system," KSL 86-31, Knowledge Systems Laboratory, Stanford University, Stanford, CA, 1986.

[Arora and Baezinger 86] Arora, J.S. and Baezinger, G., "Uses of Artificial Intelligence in Design Optimization," *Computer Methods in Applied Mechanics and Engineering*, Volume 54, No 54, pp. 303-323, March, 1986.

[Balling et al. 85] Balling, J.R., Parkinson, A., and Free, J., "OPTDES user's manual, version 3.0," Brigham Young University, Provo, UT, USA, 1985.

[Bañares-Alcántara et al. 87] Bañares-Alcántara, R., Westberger, A.W., Ko, E.I. and Rychener, M.D., "DECADE - A Hybrid Expert System for Catalyst Selection -I. Expert System Consideration," *Computer Chemical Engineering*, Vol. 11, No. 3, pp. 265-277, 1987.

[Bañares-Alcántara et al. 88(a)] Bañares-Alcántara, R., Westberger, A.W., Ko, E.I. and Rychener, M.D., "DECADE - A Hybrid Expert System for Catalyst Selection -II. Final Architecture and Results," *Computer Chemical Engineering*, Vol. 12, No. 9/10, pp. 923-938, 1988.

[Bañares-Alcántara et al. 88(b)] Bañares-Alcántara, R., Westberger, A.W., Ko, E.I. and Rychener, M.D., "The DECADE Catalyst Selection System," in [Rychener 88], pp. 53-91.

[Barr and Feigenbaum 81,82] Barr, A. and Feigenbaum, E.A., (Eds.) *The Handbook of Artificial Intelligence*, Volumes I and II, Heuris Tech Press, Stanford, California, 1981, 1982.

[Bisiani et al. 87] Bisiani, R., Alleva, F., Forin, A., Lerner, R., and Bauer, M., "The architecture of AGORA environment," in [Huhns 87], chapter 4, pp. 99-117, 1987.

[Bobrow and Winograd 77] Bobrow, D.G. and Winograd, T, "An Overview of KRL, a Knowledge Representation Language," *Cognitive Science*, Vol. 1, No. 1, pp. 3-46, 1977.

[Brachman and Levesque 85] Brachman, R.J. and Levesque, H.J., (Eds.) *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Los Altos, California, 1985.

[Brett 90] Brett, C.S., "An Interval Mathematics Package for Computer-Aided Design in Electromagnetics," *M. Eng. Thesis*, McGill University, Montreal, Quebec, Canada, 1990.

[Brett et al. 90] Brett, C.S., Saldanha, C.M., and Lowther, D.A., "An Interval Mathematics for Knowledge-Based Computer-Aided Design in Magnetics," *IEEE Transactions on Magnetics*, Vol. 26, No. 2, pp. 803-806, March, 1990.

[Brown and Buckman 82] Brown, H. and Buckman, J., "Final Report on HANNIBAL," Technical Report, ESL, Inc., 1982.

[Buchanan and Shortliffe 84] Buchanan, B.G. and Shortliffe, E.H., *Rule-Based Expert Systems*, Addison-Wesley Publishing Co., New York, 1984.

[Carter and MacCallum 91] Carter, I.M. and MacCallum, K.J., "A software Architecture for Design Co-ordination," in [Gero 91], pp.859-881, 1991.

[Cohen and Feigenbaum 83] Cohen, P.R. and Feigenbaum, E.A., *The Handbook of Artificial Intelligence*, Volume II, Heuris Tech Press, Stanford, California, 1983.

[Corkill and Lesser 83] Corkill, D.D. and Lesser, V.R., "The Use of Meta-Level Control for Coordination in a Distributed Problem Solving Network," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, Vol. 2, pp. 748-756, Karlsruhe, West Germany, August 8-12, 1983.

[Corkill et al. 86] Corkill, D.D., Gallagher, K.Q. and Murray, K.E., "GBB: A Generic Blackboard Development System," *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 2, pp. 1008-1014, Philadelphia, Pennsylvania, August, 1986. (Also in [Engelmore and Morgan 88], pp. 503-517, 1988.)

[Corkill et al. 87] Corkill, D.D., Gallagher, K.Q. and Johnson, P.M., "Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures," *Proceedings of the National Conference on Artificial Intelligence (AAAI-87)*, Vol. 1, pp. 18-23, Seattle, Waashington, July, 1987.

[Coyne et al. 90] Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M. and Gero, J.S., *Knowledge-Based Design Systems*, Addison-Wesley Publishing Company, Inc., New York, 1990.

[Clocksin and Mellish 85] Clocksin, W.F. and Mellish, C.S., *Programming in Prolog*, Second Edition, Springer-Verlag, Inc., Berlin, New York, 1985

[Davis et al. 77] Davis, R., Buchanan, B.J., and Shortliff, E., "Production Rules as a Representation for a Knowledge-Based Consultation Program," *Artificial Intelligence*, Vol. 8, pp. 15-45, 1977.

[DeMori and Prager 90] DeMori, R. and Prager, R., "Reasoning with Qualitative Linear Models," *Technical Report TR-SOCS-90.9*, School of Computer Science, McGill University, Montreal, Quebec, Canada, April, 1990.

[Dym 85] Dym, C.L., *Expert Systems: New Approaches to Computer-Aided Engineering*, *Engineering with Computers*, Vol. 1, No.1, pp. 9-25, 1985.

[Dym and Levitt 91] Dym, C.L., and Levitt, R.E., *Knowledge-Based Systems in Engineering*, McGraw-Hill Book Company, New York, 1991.

[Enderle et al. 84] Enderle, G., Kansy, K., and Pfaff, G., *GKS-The Graphics Kernel Standard*, Heidelberg: Springer-Velag, 1984.

[Engelmore and Terry 79] Engelmore, R. and Terrey, A., "Structure and Function of the CRYSALIS System," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence (IJCAI-79)*, Vol. 1, pp. 250-256, Tokyo, Japan, August 20-23, 1979.

[Engelmore and Morgan 88] Engelmore, R.S. and Morgan, T., (Eds.) *Blackboard Systems*, Addison-Wesley, Publisher Workingham, England, 1988.

[Erman and Lesser 75] Erman, L.D. and Lesser, V.R., "A Multi-Level Organization for Problem Sloving Using Many Diverse, Cooperating Sources of Knowledge," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, Vol. 2, pp. 483-490, Tbilisi, Georgia, USSR, September 3-8, 1975.

[Erman et al. 80] Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Reddy, D.R., "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, ACM, Vol. 12, No. 2, pp. 213-253, June, 1980.

[Erman et al. 81] Erman, L.D., London, P.E. and Fickas, S.F., "The Design and an Example Use of HEARSAY-III," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, Vol. 1, pp. 409-415, Vancouver, British Columbia, Canada, August 24-28, 1981.

[Feigenbaum 77] Feigenbaum, E.A., "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Vol. 2, pp. 1014-1029, MIT, Cambridge, Massachussetts, August 22-25, 1977.

[Fennell and Lesser 77] Fennell, R.D. and Lesser, V.R., "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay II, "*IEEE Transactions on Computers*, Vol. C-26, No. 2, pp. 98-111, February, 1977.

[Fikes and Kehler 85] Fikes, R. and Kehler, T., "The Role of Frame-Based Representation in Reasoning," *Communications of the ACM*, Vol. 28, No. 9, pp. 904-920, September, 1985.

[Fitzgerald et al. 83] Fitzgerald, A.E., Kingsley, C. Jr., and Umans, S.D., *Electric Machinery*, Fourth Edition, McGraw-Hill, Inc., New York, 1983.

[Forgy 81] Forgy, C.L., *OPS5 Users's Manual*, Department of Computer Science, Carnagie-Mellon University, Pittsburgh, PA 15213, 1981.

[Genesereth and Nilsson 87] Genesereth, M.R. and Nilsson, N.J., (Eds.) *Logical Foundations of Artificial Intelligence*, Morgan Kaufman Publishers, Inc., Los Altos, CA., 1987.

[Gentilhomme 91] Gentilhomme, A., "C.O.C.A.S.E: Un Système d'Aide à la Conception des Contacteurs," *Thesis du Doctorat*, Laboratoire d'Electrotechnique de Grenoble, Institut National Polytechnique de Grenoble, France, 1991.

[Gero 88] Gero, J.S., (Ed.) *Artificial Intelligence in Engineering: Design*, Elsevier/CMP, New York 1988.

[Gero 91] Gero, J.S., (Ed.) *Artificial Intelligence in Design '91*, Butterworth-Heinemann, Ltd., Oxford, London, 1991.

[Gero and Coyne 85] Gero, J.S. and Coyne, R.D. "Logic Programming as a means of Representing Semantics in Design Languages," *Environment and Planning B*, volume 12, pp. 351-369, 1985.

[Gero et al. 88] Gero, J.S., Maher, M.L. and Zhang, W., "Chuncking Structural Design Knowledge as Prototypes," in [Gero 88], pp.3-21, 1988.

[Gibson 68] Gibson, J.E., *Introduction to Engineering Design*, Holt, Rinehat and Winston, New York, 1968.

[Gill and Murray 74] Gill, P.E. and Murray, W., (Eds.) *Numerical Methods for Constrained Optimization*, Academic Press Inc., New York, 1974.

[Goldstein and Roberts 77] Goldstein, I.P. and Roberts, R.B., "NUDGE: A Knowledge-Based Scheduling Program," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Vol. 1, pp. 67-76, MIT, Cambridge, Massachussetts, August 22-25, 1977.

[GoldWorks 87] GoldWorks II, "Reference Manual, Graphics Toolkit, and GCLISP Developer Reference Manual," Gold Hill Computers, Inc., Cambridge, MA., 1987.

[Gregory 66] Gregory, S.A., (Ed.) *The Design Method*, Butterwoth & Co. (publishers) Ltd., London, 1966.

[Hayes 77] Hayes, P.J., "On Semantic Nets, Frames and Associations," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Vol. 1, pp. 99-107, MIT, Cambridge, Massachussetts, August 22-25, 1977.

[Hayes-Roth 85(a)] Hayes-Roth, B., "A Blackboard Architecture for Control," *Artificial Intelligence: an International Journal*, Vol. 26, No. 3, pp. 251-321, March, 1985.

[Hayes-Roth 85(b)] Hayes-Roth, F., "Rule-Based Systems," *Communications of the ACM*, Vol. 28, No. 9, pp. 921-932, September, 1985.

[Hayes-Roth et al. 83] Hayes-Roth, F., Waterman, D.A. and Lenat, D.B., *Building Expert Systems*, Addison-Wesley Publishing Co., New York, 1983.

[Hayes-Roth et al. 86] Hayes-Roth, B., Buchanan, B.J., Lichtarge, O., Hewett, M., Altman, R., Brinkley, J., Cornelius, C., Duncan, B. and Jardetzky, O., "PRETEAN: Deriving Protein Structure from Constraints," *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Vol. 1, pp. 904-909, Philadelphia, Pennsylvania, August, 1986. (Also in [engelman and Morgan 88], pp. 417-431, 1988.)

[Hewitt 69] Hewitt, C., "PLANNER: A Language for Proving Theorems in Robots," *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, Vol. 1, Washington, D.C., May, 1969.

[Jagannathan et al. 89] Jagannathan, V., Dodhiawala, R. and Baum, L.S., (Eds.) *Blackboard Architectures and Applications*, Academic Press, Inc., New York, 1989.

[Jensen and Wirth 74] Jensen, K. and Wirth, N. *PASCAL: User Manual and Report*, Second Edition, Springer-Verlag, Inc., New York, 1974.

[Kasper 92] Kasper, M., "Shape Optimization by Evolution Strategy," *IEEE Transactions on Magnetics*, Vol. 28, No. 2, pp. 1556-1560, March, 1992.

[Katzan 78] Katzan, H. *FORTRAN 77*, New York: Van Nastrand Reinhold Co., 1978

[Kernighan and Ritchie 78] Kernighan, B.W. and Ritchie, D.M., *The C Programming Language*, Englewood Cliffs, New Jersey: Prentice-Hall, 1978

[Knowledge Craft 85] Knowledge Craft, *CRL Technical Manual*, version 3.0, Carnagie Group Inc., Pittsburgh, PA 15219, 1985.

[Kowalik and Kitzmiller 88] Kowalik, J.S. and Kitzmiller, C.T., (Eds.) *Coupling Symbolic and Numerical Computing in Expert Systems, II*, Elsevier Science Publishers B.V., North-Holland, 1988.

[Laâsri and Maître 89] Laâsri, H. and Maître, B., "Flexibility and Efficiency in Blackboards Systems: Studies and Achievements in ATOME," in [Jagannathan et al. 89], pp. 309-322, 1989.

[Lesser at al. 75] Lesser, V.R., Fennell, R.D., Erman, L.D. and Reddy, D.Raj, "Organization of the Hearsay II Speech Understanding System, "*IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-23, No. 1, pp. 11-24, February, 1981.

[Lesser and Erman 77] Lesser, V.R. and Erman, L.D., "A Retrospective View of the Hearsay-II Architecture," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Vol. 2, pp. 790-800, Cambridge, Massachussetts, August 22-25, 1977.

[Lesser and Corkill 83] Lesser, V.R. and Corkill, D.D., "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks," *AI Magazine*, Vol. 4, No. 3, pp. 15-33, Fall, 1983. (Also in [Engelmore and Morgan 88], pp. 353-386)

[Lowther 89] Lowther, D.A., "The Need for Multiple Paradigms in the Development of a Knowledge-Based Electromagnetic Device Design System,", *International Working Conference on Expert Systems in Electrical and Power Systems*, paper 1.2, Avignon, France, 1989.

[Lowther et al. 85] Lowther, D.A., Saldanha, C.M., and Choy, G., "The Application of Expert Systems to CAD in Electromagnetics," *IEEE Transactions on Magnetics*, Vol. MAG-21, No. 6, pp. 2559-2562, November, 1985.

[Lowther and Saldanha 86] Lowther, D.A. and Saldanha, C.M., "A Frame-Based System for the Design of Electromagnetic Devices," *IEEE Transactions on Magnetics*, Vol. MAG-22, No. 5, pp. 814-816, September, 1986.

[Lowther and Silvester 86] Lowther, D.A. and Silvester, P.P., *Computer Aided Design in Magnetics*, Springer-Verlag Inc., New York, 1986.

[MagNet 85] "The MagNet User's Manual," Infolytica Corporation, Montreal, Quebec, Canada, 1985.

[Maher 88] Maher, M.L., "HI-RISE: An Expert System for Preliminary Sructural Design," in [Rychener 88], pp.37-52, 1988.

[Minsky 75] Minsky, M., "A Framework for Representing Knowledge", in [Winston 75].

[Mittal and Araya 86] Mittal, S. and Araya, A., "A Knowledge-Based Framework for Design," *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, Vol. 2, pp. 856-865, Philadelphia, Pennsylvania, August, 1986.

[Newell 62] Newell, A., "Some Problems of basic organization in Problem-Solving Programs," in [Yovits et al. 62], pp.393-423.

[Newell 69] Newell, A., "Heuristic Programming: Ill-Structured Problems," *Progress in Operations Research*, New York: John Wiley, III, pp.360-414, 1969.

[Newell and Simon 72] Newell, A. and Simon, H. *Human Problem Solving*, Englewood Cliffs, Prentice-Hall, New Jersey, 1972.

[Nicklaus et al. 88] Nicklaus, D.J., Overton, K.S., Tong, S.S. and Russo, C.J., "Knowledge Representation and Technique for Engineering Design Automation," in [Kowalik and Kitzmiller 88], pp. 67-76, 1988.

[Nii and Aiello 79] Nii, H.P. and Aeillo, N., "AGE (Attempt to Generalize): A Knowledge-Based Program for Building Knowledge-Based Programs," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence (IJCAI-79)*, Vol. 2, pp. 645-655, Tokyo, Japan, August 20-23, 1979.

[Nii et al. 82] Nii, H.P., Feigenbaum, E.A., Anton, J.J. ans Rockmore, A.J., "Signal-to-Symbol Transformation: HASP/SIAP Case Study," *AI Magazine*, Vol. 3, pp.23-35, 1982 and in [Engelmore and Morgan 88], pp. 135-157, 1988.

[Nii 86(a)] Nii, H.P., "Blackboard Systems," *AI Magazine*, Vol. 7, No. 3, pp.38-53, and Vol. 7, No. 4, pp. 82-106, Summer, 1986.

[Nii 86(b)] Nii, H.P., "CAGE and POLIGIN: Two frameworks for blackboards based concurrent problem solving," KSL 86-41, Knowledge Systems Laboratory, Stanford University, Stanford, CA, 1986.

[Nilsson 80] Nilsson, N.J., *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, California, 1980.

[Peterson and Silberschatz 85] Peterson, J.L. and Silberschatz, A., *Operating Systems Concepts*, Second Edition, Addison-Wesley Publishing Co., New York, 1985.

[Prager et al. 89] Prager, R., Belanger, P. and DeMori, R., "A Knowledge-Based System for Troubleshooting Real-Time Models," in Widman, Loparo, and Nielson, (Eds.) *Artificial Intelligence, Simulation, and Modelling*, pp. 511-543, John Wiley & sons, Inc., 1989.

[Preis and Ziegler 90] Preis, K., and Ziegler, A., "Optimal Design of Electromagnetic Devices with Evolution Strategies," *COMPEL-The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, Vol. 9, Supplement A, pp. 119-122, 1990.

[Preiss 80] Preiss, K., "Design Frame Model for the Engineering Design Process," *Design Studies*, Vol. 1, No. 4, pp. 231-243, April, 1980.

[Press et al. 88] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recipes in C: The Art of Scientific Computing*, Second Edition, Cambridge University Press, Cambridge, UK, 1988.

[Quillian 68] Quillian, R., "Semantic Memory," in M. Minsky, (Ed.) *Semantic Information Processing*, The MIT Press, Cambridge, MA., 1968.

[Rich 83] Rich, E., *Artificial Intelligence*, McGraw-Hill Book Company, New York, 1983.

[Rychener 88] Rychener, M.D. (Ed.) *Expert Systems for Engineering Design*, Academic Press, Inc., New York, 1988.

[Rychener et al. 84] Rychener, M.D., Bañares-Alcántra, R. and Subrahmanian, E., "A Rule-Based Blackboard Kernel System: Some Principles in Design," *IEEE Workshop on Principles of Knowledge-Based Systems*, pp. 59-64, Denver, Colorado, December, 1984.

[Rychener et al. 86] Rychener, M.D., Farinacci, M.L., Hulthage, I. and Fox, M.S., "Integration of Multiple Knowledge Sources in Aladin, an Alloy Design System," *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, Vol. 2, pp. 878-882, Philadelphia, Pennsylvania, August, 1986.

[Saldanha 88] Saldanha, C.M., "An Algebraic Constraint System for CAD in Magnetics," *M. Eng. Thesis*, McGill University, Montreal, Quebec, Canada, 1988.

[Saldanha and Lowther 86] Saldanha, C.M. and Lowther, D.A., "Automating the Design Process for Electromagnetic Devices," *IEE Computer-Aided Engineering Journal*, Vol. 3, No. 5, pp. 173-179, October, 1986.

[Saldanha and Lowther 87] Saldanha, C.M. and Lowther, D.A., "Device Modelling in an Electromagnetic Device System," *IEEE Transactions on Magnetics*, Vol. MAG-23, No. 5, pp. 2644-2646, September, 1987.

[Saldanha and Lowther 88] Saldanha, C.M. and Lowther, D.A., "Knowledge-Based Computation of Electromagnetic Device Parameters," *IEEE Transactions on Magnetics*, Vol. 24, No. 1, pp. 334-337, January, 1988.

[Sassine and Lowther 91] Sassine, R.M. and Lowther, D.A., "Integrating Computer Based Electromagnetic Device Design Tools to Solve Coupled Problems," *COMPUMAG - 8th Conference on the Computation of Electromagnetic Fields*, Vol. 1, pp. 279-282, Sorrento, Italy, July 7-11, 1991.

[Simon 69] Simon, H.A. *The Sciences of the Artificial*, MIT Press, Cambridge, MA, 1969.

[Smith and Davis 81] Smith, R.G. and Davis, R., "Frameworks for Cooperation in Distributed Problem Solving," *IEEE Transaction on Systems, Man and Cybernatics*, SMC-11, No. 1, pp. 61-70, January, 1981.

[Smith et al. 83] Smith, B.M., Brauner, K.M., Kennicot, P.R., Liewald, M., and Wellington, J., "Initial Graphics Exchange Specification (IGES) Version 2.0," NBSIR 82-2631(AF), National Bureau of Standards, US Department of Commerce, Washington, DC, USA, February, 1983.

[Sriram et al. 89] Sriram, D., Stephanopoulos, G., Logcher, R., Gossard, D., Groleau, N., Serrano, D. and Navinchandra, D., "Knowledge-Based System Applications in Engineering Design: Research at MIT," *AI Magazine*, Vol. 10, No. 3, pp. 79-96, Fall, 1989.

[Steele 84] Steele, G.L. Jr., (Ed.) *Common Lisp: The Language*, Digital Press, Burlington, Massachussets, USA, 1984.

[Stefic and Bobrow 86] Stefic, M., and Bobrow, D.G., "Object-Oriented Programming - Themes and Variations," *AI Magazine*, Vol. 6, No. 4, pp. 40-62, 1986.

[Stroustrup 86] Stroustrup, B., *The C++ Programming Language*, Reading, Massachusetts: Addison-Wesley Publishing Co., 1986.

[SCL 88] Sun Common Lisp 3.0, "User's Guide, and Reference Guide," Sun Microsystems,Inc. and Lucid, Inc., USA, 1988.

[Talukdar and Cardozo 88] Talukdar, S.N. and Cardozo, E., "Building Large-Scale Software Organizations," in [Rychener 88], pp. 241-256, 1988.

[Terry 83] Terry, A., "The CRYSALIS Project: Hierarchical Control of Production Systems," Technical Report HPP-83-19, Stanford University. Also in [Engelmore and Morgan 88], pp. 159-188.

[Vanderplaats 84] Vanderplaats, G.N., *Numerical Optimization Techniques for Engineering Design: with Applications*, McGraw-Hill, New York, 1984.

[VanMelle et al. 81] Van Melle, W., Shortliffe, E.H. and Buchanan, B.G. "EMYCIN: A Domain-Independent System that Aids in Constructing Knowledge-based Consultation Programs," *Machine Intelligence*, Vol. 9, No. 3, 1981.

[Velthuijsen and Braspenning 91] Velthuijsen, H. and Braspenning, P.J., "Results of Formalising the Blackboard Architecture," *Proceedings of the Fifth Annual AAAI Workshop on Blackboard Systems*, Anaheim, California, July 14, 1991.

[Winston 84] Winston, P.H., *Artificial Intelligence*, Addison-Wesley Publishing Co., New York, 1984.

[Winston and Horn 84] Winston, P.H. and Horn, B.K.P., *LISP*, Addison-Wesley Publishing Co., New York, 1984.

[Wright and Fox 83] Wright, J.M. and Fox, M.S., *SRL1.5 User Manual*, Intelligent Systems Laboratory, The Robotics Institute, Carnagie-Mellon University, Pittsburgh, PA 15213, 1983.

[Yang et al. 85] Yang, J.D., Huhns, M.N. and Stephens, L.M., "An Architecture for Control and Communications in Distributed Artificial Intelligence Systems," *IEEE Transactions on Systems, Man, and Cybernatics*, Vol. SMC-15, No. 3, pp. 316-326, May/June, 1985.

[Yovits et al. 62] Yovits, M.C., Jacobi, G.T. and Goldstein, G.D. (Eds.), *Conference on Slef-Organizing Systems*, Spartan-Books, Washington, D.C., 1962.

[Zhu 91] Zhu, Z., "An Intelligent CAD System for DC Machines," *M. Eng. Thesis*, McGill University, Montreal, Quebec, Canada, 1991.