PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

EXPERT SYSTEM ARCHITECTURE FOR DESIGN TASKS

Yasuo Nagai, Satoshi Terasaki, Takanori Yokoyama, and Hirokazu Taki

Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo, 108, Japan
csnet: nagai%icot.jp@relay.cs.net
uucp: {kddlab,mit-eddie,ukc,enea,inria}!icot!nagai

ABSTRACT

Research is being conducted to develop expert systems that solve design problems. The strategy of this research is to research and develop basic software, with emphasis on user needs and application systems. The goals of this research are:

- To clarify the architecture of the expert system for various designs by introducing constraint-based problem solving.
- To propose primitive tasks to realize the architecture.
- To provide an expert system building tool based on the above considerations.

This paper focuses on and specifies constraint-based problem solving in order to consider expert system architecture, including the modeling facility of the design object.

In this case, though the design knowledge must be supported and handled, current expert systems and tools do not always do so, especially for the design object. Therefore, the handling of design object and problem-solving mechanism are considered. Design object representation system, called `FREEDOM`, is explained. Moreover, a detailed architecture for an expert system building tool, including the knowledge compilation technique for the efficient problem solving, is described. Finally, current state of a design support system, called `MECHANICOT`, is explained as a practical example of this building tool.

1 INTRODUCTION

Design systems, such as design automation (DA) systems and computer aided design (CAD) systems, have been developed and used in various design fields. More and more of these systems have incorporated expert systems and knowledge-based systems for design problems.

Design systems can be classified as automated design systems or interactive design systems; it depends on whether there is interaction during design [Eastman 81]. Automated design systems require the determination of definition of the design process and the decision-making sequences. Automated design systems do not usually interact with the designer, and they demand vast amounts of data and computation time. Interactive systems are very flexible and open-ended to the designer, who may input multiple descriptions as input specification. They have a decentralized control structure for the design process, while the structure of the non-predetermined decision-making sequence and its control depend on the designer. Most CAD systems are interactive and are applied only to parts of the design process. Furthermore, the structure of the process model and the decision-making sequence for design systems depend on the design area.

Expert systems have two major applications: diagnosis problems and design problems. Expert systems for design problems are being developed and evaluated for various application fields, such as VLSI design [Kowalski and Thomas 83] [Subrahmanyam 86]; mechanical design [Brown and Chandrasekaran 86] [Dixon and Simmons 84] [Mittal *et al.* 86]; configuration [McDermott 82]; and process planning; [Descotte and Latombe 85] [Eliyahu *et al.* 87].

The architecture of expert systems for design problems is not yet as explicit as that for diagnosis problems. Design problems can be regarded as complicated problems that contain a synthesis task in addition to analysis and simulation tasks [Medland 86]. The descriptions of objects to be designed are changed and determined dynamically because of the trial and error nature of the synthesis task, and the results from the synthesis task are analyzed and evaluated in the analysis task. In other words, first the models for design objects are selected, modified and determined so that the design specification can be satisfied, then the synthesis and analysis tasks are performed repeatedly according to the model for detailed description of design objects.

However, this modeling facility is not provided explicitly in existing expert systems for design problems,

and the relation between the expert system architecture and this modeling function has not been considered. Most expert systems for design problems are formalized as systems that generate detailed descriptions of the design objects as the solution by combining known components, assemblies or parts, then refining them so that the design specification can be satisfied. Moreover, it seems that such systems lack recognition of the modeling concept from the viewpoint of design. Therefore, expert systems for design problems require a sophisticated architecture that considers the design process model, including planning and modeling facilities.

Section 2 is an overview of the design systems in relation to the design process model; it clarifies the relation of the expert system to the design system. When the design process models of the expert system are applied to the design problems for automated design or interactive design, we need to regard these design problems as a well-defined and well-structured problems, and they are defined as routine design. This routine design is classified in more detail, and the design process model for each detailed design, its fundamental design task and the relation between the classified designs and practical design fields are specified.

Section 3 describes the architecture and necessary problem-solving mechanism of the expert system for routine design. First, researches on architectures consisting of primitive tasks for routine design, called *design generic tasks* [Chandrasekaran 86] [Brown and Chandrasekaran 86] [Mittal *et al.* 86] are described as previous works. Next, the architecture and necessary problem-solving mechanism are described considering *generic task* research.

In our research approach, technical issues of considering expert system architecture for *design generic tasks*, especially design knowledge and problem-solving mechanisms for realization of these tasks are investigated.

Section 4 considers design object model. To make an efficient mechanism to solve design problems, we must consider a way to model the design object, which uses knowledge about the design object effectively. At present, frame-based and object-oriented representations are popular knowledge representation for this purpose. In comparison with them, it is not sufficient that the modeling method of the design object handles the dynamic management of constraint. The modeling system of the design object is described according to the above considerations.

Section 5 defines the application mechanisms for constraint representation in routine design expert systems as constraint-based problem solving, and describes them. For this purpose, constraint representation is classified as the general constraints that are needed for routine design or domain-specific constraints; application mechanisms are considered according to these kinds of constraint. Modeling of the design object corresponds to the formalization of various descriptions for design knowledge and to the method of handling design knowledge by trial and error.

These constraints are generated, derived, modified, or deleted from modeling during the design process. The constraint-based problem-solving mechanism is described according to the constraint classifications by matching the design process model for a routine design with the design system.

Section 6 describes the architecture for expert systems and expert system building tools based on this architecture, introducing the modeling concept of the design object and focusing on constraint representation. The design support system is proposed as the specialized solution of the expert system building tool. It introduces the design plan generation facility using constraint analysis method, according to the above architecture.

Section 7 explains in detail the architecture and current state of a support environment called `MECHANICOT` in which the designer can construct design systems easily. This system focuses on mechanical design, using the design of a main spindle head for a lathe as an example.

2 DESIGN SYSTEM

2.1 Necessary Functions

A design system executes the following design activity in an automated or interactive manner. Design is a creative and intelligent human activity that transforms the requirements represented by formalized languages, symbols, and figures, into physical objects. In other words, given requirements, design creates the structure and the form of the design object according to the design object model that satisfies these requirements. Design can also be considered as synthesis and analysis tasks. It is necessary to consider the correspondence between the analysis task and the model of the design object.

Design consists of three phases:

- Conceptual design
- Fundamental design
- Detailed design

The design in each phase creates the model and executes the analysis and evaluation, and modification in the indeterministic manner.

There are various design systems that cover various design levels such as conceptual design, fundamental design and detailed design. Design systems must execute the design activity at each level.

Conceptual design enumerates the requirements for the realized design objects. It focuses on and formalizes the design problem, including the specification definition. This phase begins a creative task that formalizes the concepts and ideas applied to the designed object. Except for parts of the stylized design, conceptual design is not formalized explicitly and is executed according to skills such as the designers' ideas and experience. Designers mull and judge the conceptual models and choose among them. In fact, there are design systems for conceptual design.

Fundamental design analyzes, evaluates, and modifies the models from various viewpoints according to the result of the conceptual design. It does this in order to refine the design objects and to select some models. The methods of analysis and evaluation for each model are predetermined, but when these methods are not established, they are determined by experiment and simulation.

Detailed design refines the components of the design object and relations between components using the selected model, according to the result of the fundamental design. The optimization and evaluation of the result of the design are executed according to this model.

Most design systems are intended for fundamental design and detailed design.

Next, necessary functions for the design systems, considering mechanical and VLSI designs, are described. In mechanical engineering, there are many cases in which design systems such as DA systems and CAD systems are provided for each design object. In fact, the individuality that the design object possesses makes it difficult to abstract, arrange, and use the design systems as the design environment, because the corresponding model and analysis method for this object usually change when the structure of the design object changes. In other words, of all the synthesis and analysis tasks used in DA systems or CAD systems, the analysis tasks for the performance and behavior (function) prediction and evaluation are especially determined and provided based on the model of the design object. This is caused by the fact that a change in the structure of the design object, such as its geometrical characteristics, may result in a change in its functions.

The models of VLSI design for the analysis of the performance and behavior (function) and evaluation are fixed and formalized as the design methodology, because structural changes have little effect on the function and behavior. The hierarchical structure of the VLSI design, especially the nested structure with the function, can be represented by combining the lower level functions so that interactions between sub-structures of the design can be minimized.

2.2 Routine Design

2.2.1 Definition of Routine Design

Routine design determines the structure of the design object by combining predetermined components, given the model of the basic components and structures of the design object and the methods by which they are to be analyzed and evaluated. In routine design, the two levels of design activity, functional level and physical level, are executed. First, functional level design is executed. It includes the functional decomposition of the specification into a functional description such as functional specification, unit, block, or component. Next, physical level design is executed. It contains the decomposition of the functional description into a physical description of the components for implementation. Finally, the physical description can be obtained as the solution of the design.

Routine design satisfies the following items.

1) The functional or behavioral description can be formalized explicitly as the design requirement or specification.

2) The design plans at each phase can be formalized. They consist of a problem decomposition method such as functional or structural decomposition of the specification or problem, a refinement method, an analysis method, and an evaluation method.

Routine design can formalize a design that has been realized as a well-defined and well-structured problem. It is a design that uses the same expertise and problem-solving method in the previous design. The design specification of routine design is well understood. The problem can be solved using the standard problem solving method, and DA systems and semi-DA systems are realized instances of routine design.

Modification or edit design can also be interpreted as typical routine design. They improve the results of previous designs (the explicit specification of the design parameters and their dimensions).

In contrast with these DA systems and semi-DA systems, most CAD systems that conduct drawing, mass property determination, finite-element analysis, or dynamics analysis, provide only basic assistance facilities for the designer. They are not suitable for routine design, which decides whether the design result satisfies the design requirements.

2.2.2 Design Process Model and Fundamental Design Task

This section describes the fundamental tasks in the design process model.

Considering the design process of routine design, the following items are required [Rinderle 87].

1) Interaction between conceptual design, fundamental design, and detailed design

2) Design objects can be structured with minimal or no interaction between conceptual design, fundamental design, and detailed design.

3) The design process can be structured so that the design task at each phase can be executed independently.

Fig. 2.2.2 describes the design process model including design object model. The design process decomposes the design problem into sub-problems, and the design tasks at each abstract level are executed top-down. The design specification given as input must be a well-defined representation, and the design process must be modeled as a well-structured problem.

The fundamental task of the design at each level consists of planning, problem decomposition, refinement, optimization, analysis, and evaluation tasks. It corresponds to repetitive refinement in terms of the execution of these tasks.

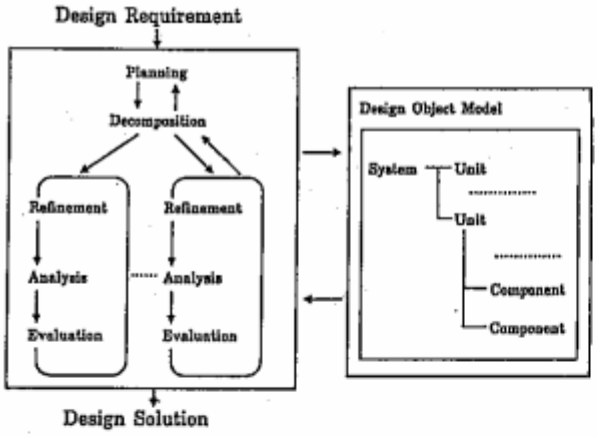

Fig. 2.2.2 Design Process Model

Especially, we must consider strategies for the decomposition of the problem or specification at each level in order to discuss the design process model for the routine design. The strategies of the problem decomposition must take into account the interactions between functional description and physical description for both circuit and mechanical design. These strategies are not always formalized clearly and applied in the mechanical design, as they are in the VLSI design. In VLSI design, the design process at each level is formalized to make the design task more simple, modular, and easy to apply.

In mechanical design, most interactions are caused by the execution of the constraint representation composed of the function concept, which is based on physical laws, and the feature concept of the form, such as the topology or geometry of the design object at a functional or physical level. The design problem must be dealt with by investigating the degree of decomposition of the problem or specification. The structure of the design object at the functional level is assumed to have already been decided when the above items are examined.

2.2.3 Classification of Routine Design

The design has three levels: new design, combinatory design, and parametric design [Tomiyama and Hagen 87].

New design is executed at each design phase as conceptual design, fundamental design, and detailed design. It is executed from scratch without using results of previous designs. It is a creative activity by the designer and the design process is modeled as an ill-structured problem.

Combinatory design realizes the design objects by combining the basic predetermined components according to the result of the previous design such that the input specification can be satisfied.

Parametric design determines and modifies the attribute parameters of the design objects when their structures can be fixed and the components can be described in the form of the attribute modeling.

Routine design applies to combinatory design and parametric design only. It includes tasks for both combinatory and parametric design.

3 EXPERT SYSTEM ARCHITECTURE FOR DESIGN TASKS

We will begin by discussing important current issues concerning expert systems, focusing on mechanical design and VLSI design. After that, we will describe related works, necessary architecture, and problem-solving mechanisms in order to consider a suitable expert system architecture for design tasks.

In mechanical design, it is difficult to modularize the design object because geometrical information (such as the representation in three dimensions) and manufacturing and assembly information are closely linked with the design object. The behavior of the design object changes as the geometric features change, because the geometric features or form description depend on the functional description or fabrication information. This behavioral change results from the dynamic creation of the model about the components of the design object.

In contrast to the VLSI design, feature description at the functional level has little effect on feature

description at the physical level and it is difficult to abstract the components of the design object from their behavior or function. Therefore, given the specifications, it is difficult to determine whether the behavior satisfies the specifications and it is necessary to consider the analysis task.

3.1 Previous Works

The present trend in architecture research for the design tasks can be divided into two types: the problem-solving based approach, and the design object modeling based approach. In research on the design object modeling based approach, the ICAD system are considered as typical example [Phillips and Rosenfeld 87].

In this section, research based on the problem-solving approach, called *generic task*, is described.

Typical research has been conducted on architectures consisting of primitive tasks for routine design, called *design generic tasks*, focusing on the mechanical design.

These architectures provide ways to structure knowledge for the various design descriptions and solve design problems, thus reducing the gaps between functions needed for the task in the design process and functions supported by expert system building tools.

We describe one branch of this research below. The DESI system [McDermott 78] represents an expert system for the design problem at an early stage. This system focuses on the design problem of an analog filter by regarding the design tasks as the problem-solving approach. It is the system that introduces *generic task* concept for the design problem.

Thus, in VEXPERT system [Dixon and Simmons 84, Dixon *et al.* 87], the design task is modeled using the problem-solving approach and is extended, and the design process of this system is regarded and modeled using the design-evaluate-redesign architecture. The redesign task in this design-evaluate-redesign architecture executes the new design according to the new design plan and the systems cannot realize the local modification facility for the analysis and evaluation of the result of the design.

The AIR-CYL system deals with the weak points of the redesign task differently from design-evaluate-redesign architecture based systems such as the VEXPERT system, in that it provides a local modification facility by improving the problem solver. DSPL is a design language supported for building the AIR-CYL system. It supports the ability to describe the design process model in terms of the combination of the problem-solving agents [Brown and Chandrasekaran 86]. It is rather difficult for the designer to build the system using this design language.

The PRIDE system [Mittal *et al.* 86] is the design support system, not for automated use, but for interactive use. In the PRIDE system, the problem-solving agent is described more easily than by using DSPL in the AIR-CYL system, and a local modification facility with multiple context management and a search control facility for the user are added and extended to the problem solver.

The AIR-CYL and PRIDE systems realize local modification facility, but when the form or structure of the design object, and the material for implementation are modified, it is very difficult for these systems to analyze and predict the behavior of the modified design object. Therefore, in this case, the behavior analysis of the design object based on the first principle of the physical environment is required. The PROMPT system [Murthy and Addanki 87] is a tool for design problems which facilitates the behavior analysis of a complicated design object by realizing the simulation mechanism, and by introducing deep knowledge and first principle to the *design generic task* concept.

In design problems, the optimization problem of parameters is especially required. The ENGINIOUS system [Nicklous *et al.* 87] is an automated design system that bases on the same design-evaluate-redesign architecture as VEXPERT system and integrates the simulation facility using the execution of CAE program and sophisticated techniques for optimization.

3.2 Architecture and Problem-Solving Mechanism

An expert system for routine design performs the following four tasks by examining existing systems, especially the mechanical design mentioned above [Nagasawa 87].

1) Determines structures (mechanisms) of the design object
2) Optimizes the attribute parameters for structures of the design object
3) Searches the attribute parameters for structures of the design object
4) Optimizes and transforms structures of the design object

The design for the expert system is applied to routine design only and includes combinatory design and parametric design.

Fig. 3.2 shows a basic structure of expert system for routine design based on the design process model. First, structures of the design object (structural model) are determined by searching the predefined design style of the design object, or by configuring or combining the predefined components. After determining structures of the design object, and refinement of the engineering model is executed. This refinement can be regarded as the optimization of the attribute parameters of structures of the design object and can be regarded as the

search of the attribute parameters considering implementation constraints such as resources. Structures of the design object are transformed or modified locally without the changes of the required functional or behavioral specifications by the optimization task, if possible.

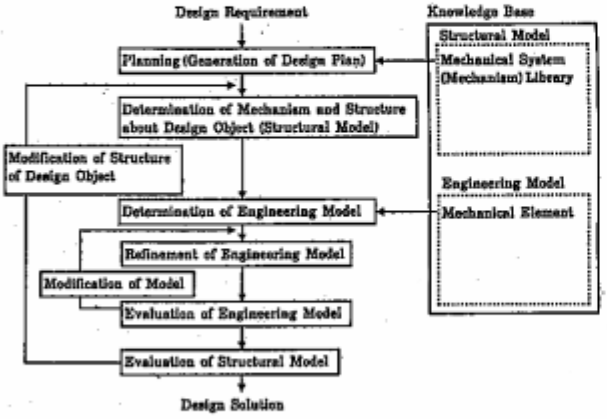

Fig. 3.2 Basic Structure of Expert System for Routine Design

In LSI design, after the structure of the design object is determined by combining the predefined components, the components and their attributes are refined. The structure is then transformed and optimized to resolve the trade-off problem so that the specifications are not violated.

In most mechanical design, structures or mechanisms of the design object are predetermined, and the search and optimization of the structural parameters are executed based on that predetermined structure or mechanism. Design where all the components and relations among them are determined after structures or mechanisms of the design object are determined is called parametric design.

Architectures based on the above *generic task* concept do not support the modeling facility of design knowledge, especially the design object; and when regarding the design knowledge as constraint representation, it seems that the architectures are insufficient for this *generic task* approach and unable to handle the constraint representation. Therefore, an architecture that includes the constraint representation, its application mechanism, and the modeling facility, is required.

This constraint representation is proposed as a new paradigm for knowledge representation, and the application mechanism is proposed as a new paradigm for the architecture of routine design expert systems.

The application mechanisms for constraint representation in routine design expert systems are defined as constraint-based problem solving. Modeling corresponds to the formalization of various descriptions for design knowledge and the handling of design knowledge by trial and error. Constraints are generated, derived, modified, or deleted from modeling during the design process.

The constraint-based problem-solving mechanism is considered according to the constraint classifications by matching the design process model for a routine design with the design system.

3.3 Technical Issues

There are many issues concerning expert system architecture for design tasks, called *design generic task*. The following items are shown as practical issues [Nagai 88a].

1) Relation of Expert System to Design System
2) Modeling of Design Process
3) Modeling of Design Object
4) Control of Problem Solving during Design Process
5) Improvement of Problem Solving and Knowledge Representation Mechanisms

We will describe our research approach for these items shown in Fig. 3.3.

To resolve the above issues, we focused on and investigated knowledge representation and a problem-solving mechanism to realize design process.

There are various kinds of knowledge in design problems. Two of them are design knowledge, composed of knowledge about the design object and knowledge about the problem solving.

Knowledge about the design object includes knowledge about the structure of functional and physical components. Knowledge about problem solving includes knowledge about ways to refine, analyze, and evaluate the design object.

When considering the expert systems for design task and expert system building tools for design problem, design knowledge, especially knowledge about the design object, must be separated from the problem-solving mechanism to make the architecture clear.

However, this design knowledge is not always classified explicitly and used separately. And current systems and tools do not support and handle the knowledge about the design object, and it is not enough that their problem-solving mechanism handles the design knowledge efficiently.

In Section 4, we investigated the design object model. The design problem requires ways to model the design objects so that we can use the knowledge that have about them. It also requires ways to solve problems by using this knowledge.

The FREEDOM system is explained as the modeling system of the design object according to this consideration.

By the way, when we consider the problem-solving mechanism for design knowledge in current systems, there are the problem-solving mechanisms both for knowledge about design object and for knowledge about problem solving, and each mechanism is not integrated into the identical problem solver. Fortunately, each mechanism can be integrated easily by regarding these knowledge as constraint representation.

However, existing expert systems and tools do not support and handle the constraint representation, and it is not enough that their problem-solving mechanism handles the constraint representation efficiently.

In Section 5, we discussed the application mechanism for constraints. By constraint, we means either general constraints necessary for routine design, or constraints that are specific to one.

As described above, the effective utilization of design knowledge and efficient problem solving are required to expert systems for design problems, because various knowledge must be treated.

Knowledge compilation is being investigated for raising of the efficiency of the problem solver in many problem areas, such as diagnostic and machine learning problems [Anderson 86].

This compilation is a technique by which knowledge in declarative form, such as facts and theories, about the domain is stored and this stored knowledge is applied and utilized by interpretive procedures. This technique makes existing paths of processing more efficient rather than enabling new paths of processing.

Therefore, more efficient procedures specific to the task domain can be generated using the knowledge compilation technique.

In our research, this compilation technique is applied to a design problems, especially mechanical design by considering the design knowledge as constraint representation.

Concretely, given the design plan generated by compiling constraint representation, derived from the concept of the design task and design object, and problem-solving heuristics, the design activity corresponds to the interpretation and execution of this design plan and it can be regarded as constraint satisfaction problem. The synthesis and analysis tasks in the design process are interpreted and executed according to this design plan generated by knowledge compilation on the predetermined system architecture. The interpretation and execution of the design plan corresponds to the execution of the design system.

However, at the moment there are many cases where only constraints are given; design plans are not.

Therefore, the environment of this design plan generation using the knowledge compilation technique and its interpretation and execution are required.

In Section 6, we describe the architecture for expert system and expert system building tool according above considerations.

At present, as various design systems including expert systems, especially for mechanical design, are implemented using a typical procedural language such as Fortran, Pascal and C, and these systems for only specific design problems are provided to designers as individualized systems. Therefore, it is inconvenient and inefficient for designers to use them for design.

In this case, for the designer, the design knowledge must not be represented merely in the form of the procedure. This knowledge must not be described specific with the specific problem solver.

To reduce these inconvenience and inefficiency of existing design systems, it is necessary for the designers to provide a support systems in which the designer can construct design systems easily.

After all, the facility where the designer can build the expert system for design problem by representing the design knowledge, not procedurally, but declaratively, and in the form independent of the problem solver, is required.

Thus, the environment of this design plan generation using knowledge compilation may be considered as support for a CAD system construction environment customizable by the designer.

In Section 7, the expert system building tool, called MECHANICOT, that supports the problem-solving mechanism that suits the design knowledge in the form of the declarative description, by considering their architectures as uniform framework from the viewpoint of constraint-based problem solving, is described.

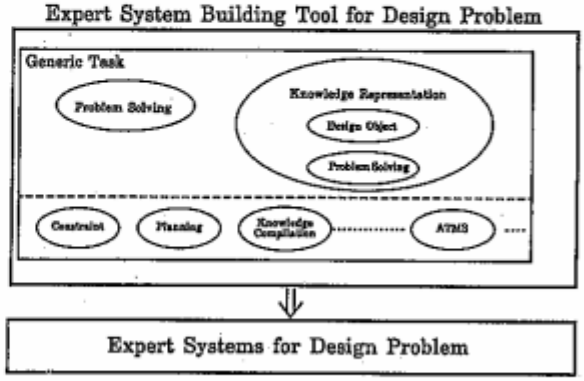

Fig. 3.3 Research Approach

4 DESIGN OBJECT MODEL

4.1 Object Model in the Design Expert System

4.1.1 Role of Design Object Model

Design objects in design systems are represented in the form of model descriptions. A design object model represents information and knowledge about design objects, such as their attributes, shapes, structures, and so on. During a design process, a model that satisfies all requirements is constructed; it represents a solution.

Models used in conventional design systems consist of data structures that are merely static. They need to be interpreted and manipulated in terms of design tasks or procedures. Only the knowledge about design methods is important, and the knowledge about design objects is embedded in model manipulation procedures or design methods. In conventional design systems, it is difficult to make effective use of the knowledge about design objects. Also, high performance design and the establishment of a general methodology by which to build design expert systems may be obstructed because two kinds of knowledge are not identified.

To avoid a combinatorial explosion and to solve design problems effectively, it is important to represent the knowledge about design objects as object models and to put those models to practical use in the design process. A framework must be developed that represents knowledge about design objects that help to make a design process support mechanism that can solve problems.

4.1.2 Frameworks for Representation

A frame system has been used to represent structures and attributes of objects in knowledge systems. Using a frame system, we can represent each element of an object in understandable, and modular form. Recently, an object-oriented paradigm whose concept is similar to frame system has been generally used and also applied to design problems. Though conventional object-oriented languages are suitable for representing structures, attributes and behaviors, they do not provide facilities for representing and using constraints on design objects. Constraints are typical and important knowledge in design problems.

Methods by which to represent knowledge about design objects that introduce constraints have been investigated [Stallman and Sussman 77] [Sussman and Steel 80] [Heintze *et al.* 87]. These provide efficient formalism for knowledge representation in terms of declarative description, but they are not suitable for the representation of large-scaled and complicated objects because they lack structural representation.

Investigations introducing constraints to an object-oriented paradigm have been done [Borning 81, Borning and Duisberg 86] [Harris 86] [Struss 87], and have made it possible to represent properties of design objects in an understandable form. However, only constraints on numerical attributes (instance variables) can be represented in these systems, and these constraints may be used only to calculate the values of attributes.

Representation of structures of design objects is an important part of solving a design problem, so a function to describe and use structural constraints is needed.

While the object model in an analytical problem such as a diagnose is formalized in a fixed form, one in a design problem takes a variable form because of the nature of the dynamic change. In a design problem, the model handling functions such as selection, modification, and refinement are important in design object modeling, since the designer constructs a object model.

4.1.3 Use of Object Model

The architectures for knowledge systems and expert systems are different according to the way the systems use the object model. We are examining two of these ways. One is to generate a design plan by analyzing and compiling knowledge about the design object model and design methods. We call this knowledge compiling method and will describe it in more detail later. It is suitable for a parametric design.

The second way is to provide a system for supporting design process interpreting knowledge that is described on object models. The system makes it possible to construct only models that satisfy the constraints, and also supports their effective construction.

This second way is suitable for a problem in which the structure of the design object is not given or not fixed. In such a case, the problem must be solved by trial and error or by interaction with users. Currently. the design object system is being developed and is explained in the next section.

4.2 FREEDOM : A Design Object Representation System

4.2.1 Basic Functions

Here, a knowledge representation system for design object modeling, A Framework for REprEsenting Design Object Model (`FREEDOM`), that facilitates an effective problem-solving mechanism, is presented [Yokoyama 88]. `FREEDOM` provides the facilities that keep the state of the model for constraint satisfaction and supports design tasks, and currently it is being developed and implemented using `ESP` language [Chikayamka 84] on a `PSI` machine [Taki *et al.* 84].

It is useful to distinguish between a model representing a solution and a model representing general and fundamental knowledge about design objects. Here, we call the former an instance model and the latter a template model. For example, in a mechanical design, the

fundamental structure of the machine and constraints on attributes are described as a template model, and the values of size and material attributes of parts are described as an instance model.

This system describes a template model and supports the creation of an instance model that satisfies the design requirements. Fig. 4.1 represents the basic structure of a design system that includes the FREEDOM system. Knowledge about a design object, namely an object model, is described in this system. Procedures caused by design tasks are positioned outside.

Procedures caused by design tasks correspond to the manipulation of an instance model on this system, modification of elements and values of attributes, addition of constraints, and so on. At this time, the FREEDOM system creates an instance model that satisfies the constraints for realization of an effective design.

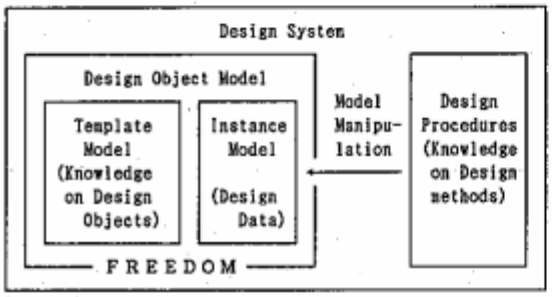

Fig. 4.1 Basic Structure of a Design System with FREEDOM

4.2.2 Knowledge Representation Framework

Knowledge representation provided in the FREEDOM system, based on an object-oriented paradigm, makes it possible to describe constraints in declarative form. A template model corresponds to classes in the object-oriented paradigm, and an instance model corresponds to instances. The features of knowledge representation are described below.

(1) Introduction of Constraints to an Object-oriented Paradigm

Constraints play important roles in solving problems in designs. They reduce a combinatorial explosion, and values of attributes and structures of objects can be determined using them. Thus, it is effective to introduce the constraints concept to an object-oriented paradigm. Constraints are described in the form of a predicate.

The whole-part relation, so-called *part-of*, is an important way to represent the structure of objects. The relation is classified into two: the first is that parts are needed to construct the whole; the second is that parts are not needed to construct the whole. For example, the relation of a rectangle and its four sides corresponds to the former case and the relation of a bookshelf and books in it corresponds to the latter case. The former is called a *consists-of* relation and can be regarded as a structural constraint of an object.

Because constraints may be generated dynamically during a design process, functions for addition or deletion of a constraint on an instance must be provided.

(2) Dynamically Changeable Relation between Class and Instance

In a design process, parts that satisfy the design requirement must be searched and the values of their attributes must be determined.

For example, after the values of attributes of an instance that corresponds to a selected class have already determined, the designer may want to perform an operation that changes the class to another class to which the instance belongs.

In this case, in existing object-oriented languages, we must remove an instance that belongs to old class, create an instance that belongs to new class, copy attributes common between these two classes, and remove the old instance.

Therefore, the FREEDOM system handles the relation between class and instance, the *instance-of* relation, which can be changed and maintained dynamically. This makes it possible to design effectively, because it is unnecessary to create a new instance and to copy attributes.

It is inefficient for a realization of a system to admit a dynamic change of class definition with no restriction. Moreover, it is not necessary to change a class definition dynamically because class change is needed when only a part of the corresponding instance is modified.

In general, class change is required because a top-down design is regarded as a refinement from a abstract level to a concrete level. An *is-a* hierarchy of classes which represents relations between an abstract level and a concrete level is applied to a refinement. A dynamic change of a limited class definition is refined according to a hierarchy of *is-a* relations.

(3) Class Hierarchy using *is-a* and *includes* Relations

As discussed before, a refinement is performed according to a hierarchy of *is-a* relations of classes. A class hierarchy with multiple inheritance is not always represented using a relation between an abstract level and a concrete level, but using inclusion relation of function in many object-oriented systems. It also makes a class hierarchy too complex to understand.

Thus, *is-a* relations can be declared only in the case when two classes belong to the same category, and they must be represented in the form of simple tree structures. A refinement in a design process corresponds to the search operation of a class that satisfies design requirements by referring to the *is-a* tree structure.

The multiple inheritance mechanism is useful for representing inclusion of functions, so a definition of this *includes* relation is introduced for representation of the class hierarchy.

Here, a class with an *includes* relation is not applied to the corresponding class, but to the corresponding subclass. Refinement is executed only to a specific class such as *include* class.

Fig. 4.2 represents an example of class hierarchy of plates. Relations, such as the *is-a* relation between *PLATE* and *BOARD*, the *includes* relation between *PLATE* and *METAL*, and the *is-a* relation between *IRON* and *METAL* are described. It is possible to simplify description of class hierarchies using the *includes* relation.

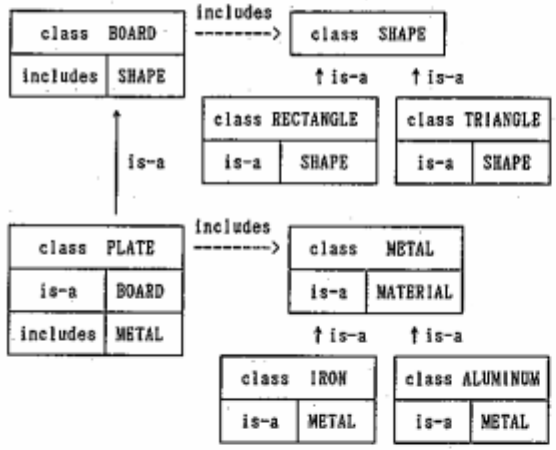

Fig. 4.2 Class Hierarchy with "is-a" and "includes" relations

4.2.3 Functions Required for Supporting the Design Process

FREEDOM provides functions to keep the states of instances, that satisfy constraints, derived from design object model. These functions make it possible to solve design problems effectively. The functions of this system are described below.

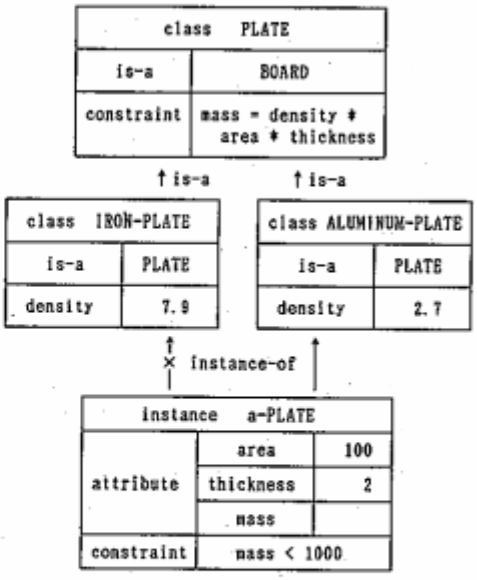

Fig. 4.3 Change of a Class by Constraint Satisfaction

(1) Maintenance of a Model that Satisfies Constraints

The attributes of a design object model are represented numerically or symbolically, and their values can be obtained by solving constraints derived from them. For example, when there is the constraint on four attributes shown below, such as mass, density, area and thickness, the value of one attribute is determined if values of other three attributes are given.

$$\text{mass} = \text{density} * \text{area} * \text{thickness}$$

In FREEDOM, a definition of a class is regarded as a description of constraints on objects belonging to a certain category. Search for a class that satisfies design requirements is realized using a constraint satisfaction mechanism.

In other words, when a structure or attribute of an instance is modified, if constraint satisfaction cannot be executed in the class to which it belongs, the class may be changed automatically to another class to satisfy the constraints. In this way, it is possible to search for a class that satisfies design requirements by a modification of a corresponding instance.

For example, a problem to determine the kind of a plate is considered. Fig. 4.3 represents a class hierarchy of plates.

Though a *a-PLATE* instance for representation of the solution corresponds to an *IRON-PLATE* instance, where an area and thickness are determined, its class is changed to *ALUMINUM-PLATE* according to the constraint on the mass.

(2) Support of Top-down Design

Generally, designs are executed in a top-down manner, so the main operations consist of refinements of objects. Refinement is executed in two ways: the first way is from the abstract level to the concrete level, and the second way is from the whole to parts. The former corresponds to refinement using *is-a* relations and the latter corresponds to refinement using *includes* and *consists-of* relations in FREEDOM.

Is-a relations are used to select a system structure or a component type, *includes* relations are used to refine parts of an object in parallel from several viewpoints, and *consists-of* relations are used to design some components divided from an object, independently.

A class declared as a part with a *consists-of* relation can be applied to a subclass of a declared class, such as a class with an *includes* relation. Thus, a design process can be considered as follows: first, an instance of an abstract class is created, then it is refined along an *is-a* hierarchy and divided into parts using *includes* and *consists-of* relations, and those divided parts are refined. This cycle is repeated in the design process.

As mentioned above, the FREEDOM system represents not only knowledge on design objects but also supports design tasks.

5 CONSTRAINT-BASED PROBLEM-SOLVING MECHANISMS

5.1 Constraint Representation

Constraints can be defined as certain relations or conditions that may exist between components of the design objects, as relations or conditions between properties of those design objects, and as expressions of laws or rules which must be satisfied (expressed in the form of equalities or inequalities). One example of an explicit constraint is a constraint on the structural information from the modeling representation of the design object. The other examples of constraints are *Kirchhoff's* laws and *Ohm's* law in circuit analysis [Stallman and Sussman 77] [Sussman and Steel 80], the number of resources, costs, operation priorities and dates in job-shop scheduling [Fox 83], and the edge connections that are physically possible in attempting to recognize a line drawing [Clowes 71]. However, the given representation of constraint is not always used effectively in existing systems. The effective use of such constraints should make it possible to restrict searches in the solution space, thus improving efficiency by eliminating unnecessary searches. Not many of the existing tools supporting the construction of expert systems provide an environment that makes it easy to express the constraint concept explicitly. Therefore, the person constructing the system must use the tool development language to attempt to realize mechanisms for applying constraint representations which depend on the design object. Most design problems can be regarded as constraint satisfaction problems. For instance, considering the concept *design generic task*, constraint handling is equivalent to checking the constraints in the test process of the generate and test method. In the step-wise refinement method, constraint handling involves refinement from an abstract level to a more concrete level, and the constraints imposed at each level also change dynamically. In general, when a set of given specifications is refined, it is also usually divided into different sub-problems, and there are often interactions between constraints on different sub-problems as well.

Those functions are required for problem solving according to the classification and effective solving mechanisms of constraints when addressing problems of a synthetic nature, such as in design and scheduling. The purpose of this is to provide the person constructing the system with an environment that enables the convenient representation of constraints.

5.2 Classification of Constraints [Nagai 88b]

(a) General (Domain-independent) Constraints for Routine Design

Constraints are classified according to the following characteristics.

1) Classification According to Generation Method

Constraints may be classified according to whether they are generated statically or dynamically. Static constraints are specified in advance, and are constant and unchanging. Dynamic constraints depend both on interactions with the user and on the system; they tend to change, with their range of applicability varying. Such constraints may be interpreted as incomplete knowledge, and, in order to manage changes in truth in the knowledge base accompanying changes in constraints, the functions of the Truth Maintenance System (TMS) [Doyle 79] and Assumption-Based Truth Maintenance System (ATMS) [deKleer 86] are necessary.

2) Classification According to Importance

Constraints may be classified according to importance into obligatory or requisite constraints, and suggestive constraints. When such a distinction is made, not all the constraints are selected and executed on an equal basis. That is, the importance of a constraint may depend on the context, the time, or another concept. All obligatory constraints must be satisfied, and these are given explicitly. Suggestive constraints are also referred to as weak constraints, and are used as guides in choosing the optimum branch at a node in the search tree. Such constraints may be described in rule form, and are given priorities and other attributes.

3) Classification According to Scope

Constraints may also be classified according to whether they apply locally or globally; this distinction is used in evaluating states in the search space. Local constraints are used to conduct searches when a state changes within a given model, object or process and the scope over which the constraint is valid is limited to within the model or object or process. Global constraints are used when a state is to be evaluated using not only local constraints, but all related constraints, without imposing any limit. For instance, when the solution space is divided and searches performed, this is equivalent to taking into account all those constraints that have been applied to states leading up to the present state, or to evaluating different parts of the solution space relative to each other.

4) Classification According to Propagating Variable Information

Constraints may be classified according to the propagating variable information, that is, depending on whether the variables of constraints propagate over value, or whether the constraint propagates over the interval bound in which the variable can take on a value or values. At present, the constraint logic programming system [Dincbas 86] [Heintze *et al.* 87], CONSTRAINT system [Sussman and Steel 80], and most other constraint systems [Borning 81] handle only constraints in which values are propagated.

Constraints that propagate over interval bounds in which variables can take certain values, are described using inequalities, and the variables of the constraint are not constant; these constraints propagate over the interval bound as a label. Most design problems include sub-problems that can be solved using the method of existing operations research; it is essential that the architecture should enable functions to operate in a unified framework based on constraint propagation in labels with interval bounds [Davis 87].

When considering practical design problems, there are some combinatory possibilities of handling of above classified constraints.

(b) Domain-specific Constraint for Routine Design

There are various domain-specific constraints for routine design. These constraints are related to the simplified design process composed of the conceptual design, fundamental design and detailed design.

In particular, structural constraints should be considered in routine design. Structural constraints are reflected in terms of the design style, and specifications, and requirements at each abstract level of the design, and determine the structural decomposition, partition, and design style at a lower design level. In hierarchical design, it should be noted that the constraints are propagated to a lower design level. The design style constraints decide the structure of the design object and the problem decomposition at a lower design level. Constraints are partitioned through the structure of the design object and decomposition of the design problem. For example, there are the implementation constraints (technology-dependent constraints at the implementation level).

5.3 Necessary Functions

The functions required for constraint-based problem solving [Nagai 88b] are listed below.

1) Function for Constraint Propagation and its Control

In the process of satisfying constraints, and when values are assigned to variables of the constraint, the values of other constraint variables may be determined by the former variable; this is the mechanism of constraint propagation. Such a mechanism must take into account both cases considering local constraint propagation and cases where the problem cannot be solved by local constraint propagation alone. A typical example of the former is the propagation method using data-flow analysis introduced in the CONSTRAINT system. An example of the latter case is the variable elimination method of simultaneous equations. In particular, when using propagation methods based on data-flow analysis, the trade-off between constraints, such as occur when regarding the TMS as a constraint satisfaction problem (CSP) [Dechter and Pearl 87], may result when the propagation is not always sufficient. Clearly, a strategy for controlling constraint propagation is needed. Interactions between constraints and the least commitment of constraints are also indispensable for realizing the constraint propagation. For instance, in practical design problems, if we consider design by step-wise refinement, interaction between constraints applying to sub-problems that are solved separately is extremely important. One approach to the problem of constraint interactions is to minimize interactions between sub-problems. This approach is the one adopted in the MOLGEN system [Stefik 81a, 81b]. It is referred to as the least commitment; by delaying constraint evaluations as far as possible, refinements according to the design plan are executed, and evaluations are performed when necessary.

2) Constraint Relaxation and Selection

Relaxation and selection are applied to weak constraints. Relaxation of a constraint is equivalent to searching for alternatives to the specified constraint. That is, at the failure stage, when a constraint has not been satisfied, an alternative constraint, at the same or a lower level, is sought. Selection involves the choice of a constraint when there are two or more competing

constraints, and is regarded as constraint interpretation. In this way, it is thought that constraint relaxation and selection can be formulated as a planning problem [Descotte and Latombe 85].

3) Preservation and Management of Dependency Relations

In processes where the values of constrained variables are propagated through the execution of constraint propagation mechanisms, when contradictions in variable values arise, the preservation and management of dependency relations among constraints, variables, and constant values are deemed important to resolve such contradictions and to explain the propagating values [Harris 86].

4) Monitoring Mechanism for Constraint Evaluation

A monitoring mechanism for constraint evaluation should not be omitted from any problem-solving mechanism that relies on constraint representations. It manages constraint checks and ensures consistency, and is to some extent realizable using demons or attached procedures.

5.4 Role of this Mechanism to Design Process

This section considers constraint-based problem solving relative to the design process. As discussed previously, the fundamental task at each design level makes the iterative design composed of the problem decomposition and refinement proceed according to the design plan. If a design fails, redesign is executed, and the problem is decomposed and refined again. It backtracks the previous design decisions in the tasks at the higher level or executes local modification at the same level, and executes the iterative design.

Mechanical design that mainly belongs to parametric design can be regarded as the generate and test + failure recovery (+ optimization + analysis and evaluation) paradigm.

Planning decomposes and refines the problem or specification according to the design plan. The design style determined from the design plan (the configurational or architectural knowledge about the design object) is indexed by the requirement or specification of the design, and can be regarded as a constraint. The refinement, optimization, and analysis and evaluation tasks are selected and executed according to this constraint. The decomposition of the requirements or specifications of the design are executed by applying this design style constraint.

Parameters and constraints between the hierarchies of the design are propagated upward or downward or the interactions among the decomposed subproblems occur when there are constraints among them, so it is necessary to consider the tasks for these types of processing, such as constraint posting and propagation.

Assuming that problem decomposition can transform or map the sub-problem to the component or assembly, there are two ways to decompose a problem. The first way is problem decomposition into subproblems with interactions; the second way is problem decomposition into independent sub-problems. In the first way, it is important to consider the relations among the compositions at the same level. In the second way, it is important to consider the relations between the components and sub-components.

Refinement transforms the divided specification into structural representation composed of the components and relations among them. These relations among components can be regarded as a constraint. Constraints on the component attributes are particularly important. For example, the propagation mechanism of constraints of the component attributes, in decomposition into interacting sub-problems is different from that in decomposition into independent subproblems. The former mechanism propagates the interactions among sub-problems as the constraints, and the latter propagates the constraints upward or downward according to the hierarchical representation of the design object when there are no interactions among independent sub-problems.

Optimization modifies the structural representation locally, so that the functions expressed in the specification do not change.

5.5 Constraint-based Problem Solving on Generate and Test (G & T) + Failure Recovery (FR) Paradigm

Tasks for routine mechanical design consist of the determination of structures and structural parameters of the design object, without the optimization and transformation of structures of the design object.

The structure of the design object in routine design is determined by combining the components or is determined according to predefined design styles of the design object. It is determined by retrieving the appropriate design style from the predetermined design plan.

The components are implemented using the standard parts by looking them up in catalogues or by using non-standard parts by the design. Most of the selection strategies of standard or non-standard parts for component implementation are described in the specifications or requirements. They mostly depend on a trade-off of the performance against cost.

The necessary architecture for this design can be formalized as the generate and test + failure recovery (+ optimization + analysis and evaluation) paradigm shown in Fig. 5.5.

The problem-solving primitives are the generator, propagator, tester, and failure recoverer.

The generator assigns values to parameters or assigns functional components to the components for implementation. This parameter can be classified in one of two ways according to the type of the value: one takes the continuous value, and the other takes the discrete value. The former assigns parameters of the attributes by local modification based on the predetermined component. The latter assigns parameters by retrieving the standard parts for implementing components from the catalogue, a table look-up method.

The propagator assigns the values to the parameters by using the active evaluation of the constraint and propagation of constraints.

The tester checks the constraint and can be considered as the passive handler of constraint. In general, the inequality description can be handled by the tester, but in some contexts it can also be considered and handled as the generator.

The failure recoverer modifies the attributes of the components locally using the advice mechanism and replans the problem decomposition. The advice mechanism can be considered as the repair of the partial or local design using the heuristics about the attributes of the components. It uses the above generator and propagator as primitives. In failure recovery handling, the obligatory or suggestive constraints must be handled. The advice mechanism by the selection and evaluation of the constraint is executed to the obligatory constraint. For the suggestive constraints, planning such as a compromising algorithm is required in order to relax and select this constraint. This is a mechanism that satisfies as many constraints as possible, too.

Design knowledge, especially the constraint representation such as the various formulas about the features of the functional or physical environment is compiled and this compiled representation can be considered as the constraint network, when the structure of the design object, configuration of the components, is fixed.

In other words, the constraint-based problem-solving mechanism in the generate and test + failure recovery architecture corresponds to the execution of the representation generated by compiling various kinds of this design knowledge based on the fundamental tasks of the design process such as planning, problem decomposition, and refinement.

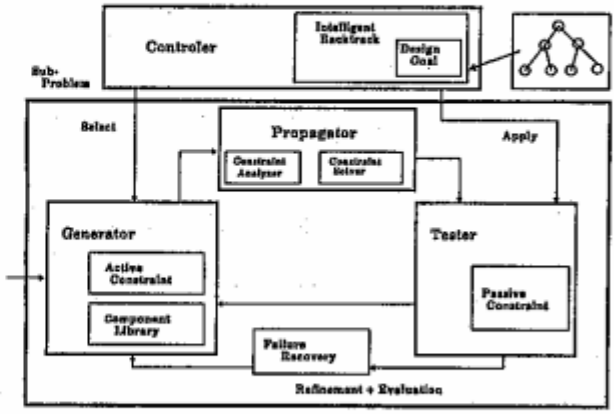

Fig. 5.5 Constraint-based Problem Solving on G & T + FR Paradigm

6 ARCHITECTURE FOR AN EXPERT SYSTEM BUILDING TOOL

6.1 Knowledge Representation

Designers' knowledge must be easily represented in a building tool, to enable them to build an expert system by themselves. In this section, representation of design knowledge about problem solving is described, since an object model which is knowledge about a design object itself has already been discussed in Section 3.3. Design knowledge about problem solving consists of methods to analyze object models, to evaluate and modify solutions, and plans to design the object and search from candidate solutions.

Methods to analyze object models are regarded as constraints; they are expressed in the form of formulas derived from physical laws or experiments, and are composed of equalities, inequalities and mathematical functions. Methods to search from catalogues, tables, and graphs are also included in this knowledge, in terms of deciding parameters. Knowledge to modify solutions generates alternatives after determining modification scope, if solutions cannot satisfy design requirements. Modification scope is not only locally close within a subproblem, but also globally related to other subproblems. Knowledge to evaluate solutions, often expressed by inequalities, decides whether solutions can satisfy design requirements such as functions, efficiency, and cost performance. Plans to design indicate orders to solve constraints given as analysis methods and requirements, but plans to search restrict the search space when many alternatives exist. Using this

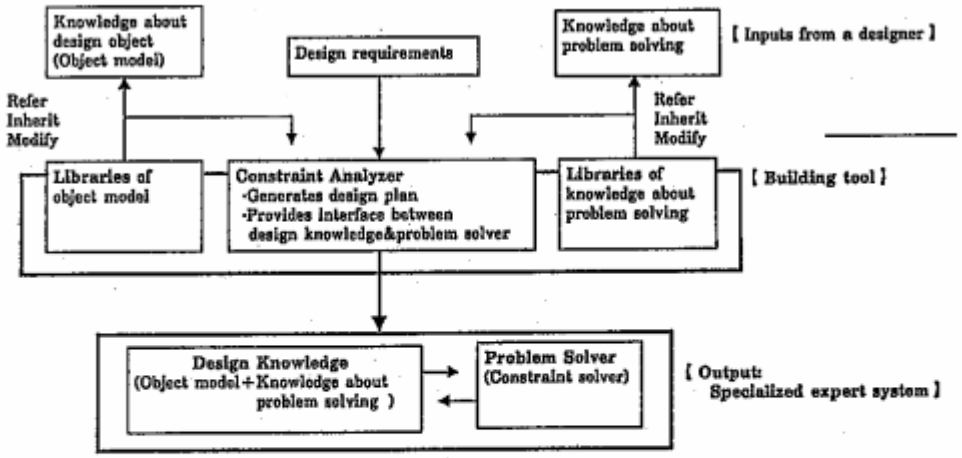

Fig.6.2 Architecture for an Expert System Building Tool

with evaluation knowledge, problem solving is made more efficient. Thus, design knowledge about problem solving has various representation types.

In addition, knowledge that is independent on a design object and heuristics that is closely dependent on a certain design object are mixed. For example, design formulas and searching from catalogues in design knowledge about problem solving, and basic parts and function units in object models are independent from a design object.

Therefore, with the aim of enabling designers to represent this knowledge easily, we employed the approach that those independent kinds of knowledge are prepared as system libraries: sets of design formulas and catalogues in knowledge about design problem solving, and sets of basic parts and function units in object models are prepared respectively. And these knowledge are expressed in object-oriented. Because an object-oriented system has advantages that parts and their attributes are represented naturally as *object* for object models, and knowledge can describe declaratively as *methods* for knowledge about problem solving. Consequently, designers' heuristics can be expressed explicitly, by referring to or by inheriting and modifying libraries.

6.2 Architecture for an Expert System Building Tool

As shown above, we divide design knowledge into object models and knowledge about problem solving. This enables us to maintain knowledge and to modify knowledge flexibly. Viewing knowledge and requirements as constraints, constraint based problem solving is employed.

To build an expert system suitable for a design problem by a designer, we propose a building tool that regards inputs of design knowledge as constraints, generates design plans by analyzing their dependencies, and provides an interface between design knowledge and a constraint solver.

We used a constraint analyzer to obtain facilities for this building tool. The constraint analyzer, similar to a knowledge compiler [Araya and Mittal 87] [Nagai 88c] and a constraint compiler [Feldman 88], analyzes dependencies among constraints and produces a design plan to solve a problem efficiently. In other words, the constraint analyzer specializes knowledge by combining knowledge independent from a certain design object and designers' heuristics which depend from a certain design object. Fig. 6.2 shows the architecture of the tool.

Inputs to the tool are design requirements, object models, and knowledge about problem solving. They are given by specifying system libraries, or by modifying libraries with referencing or inheriting libraries. Referencing results of previous design and designers' heuristics about searching from alternatives are also represented as knowledge about problem solving.

From these inputs, the tool analyzes dependencies among constraints and parameters, generates a design plan, and provides an interface between design knowledge and the constraint solver. The output from the tool is a specialized expert system including designers' heuristics.

7 MECHANICOT: A SPECIFIC EXPERT SYSTEM BUILDING TOOL

As an example of a specific expert system building tool, MECHANICOT [Terasaki *et al.* 88], which is under development, is described. MECHANICOT is a tool for a mechanical parametric design. It analyzes dependencies between structures of a design object and parameters, produces a design plan, and builds a specialized design expert system.

7.1 Design Problem of Main Spindle Head in a Lathe

The design object is the main spindle head of a lathe, shown in Fig. 7.1. It consists of a main spindle to grip a workpiece and to rotate it, a motor as a power source, V-belts and a pair of pulleys to transmit power from the motor to a pulley-shaft, bearings to support both the main spindle and the pulley-shaft, and two pairs of gears to change the main spindle speed. The problem is to determine the dimensions of each part and find each part number by searching catalogues.

The design requirements must be satisfied:

- Cutting capacity and the maximum diameter of the workpiece
- Maximum rotating speed of the main spindle
- Maximum cutting depth and feeding speed
- Minimum life time of the bearing to be evaluated

This is a parametric design problem in which the structure of the design object is fixed and knowledge about problem solving is well known [Inoue *et al.* 88]. In addition, parameters are discrete values which are decided by searching from catalogues or by adjusting to standard values. Tab. 7.1 shows input requirements and design parameters.

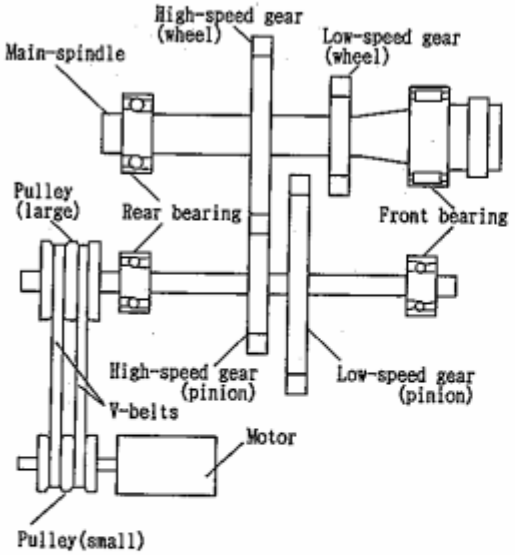

Fig. 7.1 Outline of Design Object
— Main-spindle Head Of Lathe —

Tab. 7.1 Example of Input & Design Parameters

Input parameters	
Cutting capacity	Workpiece material Tool material Workpiece diameter(max.) Main-spindle speed(max.) Cutting depth (max.) Feeding speed (max.) Drill diameter (max.) Drill speed (max.)
Evaluation	Life of bearings
Design parameters	
Decided by calculation	Main-spindle diameter Pulley-shaft diameter Gears & pulleys ratio Number of gear teeth Gears pitch diameters
Result of previous design	Bearing mount type Bearing span
Search from catalogues or tables	Bearing part number Motor part number V-belt part number Pulleys part number

7.2 Specific Example of an Expert System Building Tool

7.2.1 Input

a) Design Object Models

Design object models consist of the structure of the design object, constraints derived from structural relations, and design parameters. The structure is hierarchically represented: the whole of the design object, function units, and parts which are elements of a function unit are described as *class objects* respectively. Fig. 7.2.1.1 shows this class hierarchy in the case of Fig. 7.1. Description of object models is composed of *consist_of* for component elements in a function unit, *constraint* for constraints from structural relations, and *parameter* for design parameters. Fig. 7.2.1.2 (a) shows the object model definition for a class *pulley_shaft* which is one of the function units in the main spindle head.

b) Knowledge about Problem solving

Description of knowledge about problem solving is shown in Tab. 7.2.1. Equalities and inequalities, and search method from catalogues, tables and results of previous design are included in the form of table, excepts in the form of functions and equalities. There are two types in inequalities: one is used for limiting generating space of alternatives, and the other is used for testing solutions. Searching from catalogues, tables may have alternatives, similar to inequalities which used for restricting search space. To handle such constraints, they are classified into four types: *design_method*, *generator*, *tester*, and *adjust_by*.

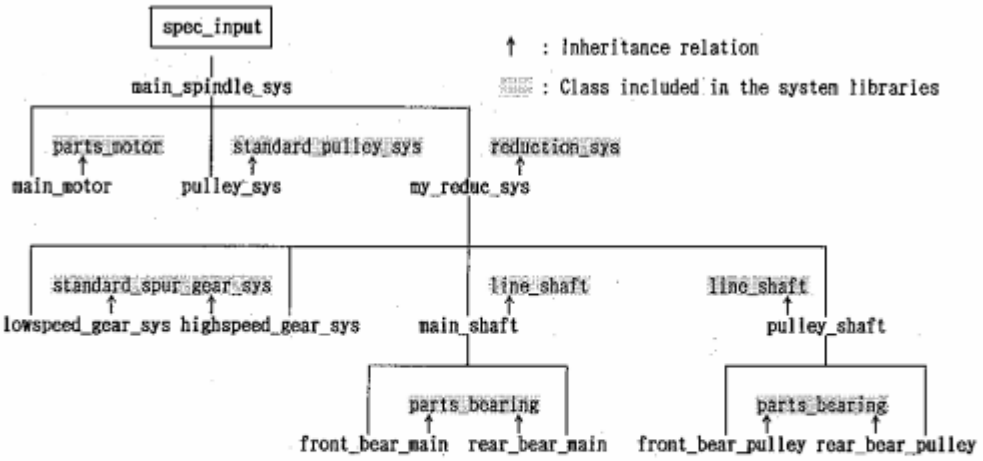

Fig. 7.2.1.1 Hierarchical Class Structure of Main Spindle Head

```
class_name
        pulley_shaft;
inherit_from
        line_shaft;
consist_of
        front_bear_pulley, rear_bear_pulley;
parameter
        front_bearing_type, rear_bearing_type;
constraint
        #front_bear_pulley!shaft_dia := shaft_dia,
    % If 'shaft_dia' is designed, propagates it to 'shaft_dia'
    % in the class 'front_bearing'. This is a constraint to
    % fit a bearing on the pulley shaft.
        #front_bear_pulley!type := front_bearing_type,
        #front_bear_pulley!shaft_rpm := rpm_max,
        #rear_bear_pulley!shaft_dia := shaft_dia,
        #rear_bear_pulley!type := rear_bearing_type,
        #rear_bear_pulley!shaft_rpm := rpm_max;
end.
```

```
class_name
        line_shaft;
parameter
        shaft_dia, hole_dia, material, rpm_max,
        twisting_moment, shearing_strength,
        torsion_angle;
end.
```

Fig. 7.2.1.2 (a) Object Model Definition for the class pulley_shaft

Functions and search method from catalogues or tables which have no alternatives indicated by *design_method*, whereas having alternatives such as inequalities used for restricting search space are expressed by *generator*. And *tester* represents inequalities used for testing solutions led from *generator* or evaluating solutions. To adjust solutions to standard values as a filter is *adjust_by*. Knowledge about design problem solving is presented by specifying a *method_name* and a type of method as described above to a parameter in an object model. Fig. 7.2.1.2 (b) shows an example of the object model definitions including *design_method* description for the class *pulley_shaft*.

Tab. 7.2.1 Description of Knowledge about Problem Solving

Description type	
Functions, equalities (Results of previous design) (Catalogues or tables search)	design_method (No alternatives)
Inequalities (Limiting generating scope) Catalogues or tables search Refer from results of previous design	generator (Generate alternatives)
Equalities Inequalities	tester (Evaluation & test)
Tables search	adjust_by (Filter)

c) Design Requirements

Design requirements indicate the design object, parameter names which are given as an input, and relations between design parameters and input parameters. To express the design object, the highest class name in a class structure is given by *design_object*. For instance, the class name for *design_object* is the class *main_spindle_sys* in the case of Fig. 7.2.1.1. Names of

```
class_name
        pulley_shaft;
inherit_from
        line_shaft;
consist_of
        front_bear_pulley, rear_bear_pulley;
parameter
        front_bearing_type, rear_bearing_type;
constraint
        #front_bear_pulley!type := front_bearing_type,
                ~
        #rear_bear_pulley!shaft_rpm := rpm_max;
design_method
        { [front_bearing_type, rear_bearing_type],
        bearing_mount_search(#result_of_previous_design,
        front_bearing_type, rear_bearing_type)
        };
   % Parameters 'front_bearing_type' and 'rear_bearing_type'
   % are designed by the method 'bearing_mount_search' in the
   % class 'result_of_previous_design'.
end.
```

```
class_name
        line_shaft;
parameter
        shaft_dia, hole_dia, material, rpm_max,
        twisting_moment, shearing_strength,
        torsion_angle;
design_method
        { [shaft_dia],
        shaft_dia(#shaft_dia_calc, twisting_moment,
        torsion_angle, shearing_strength,
        hole_dia, shaft_dia)
        },
   % Parameter 'shaft_dia' is designed by the method
   % 'shaft_dia' in the class 'shaft_dia_calc'.
   % And parameters 'twisting_moment','torsion_angle',
   % 'shearing_strength' and 'hole_dia' are input for this method.
        { [shearing_strength],
        shearing_strength_search(#material_data_base,
        material, shearing_strength)
        } ;
end.
```

Fig. 7.2.1.2 (b) Adding Design Methods for the class pulley_shaft

input parameters are represented by *parameter*. Note that specific values of input parameters are given when the system which is produced by the tool is actually executed. The relations between input parameters and design parameters are given by *constraints*, indicating which design parameter in a class receives a value from a input parameter.

7.2.2 Design Plan Generation

Design plan generations using the constraint analyzer after being given design knowledge are described below. Generating a design plan, similar to data-flow analysis in a compiler, is accomplished as shown.

1) Subgoals are assigned to *constraint*, *generator*, *tester*, *design_method*, and *adjust_by* in each class. In the case of processing *constraint*, a subgoal is assigned to each constraint statements. Otherwise, a subgoal is given to each methods. Note that the name of each subgoal should be unique. Fig. 7.2.2 (a) shows the assignment of this subgoal.

2) Subgoals are integrated into some goals, based on the input-output dependencies of parameters. The way of giving names to goals is exactly the same as the case of subgoals. Fig. 7.2.2 (b) shows this grouping from subgoals to goals.

3) An execution sequence of goals is determined based on the input-output dependencies of goals. The sequence is managed in a class that is one level higher than the class in which the goal is included. For instance, in the Fig. 7.2.1.1, the sequence of goals in the class *pulley_shaft* is managed by the class *my_reduc_sys*. And goals in the class *main_spindle_sys* are controlled by the class *spec_input*. Fig. 7.2.2 (c) shows the hierarchical control of the calling sequence.

Analyzing dependencies between constraints initiates from the lowest level of the class hierarchy. That is to say, the level including a class *input_shaft* and a class *front_bear_pulley* exist on the start level, in example of Fig. 7.2.1.1. The analysis proceeds towards the highest level class *spec_input*. In the case when inheritance relations are exist, the constraints are not processed along class hierarchy between parent classes and a child class, but are treated as a flat set of the constraints included in both parent classes and their children classes.

The advantage of this constraint analyzer is to analyze the relation between *generator* and *tester* for realization of an efficient execution of generate and test loop.

When G & T loops are included in execution statements, a *generator* corresponding to a *tester* is found by

```
class_name
        pulley_shaft;
              ~
constraint
        #front_bear_pulley!shaft_dia := shaft_dia,
              ⇒ pulley_shaft_subgoal1
              ~
        #rear_bear_pulley!shaft_rpm := rpm_max;
              ⇒ pulley_shaft_subgoal6
design_method
        {[front_bearing_type, rear_bearing_type],
        bearing_mount_search(#result_of_previous_design,
                ...)
        };      ⇒ pulley_shaft_subgoal7
end.

class_name
        line_shaft;
              ~
design_method
        { [shaft_dia],
        shaft_dia(#shaft_dia_calc, twisting_moment,
                ...)
        },      ⇒ line_shaft_subgoal1
        { [shearing_strength],
        shearing_strength_search(#material_data_base,
                ...)
        };      ⇒ line_shaft_ subgoal2
end.
```

Fig. 7.2.2 (a) Subgoals Assignment

Fig. 7.2.2(b) Grouping of Subgoals as Goals

analyzing dependencies of constraints. It is considered that an execution of those statements is equivalent to a realization of the Dependency-Directed Backtracking (DDB) mechanism.

7.3 Considerations

The MECHANICOT system provides a design support environment where a designer can input and modify design requirements, design knowledge composed of the model of design object and design process easily and where the design plan can be generated using constraints derived from that knowledge.

At present, the MECHANICOT system is being developed and implemented using ESP language on a PSI machine. This system is in the style of an automated system with no user interaction. It receives design requirements and design object representation written in ESP-like language as input, and generates the design plan written in ESP as output. The following items are not provided sufficiently or are missing.

(1) Support of Multiple Context Management

The design object must be modeled and represented from various points of view, as shown in Section 4. These points of view to the design object can be interpreted as the design contexts. A multiple context management mechanism is required for the execution and evaluation of design object models under certain design contexts as alternatives. This is a very effective and important mechanism for design systems.

(2) Improvement of Constraint Analyzer

In this system, only the handling of static con-

```
class  spec_input has
design_object  block1;

    goal(#spec_input) :-
        spec_input_goal1(#spec_input),
        block1_goal1(#block1),
        block1_goal3(#block1),
        block1_goal2(#block1);

    spec_input_goal1(#spec_input) :-
        spec_input_subgoal1(#spec_input),
        spec_input_subgoal2(#spec_input);
end.
```

```
class block1 has
consist_of  parts1;

    block1_goal1(#block1) :-
        block1_subgoal1(#block1),
        block1_subgoal2(#block1);

    block1_goal2(#block1) :-
        block1_subgoal4(#block1),
        block1_subgoal3(#block1);

    block1_goal3(#block1) :-
        parts1_goal1(#parts1),
        parts0_goal1(#parts1),
        parts1_goal2(#parts1);
end.
```

```
class  parts1 has
inherit_from  parts0;

    parts1_goal1(#parts1) :-
        parts1_subgoal1(#parts1);

    parts1_goal2(#parts1) :-
        parts1_subgoal3(#parts1),
        parts1_subgoal2(#parts1);
end.
```

```
class parts0 has

    parts0_goal1(#parts0) :-
        parts0_subgoal1(#parts0),
        parts0_subgoal2(#parts0);
end.
```

Fig. 7.2.2(C) Hierarchical Control of Calling Sequence

straint and obligatory constraint is considered. For example, because the role of a constraint such as a generator and tester to a constraint-handling mechanism is predetermined, the interpretation of a constraint is fixed.

However, the handling of dynamic constraint, such as addition, deletion and modification of constraints during design process, and suggestive constraint, is not investigated.

Therefore, both static analysis for constraint and dynamic analysis, including constraint relaxation, are required for realization of dynamic constraint handling, considering a current constraint analyzer.

(3) Improvement of Constraint Solver

In this system, a specific mechanism for the constraint-based problem solving shown in section 5, is not realized, and a constraint propagation is performed using unification function in ESP language. Moreover, for the above dynamic constraint handling, constraint solver including constraint propagation and relaxation mechanism, is required.

(4) Improvement of User Interface

Currently, the system does not provide a friendly user interface, where the designer can give knowledge about design requirements and the design object in the form of a schematic description as an input, interacting with the system.

Such a user interface facility linked with a design object modeling facility is required.

The design plan generated using constraint analyzer is executed by an inference mechanism of ESP language, but in future an interpreter for the design plan will be implemented and this design plan description will be interpreted and executed by it.

8 CONCLUSION (FUTURE WORK)

In conclusion, the architecture of expert systems, including the design object modeling facility for routine design, was proposed by focusing on constraint-based problem solving composed of constraint representation and the application mechanism.

For realization of this architecture, the design object representation system, called FREEDOM, and the design support system, called MECHANICOT, were described. Particularly, the MECHANICOT system supports machining tools, specifically a main spindle head of a lathe, a design target.

Our future research is to clarify the architecture of expert systems for various routine designs, such as LSI design, mechanical design, and configuration, by regarding constraint-based problem solving as a new paradigm. In other words, this research is to propose *generic tasks* for various routine designs. We will also propose primitive tasks for the constraint-based problem solving required to realize the architecture of expert systems for various routine designs.

Furthermore, incorporation of knowledge acquisition system, especially for acquisition of design knowledge using design object modeling facility and a sophisticated user interface, and the use of ATMS as a knowledge maintenance system is required in order to realize more effective and practical design system.

ACKNOWLEDGMENTS

We would like to thank Mr. Yuichi Fujii, Chief of the Fifth Research Laboratory for encouraging research and to thank Mr. Katsumi Inoue and other members of the Fifth Research Laboratory for helpful comments. We would also like to thank Dr. Isao Nagasawa, Kyushu University, for useful comments on needs of design plan generation for mechanical design. Furthermore, we would also like to thank Mr. Satoshi Imamura and Dr. Toshio Kojima, Mechanical Engineering Laboratory, for technical support to a formalization of the design problem of main spindle head in a lathe. Finally, we would like to express special thanks to Dr. Kazuhiro Fuchi, Director of ICOT Research Center, who has given us the opportunity to carry out research in the Fifth Generation Computer Systems Project.

REFERENCES

[Anderson 86] J.R. Anderson, "Knowledge Compilation: The General Learning Mechanism", Machine Learning, An Artificial Intelligence Approach, Vol. 2, R.S. Micahlski, J.G. Carbonell and T.M. Mitchell (ed.), Morgan Kaufmann Publisher, Inc., 1986

[Araya and Mittal 87] A.A. Araya and S. Mittal, Compiling Design Plans from Descriptions of Artifacts and Problem Solving Heuristics", Proc. of IJCAI-87, 1987

[Borning 81] A. Borning, "The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory", ACM Trans. on Programming Language and System, Vol. 3, 1981

[Borning and Duisberg 86] A. Borning and R. Duisberg, "Constraint- Based Tools for Building User Interfaces", ACM Trans. on Graphics, Vol.5, no.4, pp345-374, 1986

[Brown and Chandrasekaran 86] D.C. Brown and B. Chandrasekaran, "Knowledge and Control for a Mechanical Design Expert System", IEEE COMPUTER, 1986

[Chandrasekaran 86] B. Chandrasekaran, "Generic Tasks in Knowledge-based Reasoning: High-Level Building Blocks for Expert System Design", IEEE expert, 1984

[Chikayama 84] T. Chikayama, "Unique Features of ESP", Proc. of International Conference on Fifth Generation Computer Systems, 1984

[Clowes 71] M. B. Clowes, "On Seeing Things", Artificial Intelligence, Vol. 2, 1971

[Davis 87] E. Davis, "Constraint Propagation with Interval Labels", Artificial Intelligence, 32, 1987

[Dechter and Pearl 87] R. Dechter and J. Pearl, "Network-based heuristics for constraint satisfaction problems", Artificial Intelligence, Vol. 34, 1987

[deKleer 86] J. de Kleer, "An Assumption-Based TMS", Artificial Intelligence, Vol. 28, 1986

[Descotte and Latombe 85] Y. Descotte and J.- C. Latombe, "Making Compromises among Antagonist Constraints in a Planner", Artificial Intelligence, 27, 1985

[Dincbas 86] M. Dincbas, "Constraints, Logic Programming and Deductive Databases", France-Japan Artificial Intelligence and Computer Symposium 86, 1986

[Dixon and Simmons 84] J. R. Dixon and M. K. Simmons, "Expert Systems for Design: Standard V-Belt Drive Design as an Example of the Design-Evaluate-Redesign Architecture", Proc. of ASME Computers in Engineering Conference, 1984

[Dixon *et al.* 87] J. R. Dixon, A. Howes, P. R. Cohen, and M.K. Simmons, "DOMINIC I: Progress Towards Domain Independence In Design By Iterative Redesign", Proc. of ASME Computers in Engineering Conference, 1987

[Doyle 79] J. Doyle, "A Truth Maintenance System", Artificial Intelligence, Vol. 12, 1979

[Eastman 81] C. M. Eastman, "Recent Developments in Representation in the Science of Design", Proc. of IEEE 18th Design Automation Conference, 1981

[Eliyahu *et al.* 87] O. Eliyahu, L. Zaidenberg, and M. Ben-Bassat, "CAMEX - An Expert System For Process Planning On CNC Machines", Proc. of AAAI-87, 1987

[Feldman 88] R. Feldman, "Design of a Dependency-Directed Compiler for Constraint Propagation", Proc. of 1st International Conference on Industrial and Engineering Application of Artificial Intelligence and Expert Systems (IEA/AIE-88), 1988

[Fox 83] M. S. Fox, "Constraint-Directed Search: A Case Study of Job-Shop Scheduling", CMU-RI-TR-83-22, 1983

[Harris 86] D. R. Harris, "A Hybrid Structured Object and Constraint Representation Language", Proc. of AAAI-86, 1986

[Heintze *et al.* 86] N. C. Heintze, J. Jaffar, C. Lassez, J.-C. Lassez, K. McAloon, S. Michaylov, P. J. Stuckey, and R. H. C. Yap, "Constraint Logic Programming: A Reader", Proc. of Fourth IEEE Symposium on Logic Programming, 1987

[Inoue *et al.* 88] K. Inoue, Y. Nagai, Y. Fujii, S. Imamura, and T. Kojima, "Analysis of the Design Process of Machine Tools - Example of a Machine Unit for Lathes - " (in Japanese), ICOT Technical Memorandum, TM-494, 1988

[Kowalski and Thomas 83] T. J. Kowalski and D. E. Thomas, "The VLSI Design Automation Assistant: Prototype System", Proc. of IEEE 20th Dèsign Automation Conference, 1983

[McDermott 78] D. McDermott, "Circuit Design as Problem Solving", Artificial Intelligence and Pattern Recognition in Computer Aided Design, (ed. J. C. Latombe), North-Holland, 1978

[McDermott 82] J. McDermott, "R1: A Rule-Based Configurer of Computer Systems", Artificial Intelligence, 19, 1982

[Medland 86] A. J. Medland, "The Computer-Based Design Process. 1., Engineering design-data processing I.", Kogan Page Ltd, 1986

[Mittal *et al.* 86] S. Mittal, C. L. Dym, and M. Morjaria, "A Knowledge-Based Framework for Design", Proc. of AAAI-86, 1986

[Murthy and Addanki 87] S. Murthy and S. Addanki, "PROMPT: An Innovative Design Tool", Proc. of AAAI 87, 1987

[Nagai 88a] Y. Nagai, "Expert System for Design Problems", Proc. of 6th Symposium on Fifth Generation Computer, (in Japanese), 1988

[Nagai 88b] Y. Nagai, "Towards an Expert System Architecture for Routine Design - Focusing on Constraint Representation and an Application Mechanism for Mechanical Design", Proc. of 3rd International Conference on CAD/CAM, Robotics and Factories of the Future (CARS & FOF '88), 1988

[Nagai 88c] Y. Nagai, "Towards Design Plan Generation for Routine Design Using Knowledge Compilation - Focusing on Constraint Representation and Its Application Mechanism for Mechanical Design", ICOT Technical Memorandum, TM-504, 1988

[Nagasawa 87] I. Nagasawa, "Design Expert System", IPSJ, Vol. 28, No. 2, (in Japanese), 1987

[Nicklaus *et al.* 87] D. J. Nicklaus, S. S. Tong, and C. J. Russo, "ENGENIOUS: A KNOWLEDGE-DIRECTED COMPUTER-AIDED DESIGN SHELL", Proc. of 3rd Conference on Artificial Intelligence Applications, 1987

[Phillips and Rosenfeld 87] R. E. Phillips and L. W. Rosenfeld, "A Knowledge-Based System for Design Automation", ICAD Inc., 1987

[Rinderle 87] J. R. Rinderle, "Implications of Function-Form-Fabrication Relations on Design Decomposition Strategies", Proc. of ASME Computers in Engineering Conference, 1987

[Stallman and Sussman 77] R. M. Stallman and G. L. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", Artificial Intelligence, Vol. 9, 1977

[Stefik 81a] M. Stefik, "Planning with Constraints (MOLGEN: Part 1)", Artificial Intelligence, Vol. 16, 1981

[Stefik 81b] M. Stefik, "Planning and Meta-Planning (MOLGEN: Part 2)", Artificial Intelligence, Vol. 16, 1981

[Struss 87] P. Struss, "Multiple Representation of Structure and Function", in "Expert Systems in Computer-Aided Design (ed. J. Gero), North-Holland, 1987

[Subrahmanyam 86] P. A. Subrahmanyam, "Synapse: An Expert System for VLSI Design", IEEE Computer, July, 1986

[Sussman and Steel 80] G. J. Sussman and G. L. Steel Jr., "CONSTRAINT - A Language for Expressing Almost-Hierarchical Descriptions", Artificial Intelligence, Vol. 14, 1980

[Taki *et al.* 84] K. Taki, M. Yokota, A. Yamamoto, H. Nishikawa, S. Uchida, N. Nakajima, and M. Mitsui, "Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI)", Proc. of International Conference on Fifth Generation Computer Systems, 1984

[Terasaki *et al.* 88] S. Terasaki, Y. Nagai, T. Yokoyama, K. Inoue, E. Horiuchi and H. Taki, "Mechanical Design System Building Tool: MECHANICOT", JSAI, SIG-KBS, (in Japanese), October, 1988, (to-appear)

[Tomiyama and Hagen 87] T. Tomiyama and P. J. W. T. Hagen, "Organizing of Design Knowledge in an Intelligent CAD System", in Expert Systems in Computer-Aided Design (ed. J. Gero), North-Holland, 1987

[Yokoyama 88] T. Yokoyama, "FREEDOM : A Knowledge Representation System for Design Object Modeling", IPSJ, WG-AI, 88-AI-60, (in Japanese), 1988