A GENERAL APPROACH

TO

DESIGN AUTOMATION

A GENERAL APPROACH TO DESIGN AUTOMATION

by

SHUEJUN CHEN, B.Eng. (Mechanical)

A Thesis

Submitted to the School of Graduate Studies

In Partial Fulfillment of the Requirements

For the Degree

Master of Engineering

McMaster University

September 1990

MASTER OF ENGINEERING | McMASTER UNIVERSITY
(Mechanical Engineering) | Hamilton, Ontario

TITLE:	A General Approach to Design Automation
AUTHOR:	Shuejun Chen, B.Eng. (Beijing University of Aeronautics and Astronautics)
SUPERVISOR:	Dr. Hoda A. ElMaraghy Professor and Director Centre for Flexible Manufacturing Research and Development

NUMBER OF PAGES: xiii, 177

ABSTRACT

This thesis developed a domain independent "shell system for routine mechanical design". This shell is used to produce domain specific design systems by simply placing domain–related knowledge into it. A general "design model", which is an informal description of the mechanisms behind the design process, has been implemented. The design model is established based on the "characteristics and mechanisms common in routine mechanical design activities".

By examining particular design examples, it is concluded that the routine design activities have: 1) a "common design procedure" from specification recognition to detailed design; 2) "common mechanisms" to determine parameters and the like; and 3) "common knowledge formats" to express design knowledge. Only "detailed design knowledge" is specific to each domain, but can be represented in common knowledge formats.

The "model" and its implementation, the shell system, describe the design process in four stages: specification development, synthesis, analysis and non–functional considerations. The synthesis achieves rough structural configurations by following the "configuration decomposition approach" which is derived from the well–developed configuration decomposition patterns in the routine design, and which uses function–to–configuration, configuration decomposition and function–checking relations. In the analysis stage, configuration parameters are determined by design relations which are represented by "design slices" written in the form of "basic description elements". The analysis knowledge is organized in a multilevel structure from lower levels of basic description elements, design slices, to upper levels of "design procedures" and "knowledgeable configuration units". Design slices are classified as "solving slices" and "checking slices" responsible respectively for determining parameters and ensuring that checking criteria among parameters are met. A design procedure is a pile of design slices and determines a set of parameters since design relations are used in groups. The uppermost level consists of knowledgeable configuration units. They organize design procedures, parameter sets and configuration decomposition patterns under a configuration. The reasoning process in analysis is decentralized through a number of "interpreters" which handle various tasks such as choosing a design procedure. The non–functional design aspects are considered in the design relations and are incorporated into the analysis.

The shell system provides general design knowledge representation formats and general reasoning mechanisms. It is implemented on a SUN workstation using KEE which provides object–oriented programming and rule reasoning facilities. Connection between design components is dealt with using partial configurations and constraints which define the relationships between configurations and partial configurations involved in a connection. The iteration process caused by dependency among parameters is handled using the failure design procedures, that is, if a checking relation is not satisfied, a failure design procedure is called to modify some parameters at the early design stage. The geometry aspect is implemented parametrically based on an existing feature–based modelling system (IPDM).

Two specific design systems: a cam system and a bolted flange system, have been developed based on the shell. Both accept given specifications, and output configurations with parameters and graphic display. The development process of these two systems is simple and efficient; and design results are satisfactory. These examples illustrated the versatility and effectiveness of the developed approach to routine mechanical engineering design activities.

The major feature of this work is the explicit descriptive style in representing the design knowledge. The domain independent shell approach enhanced by this feature greatly simplifies the development of domain specific knowledge bases.

ACKNOWLEDGEMENTS

The author would like to express his deepest appreciation for the invaluable support provided by his supervisor, Dr. H.A.ElMaraghy.

The financial support from the Mechanical Engineering Department in the form of teaching assistantships and the research scholarship from Dr.H.A.ElMaraghy's research funds are greatly appreciated.

Due thanks to Larry Knoll for developing special wedges in IPDM and Todd Pfaff for improving the display interface. And also, thanks to friends and colleagues for their advice and encouragement.

Special thanks must go to Rebecca Clifford for her valuable help in making this thesis more readable.

This work is dedicated to my wife, whose understanding and patience made it all possible.

TABLE OF CONTENTS

	page
ABSTRACT	iii
ACKNOWLEDGEMENTS	v
TABLE OF CONTENTS	vi
LIST OF FIGURES	x
LIST OF TABLES	xiii
CHAPTER 1 : INTRODUCTION	1
1.1 THE RESEARCH TOPIC	1
1.2 ORIGIN	3
1.2.1 CAD Technique Development	3
1.2.2 Design Activity Study	4
1.3 FEASIBILITY	5
1.4 MOTIVATION	10
1.4.1 Potential Benefits	10
1.4.2 Expert System Technology	11
1.4.3 Current Research Status	12
1.5 THESIS OBJECTIVE, SCOPE AND PREVIEW	14
1.5.1 Objective And Scope	14
1.5.2 Preview	14

1.6 SUMMARY 15

1.7 THESIS ORGANIZATION 16

CHAPTER 2 : LITERATURE SURVEY 17

2.1 SURVEY SCOPE 17

2.2 OVERVIEW 17

2.3 UNDERSTANDING AND MODELLING OF DESIGN PROCESSES 19

2.3.1 Descriptive Models 19

2.3.2 Prescriptive Models 21

2.3.3 "Practice-Oriented" Model 22

2.4 GENERAL COMPUTER TOOLS 24

2.4.1 Conceptual And Configuration Design 24

2.4.2 Geometry Aspect 27

2.4.3 Parametric Design 28

2.4.4 New or Miscellaneous Topics 30

2.5 SURVEY CONCLUSIONS 31

CHAPTER 3 : UNDERSTANDING OF DESIGN ACTIVITIES 34

3.1 PURPOSE AND SCOPE 34

3.2 BASIC STRATEGY 35

3.3 GENERAL CHARACTERISTICS 36

3.4 OBSERVANCE OF A DESK LAMP DESIGN 37

3.5 COMMON CHARACTERISTICS 44

3.6 ANALYSIS OF A CAM DESIGN 46

CHAPTER 4 : GENERAL DESIGN MODEL 50

4.1 THE WHOLE DESIGN PROCEDURE 50

4.2 SPECIFICATION DEVELOPMENT 51

4.3 SYNTHESIS 53

4.4 ANALYSIS 59

4.5 NON–FUNCTIONAL CONSIDERATIONS 67

4.6 SUMMARY 68

CHAPTER 5 : IMPLEMENTATION 69

5.1 CONCEPTUAL IMPLEMENTATION 70

5.1.1 The Whole Design Procedure 70

5.1.2 Specification Development 72

5.1.3 Synthesis 73

5.1.4 Analysis 80

5.2 MACHINE IMPLEMENTATION 91

5.2.1 The Software And Hardware Environment 91

– A Brief Introduction To KEE 92

5.2.2 Description Of Machine Implementation 95

– The Whole Design Procedure and Specification Development 95

– Synthesis 96

– Analysis 97

– Basic Functional Design Units And "IPDM" 97

– User Interface 99

5.2.3 How To Develop A Specific Design System 101

CHAPTER 6 : SPECIFIC DESIGN SYSTEMS 103

6.1 CAM DESIGN SYSTEM 103

6.1.1 Development 103

6.1.2 A Cam Design Example 127

6.1.3 Comments 129

6.2 BOLTED FLANGE DESIGN SYSTEM 129

6.2.1 Development 129

6.2.2 A Cam Design Example 139

6.2.3 Comments 144

CHAPTER 7 : CONCLUSIONS AND DISCUSSION 146

7.1 SUMMARY OF THE THESIS WORK 146

7.2 REVIEW OF CURRENT RESEARCH WORK 147

7.3 CONTRIBUTIONS 147

7.4 COMPARISON WITH SOME RESEARCHERS' WORK 148

7.5 FUTURE DEVELOPMENTS 151

BIBLIOGRAPHY 153

APPENDIX 162

LIST OF FIGURES

Figure		page
1.1	Basic ideas of the research topic	2
1.2	Coverage of domain independence	7
1.3	A cam and a desk lamp	7
1.4	Comparison of two design examples	8
1.5	Research strategy and concrete parts in this thesis	15
2.1	Classification of this research area	19
3.1	The purpose of understanding the design activities	34
3.2	A desk lamp	38
3.3	The decomposition of the desk lamp, with all configurations	40
3.4	A screw connection between the base and the neck	41
3.5	Parameters of the base	42
4.1	Design stages in the whole design process	51
4.2	Specification development	52
4.3	Configuration Decomposition Approach	55
4.4	A portion of synthesis in desk lamp design	56
4.5	A portion of a sample configuration tree	57
4.6	The mechanism of the whole analysis process	62
4.7	The dependency / iteration mechanism	63
4.8	Geometry information of a cube	64
4.9	An organization for the geometry aspect in the analysis	64
5.1	Relations between understanding, model and implementation	70
5.2	Organization of the whole design process — file transfer	71

5.3 The basic “four-step-cycle” of C.D.A. 74
5.4 A portion of a sample configuration tree 79
5.5 Default frame for a geometry feature -- cycloidal wedge 80
5.6 Organization of analysis knowledge representation formats 83
5.7 The KEE programming facilities 92
5.8 Organization of the machine implementation 94
5.9 Basic functional design units library and IPDM 99
5.10 Comparison of a basic design unit file from the design system and IPDM 100
5.11 Procedure to develop a specific design system from shell system 102
6.1 A cam sketch 104
6.2 The configuration tree and the function tree 109
6.3 A “f-to-c” relation (left) and a knowledgeable configuration unit (right) 110
6.4 Portion of cam synthesis knowledge base 112
6.5 Synthesis running windows 113
6.6 A normal design procedure (upper left), a “no.action” procedure (upper right) and the configuration dictionary (lower) 116
6.7 A solving slice 117
6.8 A checking slice 118
6.9 Cam factor table, table header structure and rule structure 124
6.10 Cam analysis knowledge base 125
6.11 Cam analysis running windows 126
6.12 Cam geometry model produced by the IPDM display interface 128
6.13 A flange sketch 131
6.14 Forces acting on an integral flange 133
6.15 Configuration tree and function tree 136

6.16 A “f–to–c” relation (upper right), a c–subc rule (lower) and a subf–f rule (upper left) 137

6.17 Flange synthesis knowledge base(left) and the configuration dictionary(right) 138

6.18 A solving slice 140

6.19 A knowledgeable configuration unit (left) and a checking slice (right) 141

6.20 Flange analysis knowledge base 142

6.21 Flange geometry model produced by the IPDM display interface 143

LIST OF TABLES

Table		page
4.1	Function–to–configuration relation	54
4.2	Function–checking relation	54
4.3	Configuration–decomposition relation	57
4.4	Configuration item in the dictionary	58
4.5	Equation in method	65
4.6	Another method	66
4.7	A more complicated case in two methods	67
6.1	Cam system scope	105
6.2	Cam specification list	106
6.3	Synthesis shell general units	108
6.4	A "c–subc" rule	111
6.5	A "subf–f" rule	111
6.6	A "c–subc" rule with constraints	111
6.7	Analysis shell general units	114
6.8	Parameter nomenclature	114
6.9	Illustration of running process	119
6.10	The cam factor table	123
6.11	Flange system scope	132
6.12	Flange specification list	135

CHAPTER 1 : INTRODUCTION

The sole purpose of this introduction is to give a clear description of the research topic in this thesis by elaborating on its origin or its historical development, explaining its feasibility and anticipating the possible benefits. To further enhance this description, a brief overview of the current research status, the thesis objective and scope and a preview of the thesis work are also presented.

1.1 THE RESEARCH TOPIC

The research topic can be defined as finding a general model of a variety of design activities and, based on this model, developing a computer based tool to automate these design activities.

There are two issues in this topic : 1) modelling design activities in a general manner; 2) developing a computer based tool to automate these activities. A model refers to a theory regarding the mechanisms of the design process which is expressed by formal modelling techniques or informal descriptions. A general model deals with the common mechanisms of a number of design activities, and is also called a "domain independent" model. In this thesis, "design processes" and "design activities" are used interchangeably; similarly, "domain independent" and "general". This domain independence is achieved by capturing common mechanisms from a number of design processes. A computer based tool is a system which facilitates the automation of these processes.

The first step is to find a general model, while the final objective is to implement this model into a computer based tool. The computer based tool is like an "empty box" with the frame constructed from the common mechanisms extracted from various design activities, but excluding those special features linked to any particular domain. Later, by placing domain specifics into this "empty box", a specific system can be developed to automate the design in this domain. This process actually splits each design activity into two parts: common characteristics which are shared with a number of other activities, and special features of this particular activity. Thus, it speeds up the development of design systems and saves the cost. In order to do this, common mechanisms must be extracted from a number of design processes, which can be done by understanding and modelling design processes.

This computer based tool is intended for two kinds of users. One is the "knowledge engineer" who is responsible for placing special domain-related materials into the general frame which is also referred to as the "shell system", to develop specific systems. The other is the "end user" who actually uses specific systems for design (Figure 1.1).

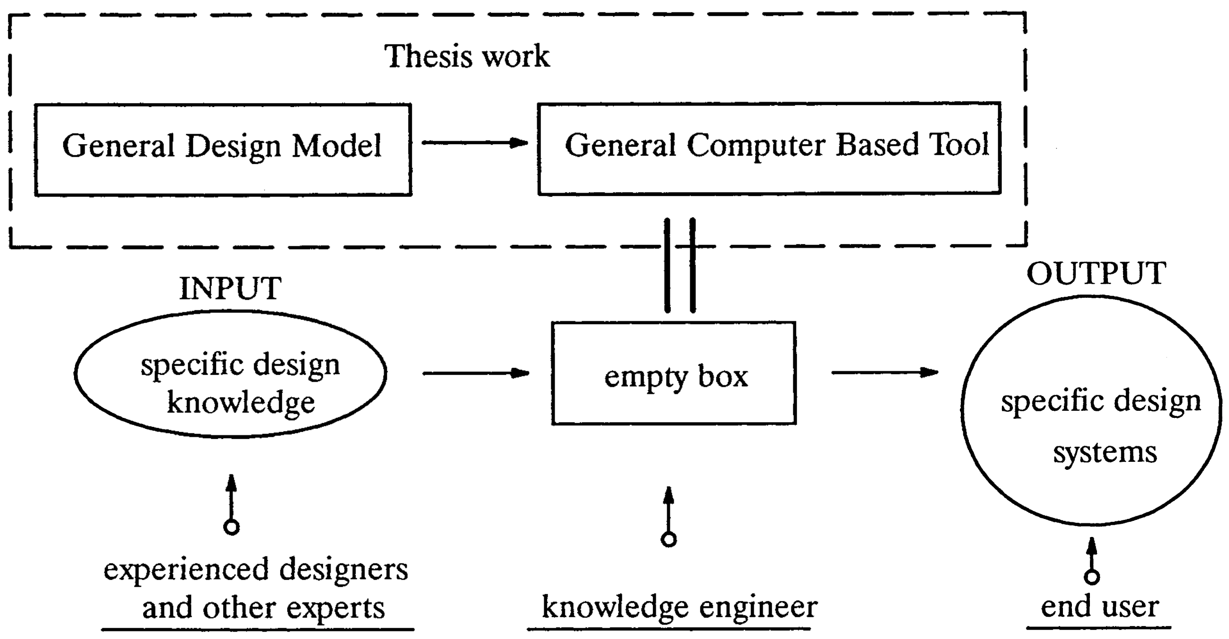


Figure 1.1 : Basic ideas of the research topic

Briefly, this serves to describe the research topic and the basic ideas behind it. The following elaboration on related materials will further explain this research.

1.2 ORIGIN

1.2.1 CAD Technique Development

Although the CAD techniques and analysis methodologies have been developed for years, the design activity itself has not been thoroughly studied from the computer implementation point of view. Most of the present CAD systems provide only assistant tools for human designers to design, such as computer drawings, finite element analysis, optimization and the like. The design activity itself still has to be done by human designers [49].

"Is it possible to go one step further -- asking the computer to design?"

Yes, some CAD systems do design. For example, a punch die system designs according to the specifications given by the user [11]. This system was programmed based on the procedure experienced designers followed in their work. The results achieved by the system were as good as the ones from the human experienced designer; some aspects were even better. But the problem is that these systems have to be developed one by one corresponding to various design domains. This means critical demand for the experienced programmers, and large expenditures of both money and time. Especially, when there are changes in the design processes, these specific design systems have to be modified accordingly. In one word, the development and the modification will cause a severe problem. Thus, another question arises:

"Is it possible to create a general computer based tool for a number of design activities, so that specific systems can be developed simply by filling domain specifics into this tool?", thereby making the development and the modification efficient and economical. Thus, the research goal becomes evident: a general computer based tool based on a general model. The answer to the question posed is "yes", because there does exist domain independence among various design activities [85, 86]. This point will be explained in the later "Feasibility" section.

1.2.2 Design Activity Study

For some time, the topic of whether "Design is a science or an art" has been debated. Some people think that design is an art. It is the intuition of the designers and, thus, it is nearly impossible to study it. Other researchers believe that design is like other science disciplines, and can be studied [17].

Those researchers who believe that design is a science have studied the design activities in various engineering areas such as mechanical, civil, electrical and so on. Some of them worked on a particular design activity, while others focused on the design process common to all areas. Researchers have used various techniques, like protocol study and cognitive model, to create a design theory or methodology. The study of design activity will be described in the next chapter. Here, some ideas about design are presented based on the work of others to give a brief impression of what they have achieved and are doing. However, these ideas do not necessarily represent the stand taken in this thesis.

Design is a transformation or mapping process from the functional domain to the physical domain which satisfies the stated functional requirements within identified constraints. The term, function, refers to the general input/output relationship of a system (or subsystem component) whose purpose is to perform a given task. It describes the expected behavior of a component or a system [27]. Design is a multi–faceted and multi–disciplinary activity. Contemporary theories attempt to formalize strategies for developing design solutions and systematize the design activity [67]. There have also been efforts to develop a generalized set of design axioms to help systematize the design methodology [83]. Designers have always sought to develop systematic approaches to design that allow a step by step progression from qualitative to quantitative phases. It has been widely accepted that a design process follows several steps, from specification recognition, to conceptual design, to detailed design [45].

The design study achieves the understanding of its process and provides theories about how design activities proceed and what characteristics they have. Such understanding and theories should be general to a number of design activities. To date, no universally accepted theories have been developed.

However, this study lays some foundation for the achievement of the general computer based tool, although some researchers originally may not work for this purpose; they may simply pursue the theories about design. It is simply a natural progression whereby the topic arises: modelling the design activities, then creating computer based tools based on the modelling.

Thus far, the research topic has been discussed through two aspects: the design study provides the foundation for modelling and urges the development of computer based tools; and the development of the CAD techniques promotes the design automation research and demands a general design tool, due to the inefficiency and cost of developing specific systems.

Now from the explanation of the origin of the research topic, the basic stand of this thesis is evident; that is, design can be studied. The question now is whether it is possible to achieve the general design model and the tool. The following section deals with this dilemma.

1.3 FEASIBILITY

Design processes exist in various areas. In engineering, there are electrical circuit design, civil structure design, mechanical component and mechanism design, and so on. Certainly, they each have different characteristics. For instance, the circuit design can usually be decomposed into a group of basic standard components, such as diodes and transistors, with standard inputs and outputs. Their design seems to be much different from a motion transmission design, which can be realized by many mechanisms such as gears, cams and linkages. Furthermore, the variation of component attributes is much broader than those of electrical components. This does not mean that the mechanical design process is more sophisticated than the electrical design, it only shows the difference among areas. Despite this, differences also exist among various design processes in one area. A mechanical component design does not share all characteristics of a mechanical system design, which needs a group of experts specializing in several areas; the coordination among them is critical.

The existence of the above differences does not mean that the achievement of common characteristics or that domain independence is impossible; it simply shows the diversity and complexity of design activities. Since they are all design processes, they must have something in common. If it is difficult to identify common elements among all design processes, it is possible to do so within a number of them. For instance, the design processes of standard mechanical products such as gears, cams and linkages share similar procedures, excluding domain specifics. Here, the decomposition idea is used to divide the whole complicated design activity into groups of processes, so that some common characteristics can be extracted from each group.

Here, a very important concept is introduced: "the coverage of domain independence". A domain independent model captures the common characteristics of a certain number of design activities and, thus, covers them. Since there are so many activities, it may be impractical to find a model independent of all of them, either because it is impossible to analyze them within a certain time limit, or they may be so diverse that it is not worth achieving such a general model for the whole area. However, it is certainly possible to achieve a model with certain coverage, within which the model is general. This model contributes a small block to a more general and broader model for all design processes. In this way, a number of these small models with their coverages may cover the overall design area. This idea is illustrated in Figure 1.2, in which the whole design area W is divided into small areas A, B, C, D and E and covered separately by model A, B, C, D and E. For instance, when a lathe is to be designed, the task is handled by an overall model which divides the design into a number of smaller domain–oriented tasks and distributes them to corresponding models, among which there will be one model for the mechanical component design.

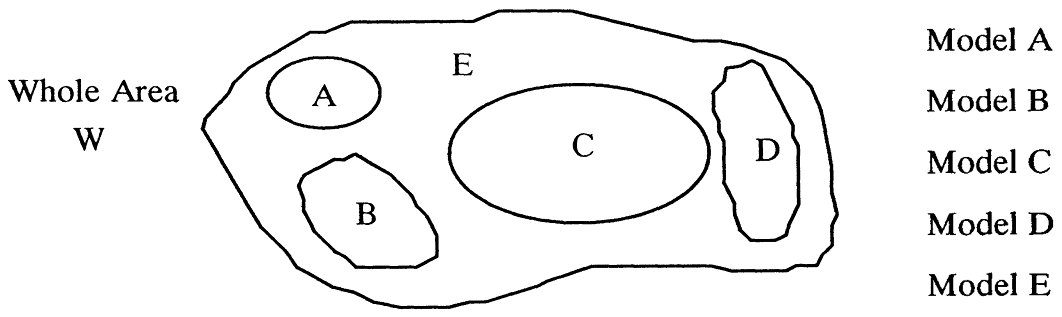


Figure 1.2 : Coverage of domain independence

To demonstrate the possibility of domain independence, the design processes of two examples:

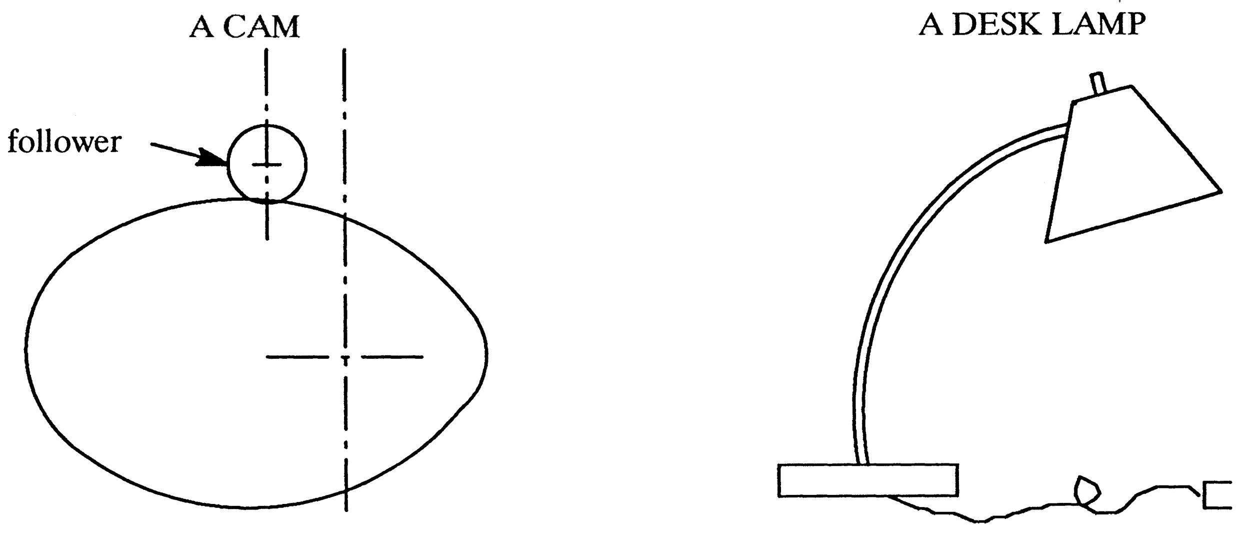


Figure 1.3 : A cam and a desk lamp

a desk lamp and a cam (Figure 1.3) are listed in Figure 1.4. In order to demonstrate the common characteristics, two processes are listed under four categories: "design procedure", "how to design", "design knowledge format" and "detailed design knowledge". Individually, they represent the steps the designers follow; the way the designers achieve their results; the representation formats for specific knowledge; and the specifics related to the particular example.

A Cam	*A Desk Lamp*
Step 1: configuration design	**Step 1: configuration design**
. design procedure:	. design procedure:
– consider functions and specifications to generate the initial conceptual product –– cam.	– consider functions and specifications to generate the initial conceptual product –– desk lamp.
– obtain all configurations.	– obtain all configurations.
. how to design:	. how to design:
– choose a product (cam) from function, then decompose it into components.	– choose a product (desk lamp) from function, then decompose it into components.
– or choose a cam from a similar design, then change it to suit new specifications.	– or choose a similar desk lamp design and change it to meet new specifications.
. design knowledge format:	. design knowledge format:
– strings for functions and specifications. For example, (initial–function , rotation–translation), (pressure–angle , <30.0). The general format is (function/specification, value).	– strings for functions and specifications. For example, (initial–function , lighting–a–desk), (volt , 110) etc. The general format is (function/specification, value).
. detailed design knowledge:	. detailed design knowledge:
– a cam must perform motion transmission: rotation to translation.	– a desk lamp must light a desk.
– no steps in acceleration, steps in jerk are acceptable.	– under 110 volts.
– pressure angle is less than 30.	– etc.

Figure 1.4: Comparison of two design examples

Step 2: detailed design	**Step 2: detailed design**
. design procedure:	. design procedure:
– obtain values for all configurations' attributes.	– obtain values for all configurations' attributes.
. how to design: achieve attributes by design relations:	. how to design: achieve attributes by design relations:
– iteration procedure;	– iteration procedure.
– dependency among attributes;	– dependency.
– design every configuration;	– design every configuration.
– design every attribute.	– design every attribute.
. design knowledge format	. design knowledge format
– design relations for attributes.	– design relations for attributes.
– a collection of basic elements such as calculation, looking–up–a–table, etc.	– a collection of basic elements such as calculation, looking–up–a–table, etc.
. detailed design knowledge	. detailed design knowledge
– pressure–angle = (a formula).	– support base weight should balance the whole lamp.
– cam factors (a data table).	– etc.
– (for more, refer to section 3.6).	

Figure 1.4 (cont'd): Comparison of two design examples

By examining these two examples, it can be concluded that the "design procedure", "how to design" and "design knowledge format" are common to both. Only the "detailed design knowledge" is specific to each particular example, but can be described by the "design knowledge format". A model for these two design processes is achievable.

Even broadly, the design processes of standard mechanical components and mechanisms classified under the "routine mechanical design" category have much in common: they all follow a certain procedure; and comply with codes, standards and formulas which are various but sharing similar formats. In other words, the mechanisms underlying these activities are similar. Therefore, a model for the routine mechanical design activities is feasible. Nevertheless, "a certain range" is still imposed, because of the impracticality of examining all of them, though it is probable that these activities can be covered by one model or by the continuing development of this model. Hence, the concrete objective of this thesis is brought out, that is: finding a general model for a certain range of the routine mechanical design activities, and based on it, developing a computer based tool.

Thus far, by introducing the coverage of domain independence and examining two particular design examples, the feasibility and the significance of the domain independent model with certain coverage has been established. Moreover, the objective of this thesis has been specified within the research topic. The next question is why this topic has been studied and why this thesis research was launched.

1.4 MOTIVATION

1.4.1 Potential Benefits

Some people may argue that even though a model for a certain range of design processes is achievable and specific design systems can be developed efficiently, the results from these systems are at most as good as those from designers who contribute their knowledge to these systems. This is not the case, as will be explained later.

What is then the use of this general model and computer based tool? To answer this question, the potential benefits from this topic have to be analyzed from different perspectives, according to various kinds of users.

For experienced designers, the specific design system can provide considerations of other aspects related to design, such as manufacturability, cost, maintenance and so on, with which these designers may not be familiar. Because the specific design systems are developed from the general computer based tool, it is very easy to incorporate multiple experts' knowledge into the design stage, and not only multiple designers, but also experts at other stages in the product life cycle. As a result, the design quality is improved (and the earlier question has been answered). Moreover, this kind of general tool can be integrated with current CAD packages conveniently, thus simplifying the process of utilizing available CAD software.

Considering the spreading of design knowledge, the available number of experienced designers is usually limited. If such a computer based tool is available, it will simplify the process of spreading the experienced designers' knowledge. Non–experts can benefit from this. This is also very helpful in teaching novice designers.

From the human resource aspect, if the mature design problems can be solved by these systems, many engineers can be relieved from this basically repetitive work and concentrate on more creative design.

1.4.2 Expert System Technology

People may argue that commercial expert system shells should be utilized as the general computer based tools for the design process. The reason of not using them is that these expert system shells do not fit the design activities very well [3].

Most of the commercial shells only provide knowledge representation methods for the classification problems, characterized as having premises, conclusions and relations connecting them. The design processes do not belong to this category. Thus, some of their features can not be covered. For example: the whole design process is a transformation from functions to physical parts, identified as a progression of stages; the design knowledge is so diverse that the conventional knowledge representation, typified as the rule representation, is not powerful enough to handle it; the design process is iterative, there is dependency among attributes and stages; geometry aspect plays a very important role in design which does not happen in classification problems. Therefore, further work on the design process representation is certainly necessary. However, this does not mean that the results from the expert system research can not be utilized at all. In fact, the present expert system shells can be used as the supporting software, which is an efficient development method utilizing the useful parts and developing those parts not available from the commercial shells.

Besides the supporting tool, the ideas from the expert system technology also influence this research, since design is the performance of human knowledge, which is just what expert system is dealing with. On the other hand, the work under this research topic can certainly enrich and broaden the original expert system area. From this point of view, this work can be viewed as a branch of the expert system research, which covers a new domain -- the design process. Benefits inherited from the expert system work include knowledge representation methods developed so far, typically the rule and its reasoning methods, and an concept -- the "shell system", which plays a key role in forming the general computer based tool idea in this thesis.

1.4.3 Current Research Status

The current literature shows that the work on this topic is still in its infancy. Two critical problems are: understanding and modelling the design process; and developing a computer based tool.

Researchers who believe that better understanding and modelling can lead to a better computer based tool have been using various techniques to pursue a model truly reflecting the design activities. So far, no generally accepted theories have been achieved yet.

In the computer based tool work, the focus is mainly on three problems: the conceptual and configuration design, the parametric design, and the geometry aspect. The difficulties in the conceptual and configuration design are how to achieve configurations from functions and how to evaluate configurations without values being assigned. Two kinds of design processes have to be distinguished and dealt with in different strategies: the "new design" and "mature design". Usually, a design task is either a mature design which more or less follows some lines or patterns developed by experienced designers (this point is the basis of a part of this thesis work), or a small portion of new design which requires the designer's intuition. These problems have not been well studied.

Some researchers have realized that "features" play an important role in the geometry design process [28]. Other researchers have been trying to find the mechanisms underlying the spatial and geometric reasoning. Thus far, no satisfactory results have been achieved yet.

In the parametric design, researchers have developed iterative models using optimization algorithms, dependency strategies based on graph theory, and so on. Although, the work in this branch is the most advanced of all in this area, one apparent limitation is that it is not easy or natural to express the design knowledge in the current representation schemes. Thus, it makes the spreading of specific design systems difficult, which happens to be one of the major purposes of this research.

Most researchers have studied the above problems separately, and thus, their results may not cover all aspects of the design process.

Among the various engineering design activities, the electrical circuit design is the most advanced, probably because of the limited number of components with relatively obvious function–configuration relations. The work in the mechanical design is just beginning, caused by some domain–related difficulties such as: the broad selection of configurations, the complicated function–configuration relations, and so on.

In summary, further work on the above topics about mechanical design process is certainly necessary.

1.5 THESIS OBJECTIVE, SCOPE AND PREVIEW

1.5.1 Objective And Scope

As stated earlier, the objective of this thesis is to find a general model for a certain range of routine mechanical component and mechanism design activities, and to develop an experimental shell system.

The "model" means an informal description of the mechanisms behind the design process and the representation formats for design knowledge.

The design activities studied here are those in the "mechanical design" domain. They are limited to the "routine design" or the mature design. Within the routine mechanical design activities, the scope is further narrowed down to the "component and mechanism design". Although the model can potentially cover all routine component and mechanism design activities, the "certain range" is imposed because it is obviously impossible to practise every activity. Moreover, the model and the shell system are devised in such a way that further development can be easily made.

The term "experimental shell system" is used because it is difficult to achieve a complete commercial system at this stage, and the purpose is just to show the feasibility of the ideas presented here.

1.5.2 Preview

The work consists of two parts corresponding to the objective: a general model and a shell system. The understanding is based on an informal observance of some design examples, from which common characteristics are extracted. Based on this understanding, certain theories about the design are formed which constitute the model. Two major parts of the model and the shell are the "synthesis" and "analysis": one follows the "Configuration Decomposition Approach" which basically complies with the well-developed configuration decomposition patterns in the routine design; the other emphasizes the explicit descriptive style in representing the design knowledge by creating a group of "basic description elements". This shell is first created conceptually without considering any specific computer implementation environment, and then implemented on a specific machine, in order to maintain the machine-independence. Later on, two domain specific design systems are developed to prove the feasibility of the approach in this thesis (Figure 1.5 illustrates all parts in this work).

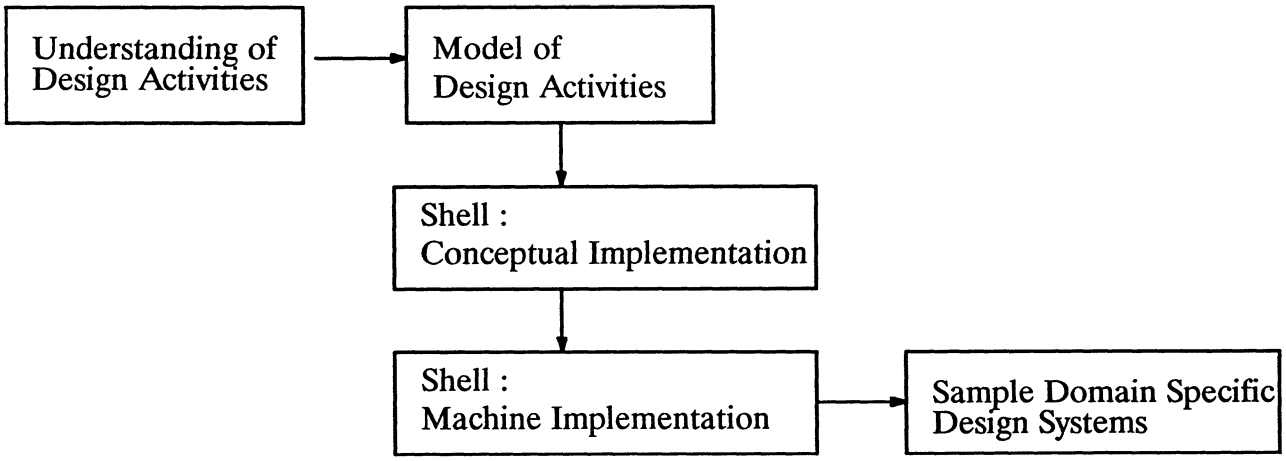

Understanding of Design Activities
Model of Design Activities
Shell : Conceptual Implementation
Shell : Machine Implementation
Sample Domain Specific Design Systems

Figure 1.5 : Research strategy and concrete parts in this thesis

1.6 SUMMARY

In this chapter, the thesis research topic was stated first. Its origin was explained by reviewing the development of CAD techniques and the study of design activity. Then, the feasibility of this topic was established by introducing "domain independence with certain coverage". The motivation for this research was exposed from the perspectives of potential benefits, shortfalls of current expert system shells and the demand of current research status. Then, the objective and scope of the thesis work were defined, followed by a preview of the work.

1.7 THESIS ORGANIZATION

Chapter 1: “Introduction” is the current chapter.

Chapter 2: “Literature Survey” gives an overview of current literature on the topic.

Chapter 3: “Understanding of Design Activities” analyses the design activities based on some examples.

Chapter 4: “General Design Model” creates a domain independent model based on the understanding.

Chapter 5: “Implementation” executes the model and achieves an experimental shell system.

Chapter 6: “Specific Design Systems” develops two domain specific design systems based on the shell: the cam system and the flange system.

Chapter 7: “Conclusions and Discussion” summarizes the contributions from this thesis, as well as the potentials for future development.

References are listed in the Bibliography.

The Appendix includes a list of knowledge base items and design examples from the cam system and the flange system.

CHAPTER 2 : LITERATURE SURVEY

2.1 SURVEY SCOPE

This survey only covers the literature about this thesis topic --- the modelling of design processes and the computer based tool to automate these processes. It is a summary of the reference work during the whole research period. Effort has been made to get as large coverage of this area as possible.

The survey is limited to the mechanical engineering domain, which may include design of components, mechanisms, machines, structures, and so on. Research in other engineering domains is cited only when it has influence on or is related to the work in the mechanical domain.

This survey is only concerned with the domain independent work, which means that the work in one design process can be extended to other design processes. Domain dependent work is not cited, although some work is very good.

2.2 OVERVIEW

There are two major branches in this area: 1) the understanding and modelling of design processes; 2) the computer based tools based on the model to develop domain specific design systems. While the ultimate purpose is to create a domain independent computer based tool, the nature of design or how designers design has to be understood to some extent. Then, based on this understanding, the development of the computer based tool is possible.

The design process is very sophisticated, and its understanding and modelling has been a challenge to researchers. This branch can be classified by research approaches developed so far. Some researchers use various techniques to create a descriptive model of the design processes. They believe that better understanding results from careful study and accurate perception of the design processes and, thus, will lead to better modelling and more satisfactory computer based tools. Contrary to studying what characteristics the design process "has", some people prescribe how the design process "ought" to proceed and what attributes the design artifact "ought" to have. Other researchers are much more realistic. They believe that the design is so complicated that an accurate understanding may not be practical in a short term. Considering the final purpose -- a computer based tool, a workable understanding (which may or may not be accurate) may serve the purpose well, so that they can concentrate on the computer based tool, and further improvements can be made according to its performance. Thus, it brings out three sub-branches accordingly: descriptive model, prescriptive model and "practice-oriented" model.

Under the computer based tool, there are four sub-branches: 1) the conceptual and configuration design; 2) the parametric design; 3) geometry aspect in design; and 4) some miscellaneous research topics. They are classified according to the stages in a design process or their features. The conceptual design achieves rough structural information about products from functional requirements, while the configuration design is slightly different from the conceptual design in the sense that functions are not mainly considered. The parametric design acquires the detailed structure. The geometry aspect, along with the parametric aspect, are two fundamental features in design. It distinguishes design from other types of problems. Finally, some new research topics are emerging, such as the study of distributed design (Figure 2.1 illustrates this classification).

The survey briefly goes through the "understanding and modelling" branch. Then the

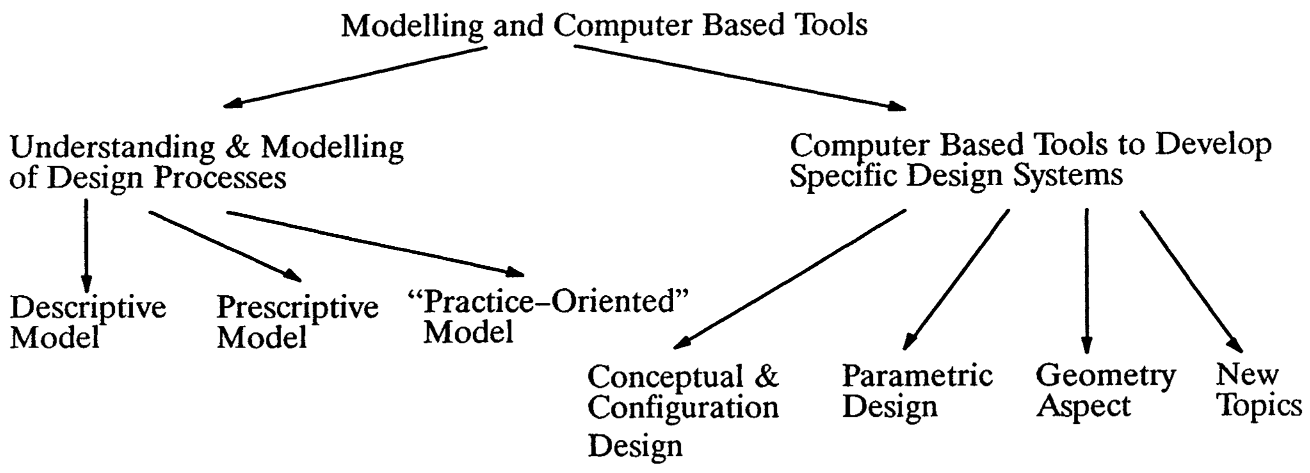

Modelling and Computer Based Tools
Understanding & Modelling of Design Processes
Computer Based Tools to Develop Specific Design Systems
Descriptive Model
Prescriptive Model
"Practice-Oriented" Model
Conceptual & Configuration Design
Parametric Design
Geometry Aspect
New Topics

Figure 2.1 : Classification of the research area

"computer based tool" branch is mainly focused with citations of various researchers' work. Some researchers work may appear in several branches, because these branches are not absolutely mutually exclusive. Finally, achievements and potential improvements in this area are summarized as the foundation of this thesis work.

2.3 UNDERSTANDING AND MODELLING OF DESIGN PROCESSES

All researchers agree that the understanding and the modelling of the design process is the first step toward the creation of the general computer based tool. However, they have different opinions and methods in reaching this understanding and modelling.

Since this thesis takes the "practice-oriented" stand in the understanding and modelling, and focuses on the computer based tool development, the survey for this branch is not a detailed one. The following text outlines various researchers' work in a brief manner. Finger and Dixon [30] have done an excellent survey in this area and their paper being cited in several places.

2.3.1 Descriptive Models

Many researchers are working on a descriptive model of the design process. They have been studying how human designers create designs; that is, what processes, strategies, and problem solving methods are involved. Most of the research is based on techniques from artificial intelligence such as protocol analysis in which data is systematically gathered from human subjects. The work can be divided into two categories: one that gathers data on how designers design and the other that builds models of the cognitive process [30].

One technique used by some researchers is Protocol Studies by individual designers. In a design protocol, the actions of a person performing a design task are recorded as the design evolves. Usually, the designer is encouraged to think aloud and is questioned when information seems to be incomplete.

Most of the protocol studies have been done during the preliminary design stage. One of the major criticisms of design protocols is that a designer's words cannot reveal those processes that are inherently nonverbal, for example, geometric reasoning. Moreover, the requirement to verbalize may interfere with the design process itself. Finally, all protocol studies must address the problem that even though subjects may not have any reason to withhold information, they may do so unconsciously. All of these factors must be taken into account when studying the results of the design protocols.

Ullman and Dietterich [88] have performed such a study of novice and expert designers designing mass–produced and one–off products. They attempted to improve the efficiency of the mechanical design process, and to use their results to support the development of computer based tools. One of their conclusions is that designers pursue a single design concept rather than generate new alternatives.

Another technique is the Cognitive Modelling [1]. The goal of much of the research in cognitive science is to build computer–based models that describe, simulate, or emulate the skills that humans use as they solve problems. A cognitive model describes the processes that underlie the set of behaviors that constitute a skill. For example, a cognitive model could be created to describe the skill of remembering a name. The model is specified as a set of mechanisms with defined functionality; each mechanism is described as a process that can transform classes of input into classes of output. The model also specifies the interactions among the mechanisms. Because the model describes a cognitive system at the level of its functional mechanisms, it generates explanations and predictions about the skill being studied [30]. Developing cognitive models to support the process of design is a relatively recent research topic. Few papers have yet been published.

Researchers have been studying the design process by Case Studies, some of which have been performed on large design projects. Some interesting results have been achieved from their observance. This is a common-sense approach to studying the design process: studying a problem from its original status. Their research work varies in the sense of formality and issues addressed in the design process. After all, the model generated from this observance more or less depends on researchers' opinions.

Many descriptive models exist which are not based on formal observance of the design process, but make intuitive sense to many designers.

To date, much of the work has focused on generating hypotheses based on observances without designing experiments to test these hypotheses.

2.3.2 Prescriptive Models

The Prescriptive Model prescribes what mechanisms and characteristics the design process should have. Prescriptive models can be divided into two categories: those that prescribe how the design process ought to proceed, and those that prescribe the attributes that the design artifact ought to have. Some of these prescriptive models can be found from design textbooks [30].

Instead of studying what the design process "is", some researchers focus on the process the designers "ought to be following", described as the Canonical Design Process. A standard description of what the design process should be recurs in much of the design literature, especially in design textbooks[4, 45]. Some examples are:

• The design can be classified as three classes: 1) creative design; 2) innovative design; and 3) routine design.

• The design process is an iterating progression through the following stages: recognition of need; specification of requirement; concept formulation and concept selection; embodiment of design details; production, sales and maintenance.

• A design is strongly influenced by the life style, training, and experience of designers.

Some researchers use Morphological Analysis [67] to develop design methodologies for the design process. This technique is a highly evolved methodology prescribed to generate and select alternatives. The other aspect of prescriptive model is the prescription of attributes the design artifact ought to have, instead of describing the procedure by which the design artifact should be generated. Some researchers have created the axiomatic systems to describe the attributes that the designed artifact should have as opposed to describing the process by which the design should be generated [83].

One critical problem about the prescriptive models is the difficulty in verifying them. For instance, the Canonical Design Process has existed for many years, but there is no way to verify if this prescription is appropriate. Maybe the descriptive model can eventually show the exact procedure designers follow. Some researchers studied some design cases in practice and they pointed out that the real design procedure does not follow exactly those lines prescribed above. But it is hard to say whether it is because of the systematization of designers or the unrealistic nature of the above theory.

2.3.3 "Practice–Oriented" Model

The work in the descriptive and the prescriptive models has shown that it may not be feasible to reach any accurate modelling in a short term. With the final purpose of this research in mind, the "practice–oriented" model focuses on the computer based tool implementation instead of the accurate modelling. The basic idea is that a model of the design process is developed based on whatever theories or observances that reflect the design process to some extent. Then, a computer based tool is developed based on this model, and at the same time, the model can be improved according to the computer based tool's performance. This iterative process continues until the computer based tool performs satisfactorily, and a workable model is achieved, although it may not be a "true" reflection of the design process. This idea lies behind the well–known fact that the direct imitation of natural mechanisms or rules may not necessarily create better results compared with the modified imitation of them. For example, the aeroplane was invented not by directly imitating birds' vibrant flying wings, but as a modified imitation: "fixed wings".

Based on the above idea, the understanding and modelling is not the major focus, while the computer based tool plays a major role. Therefore, there could be various "practice–oriented" models depending on individual researchers. It might be more appropriate to use the term, the "practice–oriented" approach, that is, researchers use this approach to develop their own models.

The development of the "practice–oriented" model may benefit from or be influenced by descriptive or prescriptive models. But the connection between them is not mandatory. It can also be based on the informal observance, researchers experience and opinions.

On the other hand, the "practice–oriented" model may not be the "accurate" understanding and modelling of the design processes, but it can certainly enrich the research in the descriptive and prescriptive models, if the computer based tool performs satisfactorily.

The approach used in this thesis can be classified under this category. Briefly, it can be described as:

- informal observance and analysis of specific design examples;
- influenced by prescriptive models from design theories in textbooks or other documents;
- benefited from the author's own design experience.

Many researchers are working in this branch. As a matter of fact, most of the research work on the computer based tool can be classified under this branch. This is the matter referred to in the next section.

2.4 GENERAL COMPUTER BASED TOOLS

The general computer based tool for developing specific design systems is the ultimate purpose of this research. Although the approaches of generating models vary, the basic problems are the same, which are mainly the conceptual and configuration design, and the parametric design.

The conceptual design and the configuration design are put under the same category, despite their differences in considering functional requirements, because they both achieve configurations, and the research in these two areas sometimes is difficult to separate. At this stage, the configuration attributes have not been assigned values.

The parametric design assigns values to those attributes of configurations achieved at the conceptual and configuration design stage.

The geometry aspect is typical of design processes. Since its characteristics are quite distinct from those of the parametric aspect, different techniques may be required for it.

Some new research problems are at their initial stage, not very well classified. They are listed under a separate branch.

2.4.1 The Conceptual and Configuration Design

In the conceptual and configuration design, functional requirements are transformed into physical configurations with a defined set of attributes, but with no particular values assigned. For example, if the expected function is to light an office desk, then a possible embodiment can be a desk lamp. Research in this area can be placed in two categories: development of an assembly from a set of standard components (e.g., gears, shafts, bearings, and motors); and development of a non–standard form (e.g., an extrusion, bracket, and truss) by redesign or directly from the functional requirements.

Freeman and Newell [32] launched an earlier study of the relationship between functions and configurations (structures) in this area. Their "design" was in a general sense. They developed a model with two stages, attempting to capture the human functional reasoning process.

Rinderle, et al. [73] has been working on the design of assemblies from components. Emphasis is placed on the function and configuration relations. They use bond graphs to represent configurations in a network of parameters (nodes) and constraints (links). Parameters include both design variables and behavioral characteristics. Constraints express physical principles, spatial relations, specified requirements, and material limitations. The network can be used to determine the important form–function relationships implicit in the configurations, and thus to evaluate the configurations. They further developed a transformational approach to design using a bond graph grammar.

Maher, Sriram, Fenves, et al. [53, 54, 72 & 80] have developed three dimensional structure design systems: HI–RISE and ALL–RISE. The programs generate different structures, made up of standard structural subsystems, based on user defined constraints of size and applied load. The rules guiding the generation are based both on heuristic knowledge and knowledge about basic physical principles.

Struss [82] has created a model to represent the function and structure relationships. Based on this model, configurations could be generated to satisfy specific functions or behaviors. His system provides for connectivity among components, although he does not address the geometric and spatial relationship. His representation points to ways of providing designers with levels of abstraction.

Ullrich and Seering [90] define conceptual design as the transformations from functional or behavioral requirements to structural descriptions. They describe a program which, given a functional description, creates new mechanical fasteners from novel combinations of existing designs. In this work, the embodiment is given; that is, the system can design only something with a drive, head, body, tail, and tip. They have extended this approach to dynamic systems and have created a system that generates a schematic description of functional components which meet the behavioral specifications.

Schmekel [75] has studied the representation of functional models (functional objects) and the design solutions (part objects), and the derivation of design solutions from functional models. By maintaining the independence between functional requirements, a design solution can be derived through many–to–many function objects to part objects relations.

Shah [76, 77] presented his work on structural configuration design, and discussed the need for formal synthesis of structural parts rather than arbitrary selection at the preliminary design stage. He has developed a method using a shape algebra to generate structures of slender and polygonal elements.

Duffey [24] has developed a system for the automated design of non–standard extruded shapes of beams. The system accepts problem specifications and produces a configuration and installation automatically.

Joskowicz [42] also addressed the function and shape relations. He proposed a two–stage algorithm for the qualitative analysis of kinematic mechanisms; first analyzing the possible relative motion from the configurations to obtain functional description, then generating a motion diagram using constraint propagation and label inference techniques.

Lai and Wilson [47] have developed an English–syntax language FDL to describe the structure and function relations of the mechanical elements, which provides basic representation for further incorporation and application of design knowledge bases.

2.4.2 Geometry Aspect

Geometry is the typical feature in design [78]. Some people argue that the design process followed by designers may be a non–verbal process, especially at the preliminary design stage. In other words, designers can express their design knowledge geometrically. Unfortunately, this process has not been well understood and the work is just beginning. The research in this branch could be placed under the "conceptual and configuration design" category. The reason for not doing this is that the geometry aspect in this thesis leans more towards the spatial and geometric reasoning without considering functions. It has unique features and, thus, requires special techniques and attention.

One fact that has been realized by many researchers is that "features" can play an important role in the geometric reasoning or design. This research can benefit from the work in feature extraction research [16, 69, 84 & 87].

Dixon's group [15, 21, 23 & 52] has developed an approach to geometry design called "Design with Features". Basically, they provide a feature library, so designers can use these features to design interactively. They are continuing their work toward the incorporation of design knowledge.

Fenves and Baker [29] proposed a geometry representation for structural design which uses shape grammars as a representation to perform the spatial and functional reasoning. The representation needs to serve all disciplines involved in the design process, where different semantics of each discipline are associated with the same spatial information about design objects. The representation is demonstrated in the building design environment.

Nevill, et al. [63 64] discussed the spatial and geometric reasoning in the mechanical design activities. They selected the preliminary design of configurations for supporting inspection or high precision machining. Multiple level representation was used in a top down abstract refinement model. Relations between objects are represented in semantic terms and high order features facilitate the spatial reasoning.

2.4.3 Parametric Design

At the parametric design stage, the structure or attributes of configurations have been obtained by the conceptual and configuration design. Parametric design is then the process of determining attribute values, which can be numeric, or non–numerical (e.g., a material choice or a motor type).

One point should be clarified that the optimization technique can be used to achieve parameters once the criterion functions can be set up. However, there is a fundamental difference between the optimization technique and the expert system. One is data–oriented, the other is knowledge–oriented. Other differences are: 1) new design strategies can be easily coded into the expert design system, while an optimization model may need major changes to incorporate the new design cases; 2) most optimization models are limited in their ability to handle discrete variables, which are involved in many design problems. In conclusion, these two techniques are developed to handle two kinds of problems. They are not excluding each other. Expert system can provide a descriptive approach to design, and thus, is useful at achieving the initial solutions. The optimization technique supplies the well–established theories to optimize a set of variables, therefore, it can be used together with expert design system to optimize the initial solutions.

Dixon [18, 19, 20, 37, 57, 79 & 94] is leading a group at the University of Massachusetts, mainly along two lines: parametric design and the geometric aspect in design. In their parametric design work, they have developed a parametric design model called DOMINIC that uses a method based on iterative redesign. DOMINIC uses a hill–climbing algorithm that guides redesign by using explicit, but domain–independent, knowledge of dependencies between design variables. Meta–control [66] has been implemented in their later development, which enables the program to monitor its progress in order to select more productive strategies from a library of strategies. This model has been demonstrated in a number of design domains, including V–belts, rectangular beams and extruded heat sinks.

Brown and Chandrasekaran [5, 6, 7, 8, 9 & 10] have been working on the computer based modelling of design processes. They have developed a language, DSPL, to model the routine design process in a top down layered fashion. Specialists choose from existing plans, make commitments and then instruct lower level specialists to refine the plan. Their work has a computer science research flavor, and maybe due to that, they have done excellent work advocating the descriptive style in representing the design knowledge.

Langrana, et al. [48, 70] are developing a domain independent parametric design model, DPMED, which is based on the model VEXED [81] for circuit design. They use a hill–climbing algorithm for parametric design. It evaluates its performance within a design domain based on information about the goals and the design criteria, along with a dependency graph based on analysis equations. They tested their work on gear pairs design.

Mittal, et al. [58, 59 & 60] chose the design of paper transport mechanisms in a copier as their example for developing a domain independent approach. They have produced a system, PRIDE, which uses a knowledge base to generate, evaluate, and redesign configurations of rollers to guide the paper along a smooth path based on design constraints. They structure design knowledge into plans and use "advice" for any modifications to the previous design, which is derived from the evaluation information. This evaluation is important in reducing the amount of design space searched in order to achieve an acceptable design.

Agogino, et al. [2, 13] have applied monotonicity to parametric design to facilitate the design solution once evaluation criteria and constraints are set up. This kind of qualitative analysis has also been used by other researchers, and shows interesting results.

Ishii and Barkan [40, 41] have produced a framework in the mechanical design, representing relations between design variables and performance levels in a rule-based format. This framework facilitates the reasoning process in interactive design and handling dependencies. They have also studied the design compatibility in a mechanical system using fuzzy theory.

Some researchers studied the incorporation of the optimization and knowledge engineering techniques. Gero and Balachandran [33] compared both techniques in the design process and presented a prototype system. Chieng and Hoeltzel [12] proposed a hybrid approach to the mechanical component design using both techniques.

2.4.4 New or Miscellaneous Topics

Some miscellaneous research topics are emerging. One of them is the distributed design. Most large product designs involve the integration of components and sub-assemblies into larger assemblies. To keep size and complexity at manageable levels, they are often designed independently. But complete independence is rare. Interaction among sub-assemblies is inevitable and, thus, the solving of distributed problems arises. Verrilli, Meunier, et al. [91] described a computer program for hierarchical distributed problem solving that was based on the iterative re–specification. Zarefar et al. [95] have developed their parallel axis gear drive system called PAGES. The whole product is divided and designed by several subsystems. The interaction between them is controlled by a "system coordinator". Lu, et al. [51] studied the distributed design in product development using a simultaneous engineering concept.

As the research in this area continues, the requirement for a taxonomy of design problems emerges. Dixon, et al. [22] proposed a taxonomy for the domain of mechanical components and assemblies. Their taxonomy defines a problem in terms of six initial and final states of knowledge. Ullman [89] also proposed a taxonomy for mechanical design. His taxonomy was developed to form a basis for classification of all mechanical design research branches ranging from specific programmes using AI techniques, to general computer based tools and cognitive models.

Some researchers have studied the whole design process in a general way [44]. They have proposed two models of the design process: decompositional and transformational models. Their research delivered an important message that the whole design process could not be forgotten while most researchers were focusing on separate stages. Eventually, a model of the whole design process with sound theoretical foundation is needed.

Factors at other stages in a product life cycle should also be addressed in this design research. The ideal situation is that those factors are considered at the early design stage. Kroll, et al. [46] developed a knowledge based system to assist engineers in the process of designing products for easier assembly. They did their research at the conceptual design stage, since at this stage the whole structure of the product is being considered. They presented the product representation and design for assembly methodology.

As stated earlier, there are the creative design process and the innovative design process. Most of the research in this area has not considerably addressed these two design processes, because of their complexity. However, some researchers are trying to capture the mechanisms of creation in the human design processes. They argue that design is about creation and, thus, it is essential to understand its process [25, 38 & 62]. So far, this is a widely open area of future research.

2.5 SURVEY CONCLUSIONS

The work in the understanding and modelling branch has shown some interesting results, as well as difficulties. No generally accepted theories or models have been created. The computer based tool work has achieved some successful first–step results. Among them, the parametric design is the most mature, though no single theoretical approach has yet evolved. The configuration and conceptual design is starting to appear. The major issue here is the understanding of the links between functions and configurations.

The following summary of achievements and potential improvements is solely about the work on the computer based tool.

1. Some fundamental research work in the configuration and conceptual design has been started. It has been realized that the main issue is the function and configuration relation. Various techniques have been developed or used. The research in this branch is in its infancy.
2. The unique characteristics of the geometric design process have been realized and researchers have launched their work in this area. Features have been recognized as playing an important role in achieving appropriate approaches in this research. Some initial results have been obtained in the "pure" geometric and spatial reasoning.
   As the case in the conceptual and configuration design, the geometry aspect is not well understood. Some techniques may be needed if some breakthrough is to be expected.
3. Some successful results have been achieved in parametric design. The iterative model of

design processes has been realized through different perspectives by various researchers. Dependency among design parameters has been studied. The research work in this aspect has been demonstrated in various design domains.

Descriptive representation in parametric design is very important. The purpose of facilitating the spreading of design knowledge can be well served only when the design knowledge can be expressed descriptively and explicitly. More effort should be directed to this issue.

4. The whole design process has not been paid enough attention. Most researchers addressed those problems or stages individually. Some problems might not be realized if they were studied separately.
5. Some new research topics have been initiated, among them are the distributed design and the taxonomy of the design problems. Further research work is certainly needed.

CHAPTER 3 : UNDERSTANDING OF DESIGN ACTIVITIES

The understanding of design activities is the foundation for the model and the computer based tool. This understanding is so interesting that it could be a research topic in itself, as the dispute has continued over the past decades as to whether design is a science or an art.

A brief understanding is conducted in this chapter. First, the purpose of such an understanding is explained, with its scope derived from the thesis objective. The basic strategy for it is determined after reviewing all possible approaches. Some general characteristics are listed from existing theories. Then a simple desk lamp design is analyzed step–by–step to obtain features in a design process, some of which are general, called "common characteristics". Finally, a cam design is examined briefly to show the generality of these characteristics.

3.1 PURPOSE AND SCOPE

This understanding is to extract facts from the design process for the later establishment of theories. Facts are gathered from the common characteristics. (Figure 3.1)

$$\frac{\text{Facts / Characteristics}}{\text{Understanding}} \Longrightarrow \frac{\text{Theories}}{\text{Model \& Computer Based Tool}}$$

Figure 3.1 : The purpose of understanding the design activities

To date, the design process has not been well studied by researchers. No generally accepted theories have yet been established. This is the reason that this understanding is undertaken in order to get first hand experience of the original problem: the design process, before continuing to further work.

The understanding of design activities may be slightly different in terms of the scope. Some researchers are interested in the understanding of design in a general sense, which covers the most fundamental nature of design and, thus, is quite sophisticated. The understanding in this work is much more practical and intends to serve the specific objective. It only covers the routine mechanical component and mechanism design processes.

It is believed that if the understanding of all design activities is difficult, then the understanding of some of them is meaningful. The more such understanding with certain scope is achieved, the more completely the whole design area will be covered.

3.2 BASIC STRATEGY

To date, three methodologies have been developed to understand the design activities: the descriptive model, the prescriptive model and the "practice–oriented" model. As explained in the last chapter, the "practice–oriented" strategy is practical and has been chosen for this work, taking into consideration the complexity of the design activities, the current research status in design theory and the basic computer implementation–oriented stand taken in this thesis.

The idea of "design case studies" is emphasized in this strategy from the point of view that the work should be based on the study of the original problem --- the design activities. So far, the desk lamp and the cam have been chosen as the examples to be analyzed. These examples are considered typical in a certain range of routine mechanical design activities.

The understanding process is also based on the author's own design experience, and is also influenced by some prescriptive theories available in many engineering design textbooks and documents related to the design study (discussed in the following "General Characteristics" section), which are thought to be reflecting the results from the observance of the chosen examples.

3.3 GENERAL CHARACTERISTICS

Various versions of design features have been prescribed by researchers in this area. Some of them have influence on the later observance of the design examples, and it is necessary to cite them.

The design process is characterized as a series of transformations leading from specifications to physical components. Each transformation implements a component or decomposes it into sub–components [61]. In other words, each part or component is replaced with a more detailed part or component in each transformation. A more general process can happen when transformation is implemented each time on the complete description of the design, instead of on one single component. The coupling between parts or components in design could be handled based on constraint propagation.

The mechanical design process usually follows a sequence of steps from the recognition of requirements and specifications, to the concept formulation and conceptual design, to the embodiment of design details, finally to production, sales, maintenance and other stages in the product life cycle. Each step is a refinement or transformation from one abstraction level to a lower one. Eventually, a product is transformed from the original requirements. The functions and configurations are important terms in this functional to physical domain transformation process. Coupling or dependency between various parts or components in design should be specially noted.

Design is a highly experience–oriented activity with enormous and diverse knowledge. This feature should be kept in mind when devising representation schemes for design knowledge. It also contributes to the large design solution space. There might be many products fulfilling a function, and many possible components constituting a product. Finally, the evaluation of design is very difficult. Since a design is achieved largely from "experience", it is hard to tell whether this design is good or not. Also, it means that it is difficult to achieve a good design.

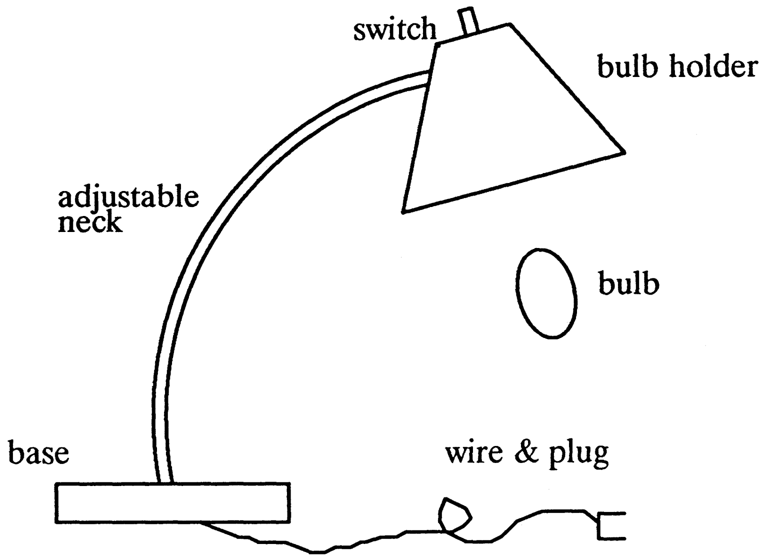
3.4 OBSERVANCE OF A LAMP DESIGN

Two points should be explained before continuing to the following observance. First, the desk lamp may not be exactly classified as a mechanical product. Such a common device is chosen as an example for its simplicity and for the convenience of explanation. Second, this understanding is supposed to be in the routine mechanical design area. In the later implementation, typical mechanical products: the cam and the bolted flange, are selected to develop their specific design systems.

The understanding process is described in such a way that when the design process in the chosen example is followed step by step, the facts in this process are exposed. Based on these facts, the common characteristics are summarized. Terms used in the understanding will be explained as they appear.

Two examples will be examined: a desk lamp and a cam. There is slight difference between the description style of these two examples. The desk lamp design is followed closely in terms of the designer's thinking process, while the concrete design procedures and knowledge may not be listed. Such a common device is chosen precisely for the convenience that its design knowledge is so obvious that it does not need to be detailed. The purpose of this observance is to expose the major stages and the features of each stage in its design process. While in the cam example, the true design procedure or "facts" are exposed first without any opinions being imposed. After that, these facts are "cross–examined" by those features summarized from the desk lamp observance, in order to prove that the features from the desk lamp design also exist in the cam design. Thus, they are common to both design examples. It is further assumed that they should be common to a number of design activities which share similarity with these two examples.

A desk lamp may consist of several parts: bulb, bulb holder, switch, adjustable neck, base, wire and plug, and connections between these parts. Figure 3.2 shows an example.



connections :

. bulb & bulb holder
. bulb holder & neck
. neck & base

Figure 3.2 : A desk lamp

In order to get a close look at the whole design process, the observance starts from the very beginning.

The following requirements are given to the designer by the client

. A lighting equipment is needed for an office desk;

. This equipment must be portable;

. It will work under 110 volts;

. The equipment must be adjustable for the user to get comfortable lighting;

. The cost is around $50.

The designer considers these specifications and regards these items as feasible to start the design process. In some cases, he may have disagreement about the given specifications, and both parties have to discuss and reach an accepted specification list.

☞ The following stage will achieve all configurations:

Now the designer thinks about what the prospective product could be. At this stage, a very important concept should be introduced, "**function**". The term refers to the general input/output relationship of a system (or subsystem component) whose purpose is to perform a given task. It describes the expected behavior of a component or a system. In this case, the function of the prospective product is "lighting an office desk".

By his knowledge or from accumulated knowledge, he realizes that a desk lamp could serve the purpose. However, it must be pointed out that not only a desk lamp can light an office desk, other kinds of illuminating sources could serve the purpose too. Nevertheless, the desk lamp is the most common one. It must be also realized that there are many possible types of desk lamps available in this commercial world. The point made here is that many possible products could serve the same function. Conversely, one product could also serve multiple functions. In this case, a desk lamp can light not only a desk, but also a part of a room.

Another term introduced here is "**configuration**". It is an aggregation of physical components, which can serve certain functions. A configuration can consist of a number of sub–configurations. What has been obtained just now is a multiple–to–multiple relationship between functions and configurations. A "principal configuration" is the final product which serves the original purpose or "principal function", and consists of a number of small configurations or components.

The designer selects the desk lamp shown in Figure 3.2 as the principal configuration in this case. Sometimes, the choice under the circumstances that multiple configurations are available can be made by other specifications, while in other cases, it is simply decided by the designer's preference as in this example. So far, a principal configuration has been obtained by considering the functional requirements.

Now, it is time to determine all components of the principal configuration. The designer has two possibilities at this moment. He may find a similar desk lamp designed before, but it is not adjustable. Therefore, he decides to simply change the old design to meet the "adjustable" requirement. The other possibility is to design from scratch. In this work, the first design approach is not considered, which may be called the "retrieve & change" approach. This approach has quite different features from, and must be based on the "design from scratch". This thesis focuses on the "design from scratch" approach.

At this time, the designer already has a rough principal configuration in his mind, which consists of a "lighting part", a "support part" and an "electricity connection part". They serve the "lighting function", "supporting function" and "connecting electricity function". These functions combined serve the principal function of "lighting a desk".

Now, the designer decomposes each part into sub–parts. For instance, a lighting part may consist of a cylindrical bulb holder, a bulb and the connection between them. In fact, he handles the lighting part in the same way as he deals with the rough principal configuration: decomposing it into several parts. Here, an important concept, "**decomposition**", is introduced, which describes a refinement process in design. The decomposition of the desk lamp is illustrated in Figure 3.3.

A "**connection**" is a configuration used to link two parts or more together. A connection can consist of several sub–parts, like an ordinary configuration. But unlike an ordinary configuration, it can consist of "**partial configurations**". Figure 3.4 shows a screw connection between the base and the neck, which consists of two partial configurations — a thread on the neck and a hole on the base, and an ordinary configuration — a screw nut.

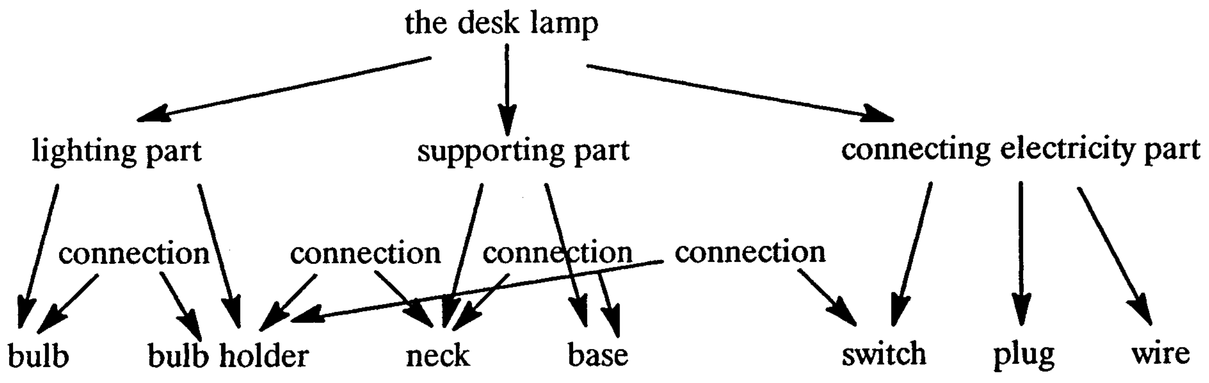


Figure 3.3 : The decomposition of the desk lamp, with all configurations

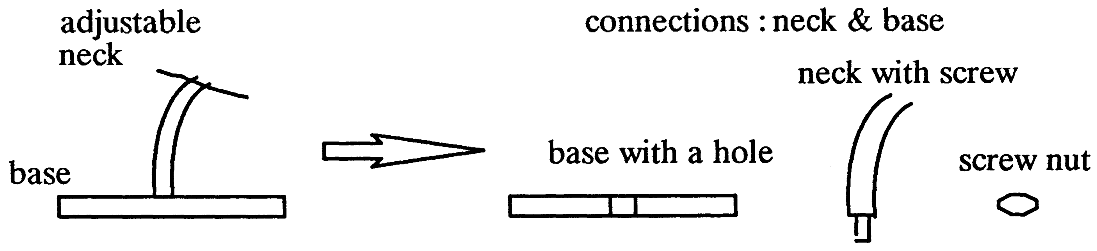


Figure 3.4 : A screw connection between the base and the neck

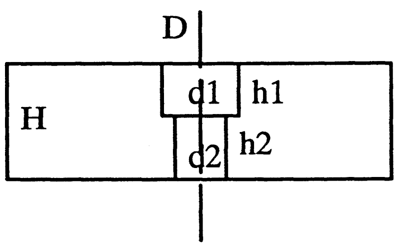
Thus far, the following facts have been revealed:

1. Considering the principal function, a principal configuration can be obtained;
2. The decomposition process is applied to the acquirement of all configurations. Each decomposition process produces a group of sub–configurations with their sub–functions fulfilling the original configuration's function. This decomposition process continues until all sub–configurations can not be further decomposed, such as the bulb. These undecomposed configurations are defined as the "basic configurations".

☞ All detailed values are to be obtained from now on:

The above decomposition process achieves all conceptual configurations, which have not been assigned attribute values yet. For instance, no dimensions, materials and the like have been determined.

Detailed information about configurations can be expressed in attributes or parameters. For example, the base is expressed as a set of parameters (Figure 3.5).



parameters :
structural – H,D,d1,h1,d2,h2
relational – x0,y0,z0,rotx,roty,rotz
material – steel
etc. ...

Figure 3.5 : Parameters of the lamp base

Generally, the parameters can be classified under two categories: numerical and non–numerical. The dimensions are typically numerical parameters, while the material is non–numerical.

The parameters can be organized according to the configurations. This is obvious in the dimension and the material case. Relational dimensions of two or more configurations can be attached to their upper configurations or a configuration with an overall coordinate system, depending on the convenience of the design or the designer's preference.

Basically, there are two aspects in a product design: geometric aspect and parametric aspect. One covers the geometry information such as drawings, and especially, the geometric and spatial reasoning process. The other provides information in parameters, and is not related to non–verbal geometry design process. It should be noted that the parameters provide geometry information also, such as dimensions.

Up until now, the detailed information can be expressed in parameters and organized according to configurations. The next question is how to assign values to these parameters.

The parameters are determined by "**design relations**". A design relation is a relationship among a group of parameters. "The neck should be strong enough to stand the bending moment" is a relation involving the yielding strength of the neck, the neck diameter, the neck length, and some other parameters. This relation can be used to achieve the minimum neck diameter, or to check the neck strength.

The detailed design is an iterative process. The designer makes certain assumptions to get some initial parameters. Then, when enough information is obtained, the assumptions are checked and redesigned, if necessary, according to the designers' experience and the current circumstances. For instance, if the neck diameter is checked to be too small to support the bulb holder after the bulb holder has been redesigned, the designer decides to increase the diameter according to the strength calculation. This iteration can happen many times.

The iterative process is also related to the dependency among parameters. The term, "**dependency**", means the direct relation among parameters.

Besides using design relations, the designer also assigns values to some parameters by experience, when relations are not available. This kind of parameters may be called the "assumed parameters", and they have to be checked, once the design proceeds to a certain stage. For example, there is no obvious relation to determine the desk lamp overall dimensions, and thus, they are assigned according to the designer's preference and the portability requirement. Later, these dimensions are checked to ensure that they meet the strength requirement.

Therefore, there exist some principles behind the parameters and design relations:

1. *if number of design relations > number of unknown parameters, some relations are used for checking* *("design-by-relations")*

2. *if number of design relations < number of unknown parameters, some parameters are assigned by the designer* *("design-by-experience")*

3. if number of design relations = number of unknown parameters, all relations are used to obtain a unique set of parameter values ("unique case")

There are many ways to use design relations and assumptions to achieve the same parameters. For instance, the dimensions of the base could be determined by considering the size of the possible desk lamp, or could be designed from the maximum weight of all parts if they have been decided. The designer's experience plays an important role in deciding what design relations are to be used, and at what circumstances. Therefore, the design relations should be organized in groups. The introduction of design relation groups from one perspective shows the diversity and large amount of knowledge in the detailed design.

☞ Non–functional considerations:

Although the major concern at the design stage is functional, the designer has to consider other factors at various stages of the product life cycle, such as the desk lamp cost. The contemporary trend in design is to cover these factors at the early design stage.

Now the detailed parameters of all configurations of the desk lamp have been achieved. The whole design process is ended.

3.5 COMMON CHARACTERISTICS

The following is a summary of facts from the above observance and analysis. There are four stages in the design process:

☞ Given Specification Stage:

. Given specifications are processed by the designer to obtain an agreeable set of specification items.

☞ Configuration Achievement Stage:

. The decomposition process obtains the sub–configurations of the product. This decomposition continues until the basic configurations which can not be decomposed further.

. There is a multiple–to–multiple relationship between functions and configurations. The configuration decomposition is also a multiple–to–multiple relation. A configuration could be decomposed in more than one way.

. Once the prospective product ("principal configuration") is determined from the original functions ("principal functions"), the functional considerations no longer play an important role in the decomposition process, since the possible configuration combinations are known in mature design activities.

. The product consists of configurations and connections in a layered graph structure. The connections may be comprised of partial configurations, which are parts of normal configurations.

☞ Detailed Design Stage:

. This stage is responsible for assigning values to the configurations' attributes. Parameters are organized according to configurations. Lower level configurations can access parameters of the upper level configurations.

. The parameters are achieved by either design relations or assumptions, depending on whether there are enough relations.

. Since there may be various ways to use the same design relations to achieve certain parameters, or different design relations to achieve the same set of parameters, and the design relations are in large amount and great diversity, it is necessary to organize them in groups.

. The iteration happens frequently at this stage. Whenever a checking fails, it is advisable to redesign some parameters. These advices come from the designer's experience. Furthermore, the dependent parameters have to be checked and modified, due to the dependency among parameters.

☞ Non–functional Considerations Stage:

. Some non–functional considerations affecting other stages of the product cycle can be expressed in design relations.

If any one of these stages fails, the message is sent to the previous stage, with suggestions for the redesign of the previous stage. So far, the redesign of parameters has been handled, but the mechanism of how to redesign conceptual configurations and re–specification due to any failure in the detailed design has not been well understood. This case is rare in mature design activities.

To prove the generality of these summarized features, a cam design is shown below in such a way that the designer's real working process is described first (an excerpt), then a brief explanation points out the stages and common features at each stage.

3.6 ANALYSIS OF A CAM DESIGN

☞ Problem statement

- a required motion transformation from rotation to translation in a sewing machine
- 2.00″ rise per 0.05 seconds
- then dwell 0.015 seconds
- a small dwell at the bottom of the motion is required
- because of space limitation, radius of base circle is no more than 1.25″
- pressure angle is no more than 30°
- smooth acceleration, no steps in acceleration are allowed
- steps in jerk are acceptable

☞ Start

The motion pattern will be a "DRD" — dwell rise dwell return dwell.

Since in the rise part, no steps in acceleration are allowed and steps in jerk are acceptable, the cycloidal motion is chosen.

Since there are no specific requirements for the return, SHM (simple harmonic motion) is selected.

A commonly used disc follower is selected.

☞ Continue

Angular velocity is $2\pi/0.05$ or 40π (rad/sec).

☞ Program

Dwell at the top of rise: $0.015 \times 40\pi = 108^\circ$ (approximately).

Allow 10° for lower dwell. Total angle for rise and return is: $360-108-10=242^\circ$.

Cam factors for rise and return: f–SHM = 2.72, f–cycloidal = 3.46 .

Rise program: $3.46/(3.46+2.72) \times 242 = 136^\circ$. Return program: $242-136=106^\circ$.

☞ Maximum velocity and acceleration

Rise -- cycloidal :

$$y = d\left(\frac{\theta}{\beta} - \frac{1}{2\pi}\sin 2\pi\frac{\theta}{\beta}\right) \qquad (3.1)$$

$$v = d\frac{\omega}{\beta}\left(1 - \cos 2\pi\frac{\theta}{\beta}\right) \qquad (3.2)$$

$$a = 2\pi\frac{\omega^2}{\beta^2}\sin 2\pi\frac{\theta}{\beta} \qquad (3.3)$$

When $\theta = \beta/2$, $v_{max} = 211.7$ ips; and, $a_{max} = 35221$ ips^2 .

Return — SHM :

$$y = \frac{d}{2}\left(1 - \cos\pi\frac{\theta}{\beta}\right) \qquad (3.4)$$

$$v = \pi d\frac{\omega}{2\beta}\sin\pi\frac{\theta}{\beta} \qquad (3.5)$$

$$a = \frac{d}{2}\left(\pi\frac{\omega}{\beta}\right)^2\cos\pi\frac{\theta}{\beta} \qquad (3.6)$$

When $\theta = \beta/2$, $v_{max} = 213.4$ ips; and, $a_{max} = 27662$ ips^2 .

☞ Pressure angle

Pitch radius : $R_p = R_b + R_f + 1/2 \times \text{rise} = 2.75$ (3.7)

Cam factor: $f = R_p \times \beta/\text{rise} = 3.264$ (3.8)

$$pressure\_angle : \phi = \phi 1 + \frac{(cam.f2 - cam.f)}{(cam.f2 - cam.f1)}(\phi 2 - \phi 1) \qquad (3.9)$$

$$= 30 + (3.46-3.264)/0.6 \times 5 = 31.6^{o}$$

(These are two nearest cam factors and pressure angles from the cam factor table in [65]. An excerpt is shown in Table 6.10)

Another way to obtain the pressure angle is:

$$\tan\phi = (V_{fmax}) \,/\, (R_b + y_p)\,\omega \qquad (3.10)$$

Checking pressure angle, which should be less than 30.0

But, now $31.6 > 30.0$. Try to reduce the pressure angle by offset.

☞ Reducing pressure angle by offset

$$\tan\phi = (V_{fmax} - e\omega)/y\omega, \quad y = (R_b{}^2 - e^2)^{1/2} + y_p \qquad (3.11)$$

try $e = 0.5''$, $\phi = 30.64$, increase e by $0.1''$, . . ., until $\phi < 30.0$

☞ Check the undercutting, the minimum radius of curvature should be greater than the follower radius. The radius of curvature :

$$\varrho = \frac{[(R_0 + y)^2 + (\frac{dy}{dt}\frac{1}{\omega})^2]^{3/2}}{(R_0 + y)^2 + 2(\frac{dy}{dt}\frac{1}{\omega})^2 - (R_0 + y)(\frac{d^2y}{dt^2}\frac{1}{\omega^2})} \qquad (3.12)$$

ρ_{min} usually occurs at the point of maximum negative acceleration.

At A_{max}, $\theta = 3/4\ \beta$, $\rho_{min} = 2.23 > R_{follower}\ (0.5)$, Checking satisfactory !

The following is a brief explanation of the features exposed in the above design process excerpt, compared with the common characteristics summarized earlier.

☞ The "problem statement" is the specification development stage. Some specification items are derived from the original specifications. No process is needed to reach an agreeable specification list.

☞ In "Start", the conceptual product was determined as a cam with four sub–configurations and a follower. These sub-components were obtained by decomposition and functional considerations. The connections between the sub-components in the cam are partial configurations: plane geometry features, which deliver constraints which require that the two adjacent parts have equal length and height.

☞ From "Continue" to the end, all parameter values were achieved. These parameters can be organized under each configuration.

The parameters were achieved by design relations (e.g., rise program), or assumptions (e.g., lower dwell is assigned 10^o). There are various ways to use design relations to get parameters, as in the case of pressure angle. The designer has to express his knowledge about how to use these relations.

The iteration happened in checking the pressure angle. When it exceeds 30^o, the offset is advised, and the pressure angle is redesigned, as well as all dependent parameters. This iteration process continues until the pressure angle is reduced to less than 30^o.

The design relations were in diverse formats, such as equations, tables, constraints and so on. Even this very small piece of design process excerpt has shown the large amount of design knowledge.

The above stages and features at each stage coincide with the common characteristics presented in section 3.5. These summarized features are common to both cases. It is further assumed that these features are basic and common to a number of routine mechanical component and mechanism design processes. **This assumption is the basic foundation of further work in this thesis.**

CHAPTER 4 :
GENERAL DESIGN MODEL

This chapter describes a general model for the routine mechanical design activities. This general model is developed from the common characteristics extracted from the understanding of routine mechanical design activities. Since these characteristics are basic and common to a certain range of design processes, the model developed this way is general to these processes.

The model only covers the "design from scratch" approach in the routine mechanical design. It is an informal description of the design process which means that it is not bound by any formal modelling techniques. It is like a method for organizing exposed facts from the design process according to certain points of view.

The purpose of achieving such a model is to develop its computer implementation, which will serve as a shell system with general reasoning mechanisms and knowledge representation formats. Thus, specific design systems can be developed efficiently by simply filling domain knowledge into this shell.

The model is described according to the whole design process and individual stages including Specification Development, Synthesis, Analysis and Non–functional Considerations. The key issue in the model is the knowledge representation scheme which consists of the Mechanisms underlying the design process and the Knowledge Representation Formats.

4.1 THE WHOLE DESIGN PROCEDURE

According to the Chapter 3, the whole design process consists of four stages : "Given Specification", "Configuration Achievement", "Detailed Design" and "Non–functional Considerations". In the model, they are called "Specification Development", "Synthesis", “Analysis” and “Non–functional Considerations” correspondingly. There is feedback between these stages, which may be used to guide the redesign process in the upper level design stages (Figure 4.1). These major stages roughly describe a common procedure to all design activities.

$$\begin{tikzcd}
\text{Specification Development} \arrow[d, shift right=2] \\
\text{Synthesis} \arrow[u, shift right=2, "\text{feedback}"'] \arrow[d, shift right=2] \\
\text{Analysis} \arrow[u, shift right=2, "\text{feedback}"'] \arrow[d, shift right=2] \\
\text{Non–functional Considerations} \arrow[u, shift right=2, "\text{feedback}"'] \arrow[d] \\
\text{End}
\end{tikzcd}$$

Figure 4.1 : Design stages in the whole design process

In the following sections, each stage will be described by addressing three issues: 1) the mechanisms underlying the design activities; 2) the knowledge representation formats; and 3) the examples from specific design knowledge. The first two issues are common to all design activities which make the general model possible. They constitute the “knowledge representation scheme”, of which the first contributes to the reasoning process, and the second to the formats of representation.

4.2 SPECIFICATION DEVELOPMENT

The specification development is a very complicated process. A number of activities are involved: discussion between the designer and the client to reach a mutually acceptable specification list, to eliminate conflicts, and the like. It is not necessary to put much effort into such mechanisms at this moment, if the major focus is on the achievement of design from a given specification set. In this work, the above issues are not addressed, and a workable specification set is assumed to be provided by the user.

The application environment of a mechanical product or system can vary greatly. Thus, the specifications for given applications can be very diverse. It is necessary to reduce the application requirements into the commonly used standard specifications, when the computer implementation is concerned. For example, a cam may be used in a sheet metal transfer mechanism for a punching machine with specifications like punch cycle, maximum acceleration and so on, or it may be used in a sewing machine with specified angular velocity, suggested motion patterns and space requirements. Despite their differences, these application requirements can be transformed into items like angular velocity, basic motion constraints, maximum acceleration, pressure angle and so on. By simplifying the above complicated process into a transformation of the requirements from a particular application to a standard set of specifications, the modelling of its mechanism can be quite straight forward. Figure 4.2 shows this simplified process.

$$\begin{tikzcd}
\text{Original Applications} \arrow[d, "\text{extraction}"] \\
\text{Commonly Used Specifications}
\end{tikzcd}$$

Figure 4.2 : Specification development

Specifications are organized into a group of items. Each item is represented descriptively by a string pair, where the first element is the item name, and the second a string of arbitrary length for the value of this item. For instance, the following specification excerpt:

(angular.velocity , 136.0)

(motion constraint , no steps in acceleration, steps in jerk acceptable)

is a part of the specifications for the cam design.

4.3 SYNTHESIS

The synthesis stage achieves all structural information about configurations without any values assigned to their parameters. The description of this stage follows first the basic relations, then the mechanisms underlying the synthesis process, and finally the knowledge representation formats.

o Basic Relations

In the routine mechanical design, a multitude of relations between functions and configurations and various configuration combinations are known. For example, a desk lamp has the function of "*lighting a desk*", and it can be decomposed into "*lighting part*", "*supporting part*" and "*electrical power connection*" parts. There are two types of relations in the above description: the "<u>function–to–configuration</u>" and the "<u>configuration–decomposition</u>" relation.

Both types are multiple–to–multiple relations. A number of configurations may fulfil one function and various functions may be achieved by a single configuration. A configuration may be decomposed in a number of ways, producing different groups of sub–configurations.

Either function–to–configuration relations or configuration–decomposition relations can carry conditions and constraints. Conditions must be satisfied to execute the relation. Constraints are the side effects which occur once the relation is in effect. For instance, Table 4.1 shows a function–to–configuration relation in the cam design.

The function "*rise*" combined with the motion condition implies a "*cycloidal wedge*" configuration plus the motion pattern constraints.

It should be noted that although the focus of the synthesis stage is the configuration decomposition, the "principal function" is essential in determining the prospective product.

Table 4.1: Function–to–configuration relation

Condition	*Function*	*Configuration*	*Constraints*
No steps in acceleration, steps in jerk acceptable	*rise*	*cycloidal wedge*	*motion–pattern: cycloidal–motion*

Thus far, the two most important kinds of relations have been described. The third type is the "function–checking" relation which describes a group of functions combined to perform a certain function. The function–checking relations can be used to ensure that the functions of sub–configurations fulfill the function of the decomposed configuration. However, this is not the major focus in this work, since the possible patterns of decomposition already exist in the routine design. It is used as a secondary checking line in the reasoning process. One example of this kind of relation looks like :

Table 4.2: Function–checking relation

Function	*Sub–functions*	*Conditions*
Rotation–to–translation	*Rise, Dwell, Return*	*Motion–pattern : RDR*

Functions "*rise*", "*dwell*" and "*return*" combined together fulfil the motion transmission function from "*rotation to translation*", with the condition of motion pattern "*RDR*".

o The Mechanism Underlying The Synthesis Process

The basic mechanism of this stage is defined as the "Configuration Decomposition Approach" (C. D. A.), and can be described as follows:

. Define the principal function / a function;

. Find the principal configuration / a configuration, which fulfils this principal function / the function;

. Decompose the principal configuration / the configuration into a set of configurations;

. Continue this decomposition process until the basic configurations ("basic functional design units") are reached, which can not be decomposed further.

This mechanism is further illustrated in Figure 4.3. The solid line shows the major concerns at the synthesis stage. This is where the term "configuration decomposition" comes from. The dash line implies the checking process by using function–to–configuration relations and function–checking relations.

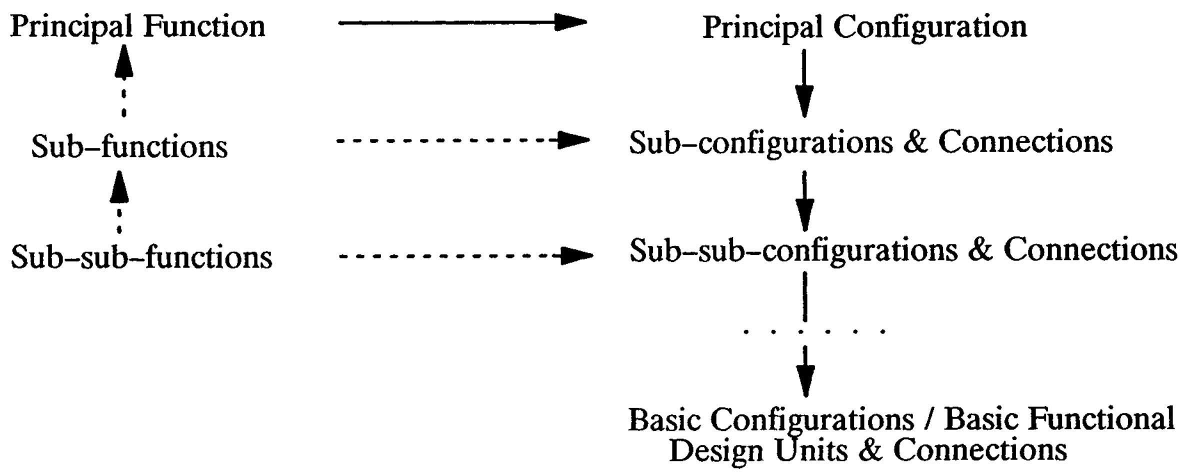

Principal Function
Principal Configuration
Sub–functions
Sub–configurations & Connections
Sub–sub–functions
Sub–sub–configurations & Connections
Basic Configurations / Basic Functional Design Units & Connections

Figure 4.3 : Configuration Decomposition Approach (C. D. A.)

Special attention is paid to the interaction or "connections" among configurations. A connection is represented almost in the same way as normal configurations. The only difference is that a connection may consist of a "partial configuration", which is a part of another configuration (either ordinary or partial).

To make clearer the C. D. A. mechanism at the synthesis stage, an example of the desk lamp synthesis is shown in Figure 4.4.

Thus far, the mechanism of the synthesis stage has been described. The next key issue is the knowledge representation formats.

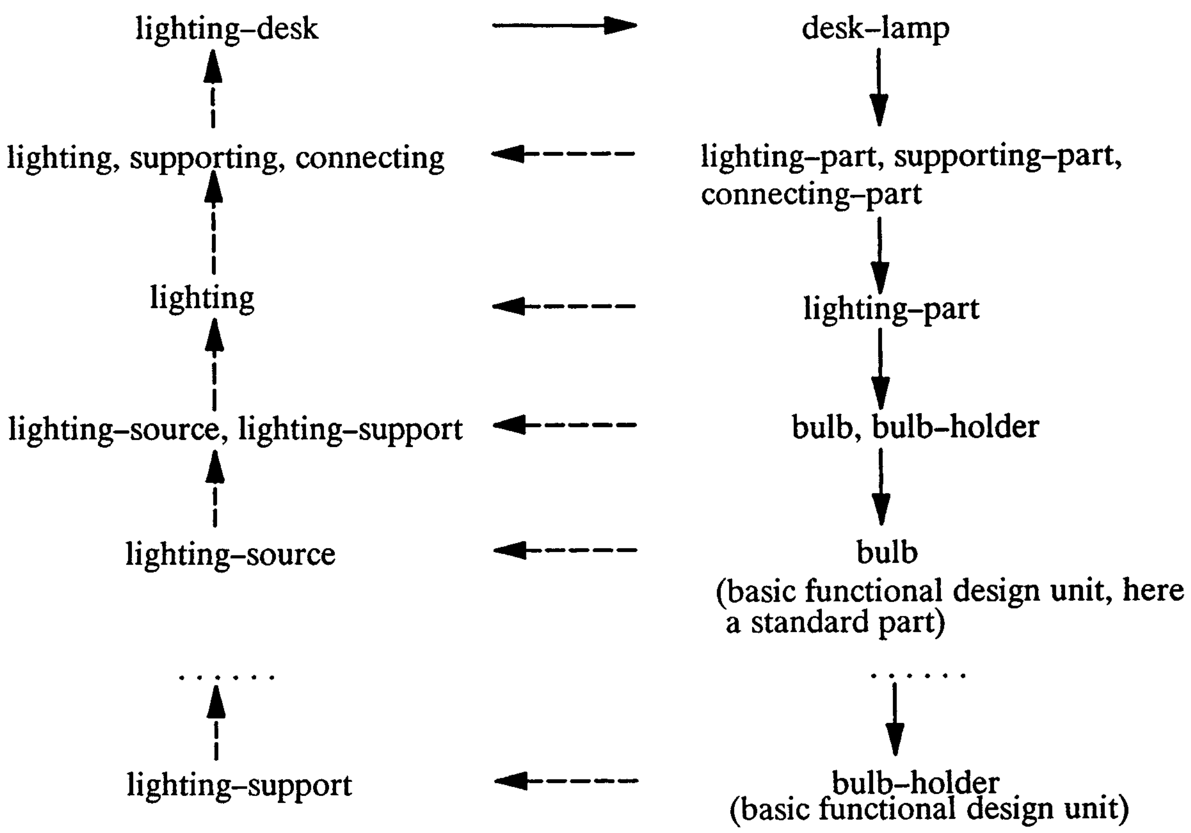

lighting-desk
desk-lamp
lighting, supporting, connecting
lighting-part, supporting-part, connecting-part
lighting
lighting-part
lighting-source, lighting-support
bulb, bulb-holder
lighting-source
bulb
(basic functional design unit, here a standard part)
......
......
lighting-support
bulb-holder
(basic functional design unit)

Figure 4.4 : A portion of synthesis in desk lamp design

o Knowledge Representation Formats

Basically, there are two types of representations required at the synthesis stage: for the basic relations and for the configurations and functions. No matter what kind of basic relations are concerned, they all consist of four parts: "*(condition, object1, object2, constraints)*". Under the "*conditions*", the relation between "*object1*" and "*object2*" is effected, along with the generated "*constraints*". These four parts are represented in string list.

The functions and configurations are simply represented in strings, since at this stage, no detailed information about the configuration is necessary.

A function-to-configuration and a function-checking relation have been given earlier (Table 4.1 and 4.2). Here, a configuration-decomposition relation is shown in Table 4.3.

Table 4.3: Configuration–decomposition relation

Conditions	*Configuration*	*Sub–configurations*	*Constraints*
(Motion–pattern: RDR)	*(cam)*	*(rise, conn.of.rise.dwell, dwell, conn.of.dwell.return, return, conn.of.return.rise)*	*(nil)*

Another representation issue is how to organize configurations. Since the configurations produced at the synthesis stage will be the basis for the analysis, an organization for these configurations is essential. Considering the decomposition approach in the generation of these configurations, it is natural to come out with a layered graph structure, which is defined as the "configuration tree". One configuration is represented by a node in a given level with its sub–configurations represented by nodes attached to it in a lower level. This configuration tree stretches to the lowest level in which basic configurations are leaf nodes. A connection lies one level higher than the configurations of which it consists. Figure 4.5 is a part of the configuration tree from the cam design.

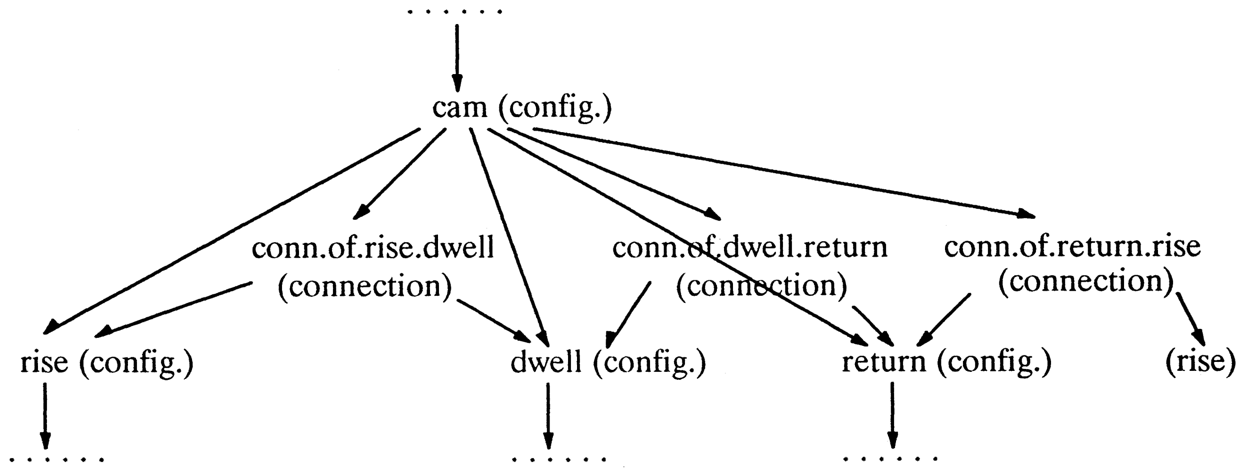


Figure 4.5 : A portion of a sample configuration tree

This tree will be passed to and used in the analysis. The difference between the two trees is that the synthesis one is a tree without any values for parameters, while the analysis one is full of values.

There is a function tree with the same structure as the configuration tree. Each function is associated with a configuration. A group of sub-functions in a lower level can fulfil a function in an upper level. It is the product of the function checking during the reasoning process. This function tree keeps track of the synthesis process, and can be used for subsequent retrieval, checking and explanation.

Although the configurations and functions are represented in string lists, basic definitions of them are necessary for explanation and conflict avoidance, when the development moves on to a large scale. Thus, a <u>dictionary structure</u> is devised, called the "Configuration and Function Dictionary". Every configuration or function has an entry in the dictionary with its name, explanation and optional items. A configuration "cam" is defined in Table 4.4.

Table 4.4: Configuration item in the dictionary

Name	*Explanation*	*Optional items, such as "type"*
cam	*a cam in a cam system . . .*	*type : non–basic*

o In summary, the following issues in the synthesis model have been described:

– Reasoning Mechanism: Configuration Decomposition Approach achieves configurations by using function–to–configuration, configuration–decomposition, and function–checking relations.

– Knowledge Representation Formats: <u>Basic relations</u> are represented in the form of "*(conditions, object1, object2, constraints)*" ; configurations and functions are formatted in strings and stored in the <u>configuration and function dictionary</u>; <u>partial configurations</u> are created for the connections; configurations and functions are organized in the <u>configuration</u>

tree and the function tree; basic functional design units form the basic level of the configuration tree.

4.4 ANALYSIS

The input to the analysis from the synthesis is the configuration tree, full of rough structures. The output will be a similar configuration tree with parameters and their values.

The same description style is used for the analysis: first, the mechanisms underlying the analysis process, and then the knowledge representation formats.

o The Mechanisms Underlying The Analysis Process

▪ The Basic Mechanism And The Fundamental Principle

The basic mechanism in solving these parameters is by the use of design relations. A design relation is a relationship among certain parameters, which is a different concept from the "basic relation" in the synthesis. Once given some parameters, this relation can be used for solving unknown parameters. For example, a design relation given in the cam design reads:

$$tan(\theta)_{(pressure.angle)} = \frac{V_{fmax_{(maximum-velocity)}}}{(R_{b_{(base-radius)}} + Y_{p_{(corres-displacement)}})\omega_{(angular-velocity)}} \qquad (4.1)$$

Given V_{fmax}, R_b, Y_p and ω, the pressure angle θ can be determined, or given the pressure angle θ, V_{fmax}, Y_p and ω, R_b can be calculated.

The design relation is not limited to equations. It can be inequalities such as :

$$\theta_{(pressure-angle)} < 30.0^{o} \qquad (4.2)$$

This relation can be used to determine parameters by constraining the pressure angle not to exceed 30.0.

A design relation can also be expressed in words. For example, a design relation in designing the offset of a cam is:

. if the pressure angle is more than 30.0

and the cam size (R_b) can not be changed due to space limitation,

. then one way to reduce the pressure angle is to set an offset e

This design relation expresses a relationship among pressure angle, R_b, and offset e.

The design relations can be classified under two categories: numerical relations which can be expressed in equations, inequalities, tables, figures, default values and so on; and non–numerical relations which can be expressed in words with conditions, or constraints, and certain consequences. But generally, both numerical and non–numerical relations express relationships among parameters.

The fundamental principle of achieving parameters by relations has been described by:

if number of design relations > number of unknown parameters, some relations are used for checking (*"design–by–relations"*)

if number of design relations < number of unknown parameters, some parameters are assigned by the designer (*"design–by–experience"*)

if number of design relations = number of unknown parameters, all relations are used to obtain a unique set of parameter values (*"unique case"*)

This principle seems very simple. It becomes very sophisticated once it comes to its realization, because this principle lies deep beneath the designer's real working process. The following is the modelling of the designer's analysis process based on the above principle.

▪ Analysis Process Mechanism

To simplify the description, two kinds of design knowledge classified earlier: "design relations" and "design–by–experience", are listed under one category -- design relations, since the "design–by–experience" can be represented by design relations in the form of "the parameter P is assigned value V".

At the analysis stage, the number of design relations is very large. More than one design relation may lead to the determination of one parameter. Conversely, one design relation could be used to solve different parameters. For example, besides the equation (4.1), another relation can also be chosen to decide the cam pressure angle:

$$\theta_{(pressure\text{-}angle)} = \theta_1 + \frac{(cam.factor_2 - cam.factor)}{(cam.factor_2 - cam.factor_{1)}}(\theta_2 - \theta_1) \qquad (4.3)$$

It is necessary for the designers to decide under what circumstances, certain design relations are to be used and in what order. To solve this problem, certain organization on design relations has to be devised.

Parameters can be organized according to configurations. Each configuration is attached to a number of parameters. When configurations are handled one by one, their parameters are solved simultaneously.

To determine parameters, a group of design relations have to be chosen from the whole relation set as the "usable design relations". It is critical to develop a method to choose these "usable design relations". The basic idea is to select the usable set of relations according to the "solved parameters" and "to–be–solved parameters". Thus, it is required that parameters of configurations be always classified into two sets : "solved" and "to–be–solved". In addition, the design relations have to be organized in a certain order, since there may be many ways to use them to determine the same parameter(s). The details will be contained in the next chapter. Here, it is simply described as choosing the "usable design relations" from the "known" situation in the design process.

The solving process of configurations and parameters is an iterative process. While configurations are handled one by one, parameters of one configuration may not be completely solved at once. Therefore, later iteration is inevitable.

Figure 4.6 gives an overall illustration of the mechanisms described above.

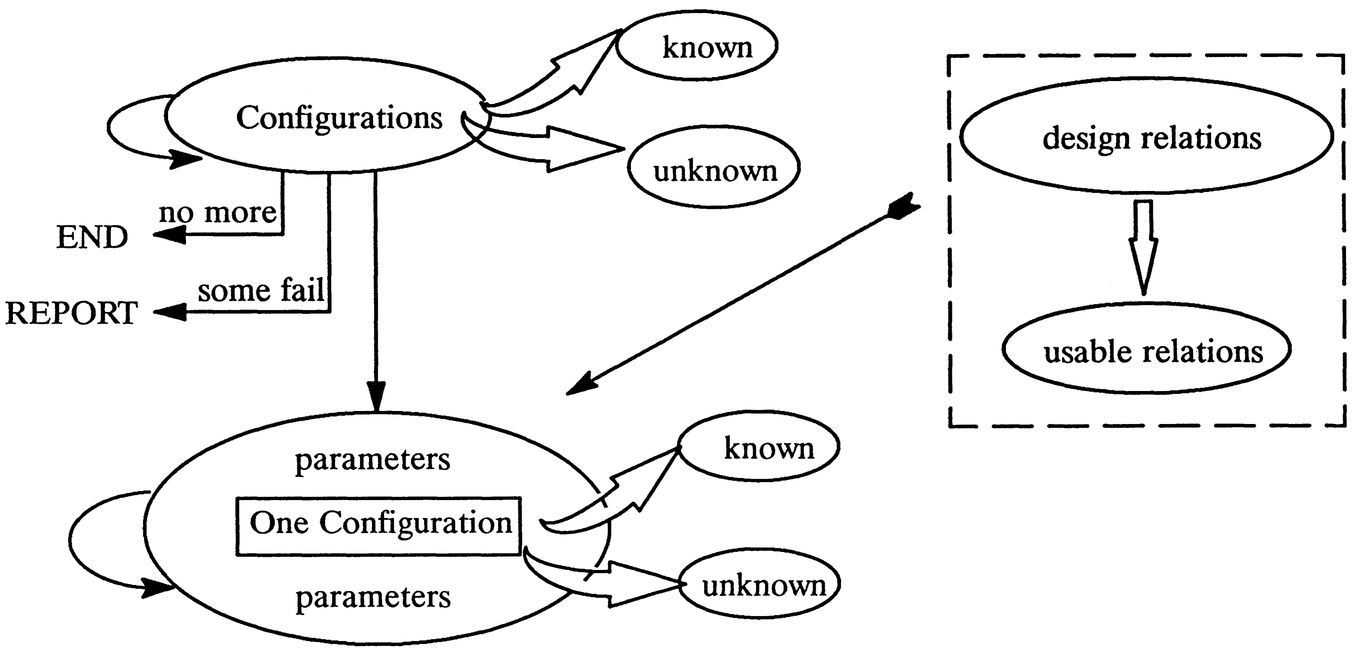


Figure 4.6 : The mechanism of the whole analysis process

▪ Dependency / Iteration Process

Iteration happens frequently in solving parameters. Unlike the iteration process explained earlier, this iteration may be best described as "dependency / iteration process". This occurs when a checking relation is executed and not satisfied. Then some design relations are called in to change the earlier design. Once some parameters are changed, parameters which "depend" on those changed parameters may need modifying. The dependency means that the solving of some parameters is related to others. For example, in relation (4.1), the pressure angle θ depends on V_{fmax}, R_b, Y_p and ω. If R_b is modified due to some checking relations, the pressure angle θ has to be changed too.

In a real design process, the designer assumes some parameters at the very beginning, because of the lack of known parameters. Later, when the design continues and enough parameters have been obtained, he has to check his previous assumptions. If the checking fails, he has to modify the parameters according to his knowledge (certain design relations) toward the satisfactory direction. He may also have to change the parameters based on earlier assumptions. The dependency / iteration mechanism is illustrated in Figure 4.7.

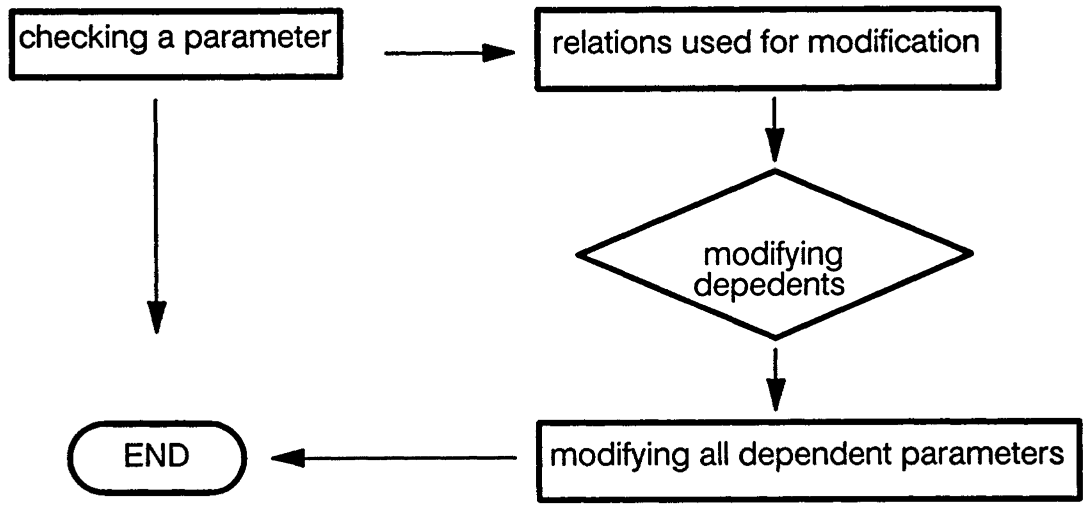
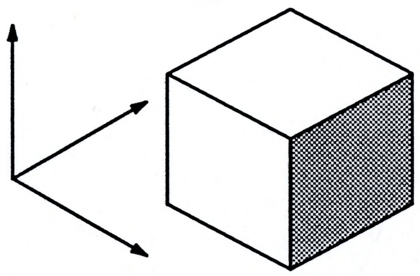

checking a parameter
relations used for modification
modifying depedents
END
modifying all dependent parameters

Figure 4.7 : The dependency / iteration mechanism

▪ Geometry Aspect

Thus far, only the parametric aspect at the analysis stage has been described, while the geometry aspect has not been mentioned yet. The geometry aspect in the routine mechanical design consists of relational geometry information and structural geometry information. Figure 4.8 illustrates the typical geometry information about a cube, which includes translational and rotational coordinates and three basic structure parameters.

Since the rough geometric structures of configurations have been decided at the synthesis stage, the geometry aspect in the routine mechanical design can be handled parametrically, provided that all basic functional design units are available. For instance, in the cam de-



structural information :
height, width, length

relational information :
trans–x0, trans–y0, trans–z0,
rot–x0, rot–y0, rot–z0

Figure 4.8 : Geometry information of a cube

sign, if all motion types and followers are stored in a library as basic design units, then they can be accessed by the parameters. In this work, these basic functional design units are stored in a library supported by a feature–based modelling system. This organization is illustrated in Figure 4.9.

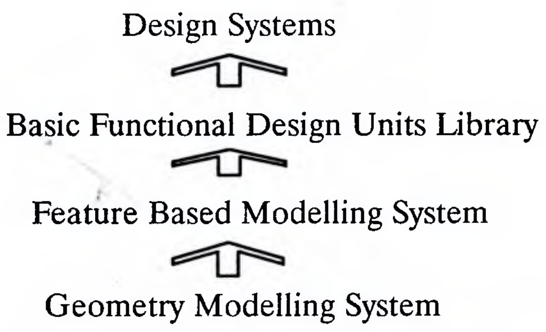

Design Systems
Basic Functional Design Units Library
Feature Based Modelling System
Geometry Modelling System

Figure 4.9 : An organization for the geometry aspect in the analysis

o Knowledge Representation Formats

The knowledge representation formats can be classified into four types: the basic parameter representation; the design relation representation; the organization of parameters according to configurations; and the representation of knowledge about the reasoning process, such as how to select the usable design relations, and the like.

The basic parameter representation can be handled by a string pair with two elements representing the name and the value. For example, "*(pressure.angle . 28.6)*" indicates a pressure angle.

The parameters are organized as the parameter set under each configuration. The following is a part of the cam parameter set:

```
(pressure.angle  .  28.6)
(Rb              .   5.0)
( ω              .   40π)
```

One important aspect in the design process needs to be mentioned here: the inheritance in the design process. The design of lower level sub-configurations can use parameters achieved in the upper level configurations. In this analysis model, the inheritance is realized naturally by the configuration tree.

Between the two kinds of design relations, numerical and non-numerical, the numerical category is further divided into various types of relations: equations, inequalities, tables, figures, and so forth. No matter what type it might be, a relation can be generally expressed as a "method" to obtain "unknown parameters" from "known parameters". This is obvious in the case of equations. For instance, the pressure angle relation (4.1) can be represented as:

Table 4.5: Equation in method

the known	*the unknown*	*the method to get it*
Vfmax, Rb,Yp,w	*press.angle*	*tan(press.angle) = Vfmax / (Rb + Yp) w*

Other types may not be as obvious as equations, but they certainly can be organized into this general format, provided that some changes are made in the "method" part. A natural idea is to create a number of "basic elements" corresponding to these various types. Later on, the "method" can be represented by organizing these basic elements in a certain order. For example, the design relation to determine the "rise program: pri" in the cam design can be expressed as:

"Find the cam factors for the rise part and the return part according to their motion types; Calculate "pri" by the equation:

pri = (rise.factor/(rise.factor + return.factor)) x total.program." (4.4)

This design relation can be represented by two basic elements: finding a value from a table; and calculating a variable from an equation (Table 4.6).

Table 4.6: Another method

the known	*the unknown*	*the method to get it*
rise.motion.type *return.motion.type* *tatal.program*	*pri*	*1. rise.factor and return.factor from cam factor table;* *2. pri = (shown in equation (4.4))*

The non-numerical design relations can also be described as a method in this format. For instance, the rule relation can be represented as one "basic element", so that the method simply incorporates this element whenever the rule is used.

A more complicated case occurs in the cam offset design:

"if the pressure angle is > 30.0, then reduce it by setting offset starting from 0.5, and use 0.1 increase each time until the requirement is met."

It has to be represented by two methods (Table 4.7).

On the other hand, this is also the representation of knowledge about the reasoning process, since the checking is invoked and some reasoning is involved. This cam offset is an example of the dependency / iteration process. Here, only the representation aspect is concerned, while the control mechanism has been explained previously.

Another issue is the representation of usable design relations. Now some formats have

Table 4.7: A more complicated case in two methods

method 1: checking relation

the known	*the unknown*	*the method to get it*
press.angle	*nil*	*press.angle < 30.0*

method 2: assigns the offset value if the checking fails

the known	*the unknown*	*the method to get it*
nil	*offset.e*	*start offset.e from 0.5* *increase 0.1 each time*

been established for the design relations. The next issue is how to organize these relations into groups of usable design relations, and how to choose them. A simple sequential "grouping" organization can handle this well. All relations are organized in "groups", with their "goal" parameters indicated. The mechanism to choose usable relations works by checking the current design status and the goal parameters.

The above several points have already been involved with the knowledge representation of controlling the reasoning process. Generally, the best possible effort should be made toward formatting the reasoning process control into methods, in order that more knowledge can be expressed explicitly.

The basic functional design units are supported by a basic design units library, which contains a number of standard units with their frames. Each frame has a number of parameters. The geometry modelling of these units will be handled by a feature–based modelling system.

4.5 NON–FUNCTIONAL CONSIDERATIONS

Basically, non–functional considerations can be expressed as design relations, and can be handled in the same way as those design relations used at the analysis stage. In other words, non–functional relations can be handled at a separate stage by using these relations as checking, or integrated into the analysis, which is an ideal situation of incorporating considerations at other stages of the product life cycle into the early design stage.

4.6 SUMMARY

Thus far, the whole design procedure and its major stages have been modelled. This model is general based on the common characteristics. Each stage consists of reasoning mechanisms and knowledge representation formats. The domain specific design knowledge is separated from these general mechanisms and formats, and thus, domain independence has been achieved. Furthermore, this model has potential to obtain the explicitly descriptive representation of design knowledge to facilitate the spreading of specific design systems.

As explained in the last chapter, the iteration between major stages, that is, the synthesis and the analysis, and the synthesis and the specification development, has not been dealt with here. Instead, a message is reported if such a case occurs. However, the iteration within the analysis stage occurs frequently, and is dealt with by using suggested design relations to modify the earlier designed parameters toward satisfactory direction.

CHAPTER 5 : IMPLEMENTATION

The implementation of the general model presented in the Chapter 4 will create a shell system. The main issue in this implementation is the "knowledge representation scheme", which consists of two basic parts: the general "reasoning machines" and the general "knowledge representation formats". The reasoning machines are responsible for the control of the design processes, and are implemented from the "underlying mechanisms of the design activities" established in the model. The knowledge representation formats are forms for expressing design knowledge, and are implemented according to the "knowledge representation formats" in the model. These two parts are inseparable, combined to form the knowledge representation scheme.

The model implementation is divided into two stages: the "conceptual implementation" and the "machine implementation". One deals with the implementation without considering the specific machine environment, the other implements the conceptual version on a particular machine. The advantages of so doing are: the achievement of machine independence of the implementation, and the simplification of the description of key ideas without the tiresome machine level details involved. In this chapter, the conceptual implementation is described first; then, the machine implementation.

The descriptive style for the development of the knowledge bases is emphasized during the implementation. The descriptive style is a methodology for knowledge base development which allows the knowledge to be represented in an explicit and descriptive manner, not in any algorithmic or programming style. For example, the knowledge regarding weather prediction can be expressed in the form of rules. It can also be described in an algorithm based on the calculation of a number of factors. The first case is presented in the descriptive style, and the latter in the algorithmic style. The purpose of achieving such a style is to minimize the cost and the time involved in the development, modification and maintenance of the knowledge bases. The descriptive style is the basic idea of expert system technology, which is also the best part adopted from it into this work. Although the reasoning machines cannot be totally realized descriptively, the effort toward it is still made so that changes and further development can be made quite easily.

One point that should be explained is that this implementation does not cover completely the whole model. The model is developed to pursue the understanding of design activities and to attempt to cover all aspects, while the shell is implemented to serve the practical application purposes. Thus, it may not be necessary to implement some parts of the model. In the following description, the parts not implemented will be mentioned with reasons. Figure 5.1 shows the relations between the understanding, the model and the implementation.

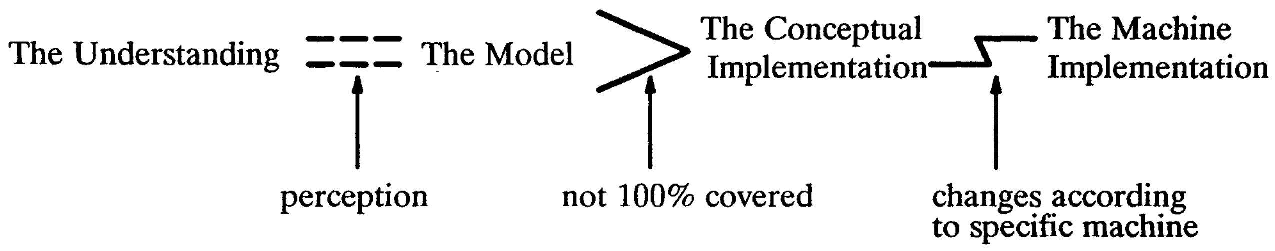

The Understanding
The Model
The Conceptual Implementation
The Machine Implementation
perception
not 100% covered
changes according to specific machine

Figure 5.1: Relations bewteen understanding, model and implementation

5.1 CONCEPTUAL IMPLEMENTATION

5.1.1 The Whole Design Procedure

Although the whole design procedure has been divided into four stages in the model, the implementation deals with only three stages: Specification Development, Synthesis and Analysis. The fourth stage — Non–functional considerations which are represented in design relations are incorporated into the analysis stage.

The organization of these three stages is in a loose form. The connection between them is realized by file transferring. This loose organization reflects the features of the design process to a certain degree and can be realized easily. Since the design problem is large and very complicated, the separation of major parts helps to simplify the problem (Figure 5.2).

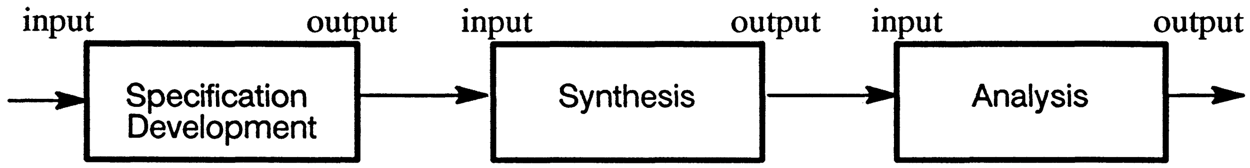


Figure 5.2: Organization of the whole design process -- file transfer

An important issue which should be mentioned is the feedback between stages. For example, the feedback from the analysis can be "design succeeds" or "design fails" with failure information provided. This kind of feedback is inevitable because the design is an iteration process. The feedback from the analysis may guide the redesign of the synthesis. Even the feedback from the synthesis stage may cause the re–specification. If the general design activities are considered, this feedback and its utilization are essential. However, since the scope of this work is the routine mechanical design activities, the re–structuring of all configurations from the failure of the detailed analysis is relatively unlikely. Moreover, this work is the first step toward the modelling of the design activities. Therefore, at this moment, the feedback from the lower stages is simply reported, but not utilized to guide the redesign of the upper stages. This is an example which shows that not all parts in the model are covered in the implementation.

The inputs and outputs of each stage are listed below, supplementing Figure 5.2:

Specification Development:

– input: the client original specification list

- *output: the standardized specification list and the principal function*

Synthesis:

- *input: the standard specification list and functions*
- *output: the configuration tree*

Analysis:

- *input: the configuration tree from the synthesis*
- *output: the configuration tree with parameter values, and drawings of the final product*

5.1.2 Specification Development

The specification development could be very sophisticated. However, as explained before, instead of the implementation of the complicated mechanisms behind its process, a simple alternative serves the purpose: creating <u>a standard list of specification items and principal function items</u>. Considerations are taken carefully at the time these standard items are selected, to ensure that the agreement between the client and the designer can be achieved, and that conflicts are unlikely to happen.

The extraction of standard specification and principal function items in a domain is done by the knowledge engineer consulting the experienced designers at the time of the domain design system development. Therefore, the first task in developing a specific system is to create such a standard list. The principal function items are filled with standard functions which are available in the knowledge bases. These standard functions have corresponding configurations to perform them.

After the standard list has been generated and the specific design system has been developed, it is the end user's responsibility to fit his specification items and principal functions into this list. Thus, certain "pre–specification" work is necessary. For example, if one of the original requirements for a cam in a sewing machine is *2.0″ per 0.05 seconds*, it has to be converted to the standard items: *angular velocity $2\pi/0.05$ (40π) and a rise of 2.0″.*

In summary, the implementation of the specification development stage does not create any physical parts to the prospective shell system. Instead, it simply provides an approach to the specification development: a standard list of specification and principal function items.

5.1.3 Synthesis

Once the standard list of specifications and the principal functions is created, the design proceeds to the synthesis stage. Based on the "synthesis design relations" which have three types: function–to–configuration relations; configuration–decomposition relations; and function–checking relations, all configurations are to be achieved. These configurations contain only structural information, without any values assigned to the structure attributes. For example, a cylinder may have two attributes: height and diameter. At this stage, no values for these two attributes are available, whether they should be (50, 40) or (200, 10). It appears to be an empty frame.

According to the model, the synthesis stage follows the Configuration Decomposition Approach (C.D.A.) considering the functional requirements. The C.D.A. works by a basic "four–step–cycle":

1. *recognize a function;*
2. *select a configuration according to this function;*
3. *decompose this configuration into sub–configurations;*
4. *find the functions of these sub–configurations, and ensure that these sub–functions fulfil the original function.*

This is illustrated in Figure 5.3.

The entire synthesis process consists of a number of these basic "four–step–cycle" patterns. This process continues until all the lowest level of configurations are basic functional

$$\begin{tikzcd}
\text{Function} \arrow[r] & \text{Configuration} \arrow[d] \\
\text{Sub . functions} \arrow[u, dashed] \arrow[r, dashed] & \text{Sub. configurations}
\end{tikzcd}$$

Figure 5.3: The basic “four–step–cycle” of C.D.A.

design units, which cannot be decomposed further.

The full description of the implementation should also include the organization of all configurations as well as functions, the basic functional design units and some other detailed issues.

The following description is organized under two major categories : the “reasoning machine” and the “knowledge representation formats”. The “knowledge representation format” is further divided into the representation of relations; the representation and organization of configurations and functions; and the organization of basic functional design units.

o The Reasoning Machine

• The Reasoning Machine is an algorithm following the C.D.A. It can be informally described as:

. *Recognize the principal function or a function;*

. *Obtain a configuration according to the function, based on the function–to–configuration relations;*

“iterative.point”:

. *Decompose this configuration into sub–configurations, based on the function–decomposition relations;*

. *Obtain all sub–functions of the sub–configurations, based on the function–to–configuration relations;*

. *Check whether these sub–functions can fulfil the original function, based on the function–*

checking relations;

. If all above steps return "success", then the design continues. Otherwise, it reports which step fails and other related information. For example, the failure can be caused by the "knowledge base being incomplete". The current basic four–step–cycle is abandoned, and a new four–step–cycle starts. If no new cycle is possible, then report failure of the synthesis and indicate the current configuration and function;

. If all above steps succeed, configurations are achieved and stored in the "configuration tree" which organizes them in a layered graph structure according to the decomposition process; and functions are in the "function tree" with a similar structure;

. Choose the next to–be–decomposed configuration guided by the width–first search. Find its function from the "function tree". Go back to "iterative.point";

. If no more configurations can be selected from the tree, in other words, all leaf nodes at this time are basic functional design units, the whole synthesis stage ends.

○ Knowledge Representation Formats

The knowledge representation formats are comprised of descriptive representation for: three basic relations; configurations and functions; and basic functional design units.

▪ The function–to–configuration relation:

Name: (ID of the relation)

Domain: (which domain this relation is in)

Function: (function string)

Configuration: (configuration string)

Conditions: (condition string)

Constraints: (constrain string)

For example, one function–to–configuration relation reads like:

Name: (cam–rel11.ftc)

Domain: (K.B.: cam1)

Function: (a–part–of–rising)

Configuration: (cycloidal–wedge)

Conditions: (no steps in acceleration, steps in jerk acceptable)

Constraints: (motion–pattern: cycloidal–motion)

- The configuration–decomposition relation:

  Name: (ID of the relation)

  Domain: (which domain this relation is in)

  Configuration: (configuration string)

  Sub–configurations: (configuration string)

  Conditions: (condition string)

  Constraints: (constrain string)

One example is:

Name: (cam–rel23.cts)

Domain: (K.B.: cam1)

Configuration: (cam)

Sub–configurations: (rise, conn.of.rise.dwell, dwell, conn.of.dwell.return, return, conn.of.return.rise)

Conditions: (motion–pattern: RDR)

Constraints: (nil)

- The function–checking relation :

  Name: (ID of the relation)

  Domain: (which domain this relation is in)

Function: (function string)

Sub–functions: (function string)

Conditions: (condition string)

Constraints: (constrain string)

For instance, a relation in the cam design :

Name: (cam–rel5.fts)

Domain: (K.B.: cam1)

Function: (rotation–to–translation)

Sub–functions: (rise, dwell, return)

*Conditions: (motion–*pattern: RDR)

Constraints: (nil)

It should be noted that it is convenient for these representation formats to handle the multiple–to–multiple relationship demonstrated in design relations. Since one object could correspond to several objects, the representation format must be flexible enough to incorporate as many and diverse relations as the real design domains can have. The "conditions" and "constraints" in these relations define the circumstances under which these relations are valid, and the corresponding side effects these relations bring in. They are optional which further enhances the flexibility of these representation formats.

▪ Representation and organization of functions and configurations

The basic definition of configurations and functions is necessary for the unification of these terms since there may be possible conflicts among these terms, and for the explanation which may be expected later. All these basic terms are organized in a dictionary with the structure:

Name: (a function or a configuration)

Domain: (specific design domain)

Text: (full explanation of this function or configuration)

Type: (one of "non–basic", "basic" and "both", meaning whether it can be further decomposed)

Following are some examples:

a configuration — (cam, K.B.:cam1, "cam in a cam system", non–basic)

a function — (rotation–to–translation, K.B.:cam1, "function for a cam–system", nil)

The configurations obtained during the synthesis will form a layered graph structure called the "configuration tree". Each configuration is a node in this graph structure. The basic structure of a node is as follows:

Node: (a configuration)

Super–nodes: (nodes in the upper level)

Sub–nodes: (nodes in the lower level)

Constraints: (generated in the synthesis stage and attached to this node)

Figure 5.4 is a sample configuration tree. One node from this tree: cam, is illustrated below:

Node: (cam)

Super–nodes: (cam–system)

Sub–nodes: (rise, conn.of.rise.dwell, dwell, conn.of.dwell.return, return, conn.of.return.rise)

Constraints: (Motion–pattern: RDR)

The "function tree" stores all functions generated in the synthesis process, and has the same structure as the configuration tree, excluding the "constraint" item. One function tree example is given in Figure 6.1.

Node: (a function)

Super–nodes: (nodes in the upper level)

Sub–nodes: (nodes in the lower level)

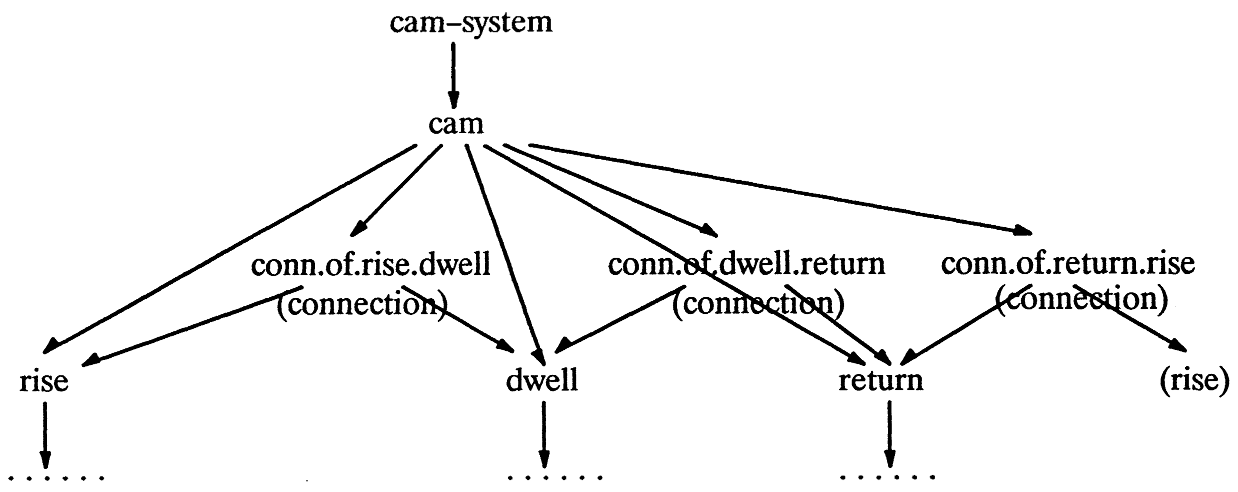
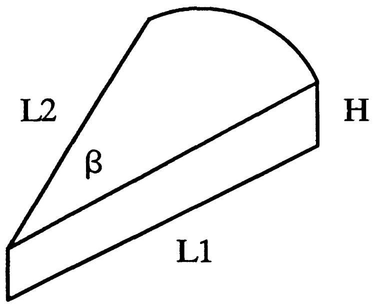


Figure 5.4: A portion of a sample configuration tree

- Representation of basic functional design units

The leaf nodes of the configuration tree are basic functional design units which cannot be decomposed further. They are organized in a library, like standard components, with basic functions they can perform. This library is supported by a feature–based modelling system. Corresponding to these basic functional design units in this library, there are "basic geometry features" in the feature modelling system. These basic geometry features have default frames of relational and structural geometry parameters. Once a basic functional design unit is selected from the library, a corresponding basic geometry feature is chosen from the feature–based modelling system, whose parameters will be filled when the analysis process is completed. For example, a basic functional design unit, *cycloidal wedge*, has the default frame shown in Figure 5.5.

In one of the example systems developed later, the cam design system, the basic geome-



name : cycloidal wedge

structural parameters :
H, L1, L2, β

relational parameters :
trans–x0, trans–y0, trans–z0,
rot–x0, rot–y0, rot–z0

Figure 5.5: Default frame for a geometry feature — cycloidal wedge

try feature library includes a cycloidal wedge, a simple harmonic wedge, and other types of wedges, as well as some common features such as a cylinder.

o Summary Of Synthesis

Thus far, the representation scheme at the synthesis stage can be summarized as follows:

. Reasoning machine: Configuration Decomposition Approach

. Knowledge representation formats include:

- Representation of relations;
- Representation of configurations and functions, organized in a dictionary;
- Organization of configurations and functions in a layered graph structure;
- Organization of basic functional design units in a library, supported by a feature based modelling system.

5.1.4 Analysis

At the analysis stage, all configurations with structural information have been achieved and transferred from the synthesis. The task now is to assign values to parameters in these "empty" structural frames. The parameters include dimensions, relational geometry dimensions, materials and so on.

Before the implementation of the analysis is explained, it is necessary to review the characteristics of the analysis summarized in the Chapter 4.

. Parameters are determined by design relations;

. A design relation is a relationship among a number of parameters, which could be a numerical relation or a text relation. Furthermore, a numerical design relation could be an equation, an inequality, a table, a figure, a default value and so on. A textual design relation usually can be expressed with conditions, constraints and certain consequences. Frequently, these two basic types are combined together; that is, numerical information in a textual design relation;

. The fundamental principal of the parameter solving process is the comparison of the number of unknown parameters and the number of design relations;

. Design relations are very large and greatly diverse, and certain organization is necessary;

. Parameters are organized according to configurations, and are handled along with the processing of configurations;

. It is critical to choose the "usable design relations" according to "solved parameters" and "to-be-solved" parameters;

. Design relations can be represented in "methods" and comprised of basic elements;

. Design relations are organized in "groups" in a certain order to be applied to the parameter solving process. There may be many ways to organize certain design relations to achieve the same set of parameters;

. The determination of parameters is an iterative process. Parameters of a configuration may be handled partially at the beginning and later iteration may be needed to complete them. The dependency / iteration process is a very important feature in the analysis stage;

. The geometry aspect is handled parametrically based on a library of basic functional design units;

. The descriptive style of representation should be emphasized.

The implementation of the analysis stage is based on the above points. The key issue here is how to organize these large and diverse design relations, so that they can be used efficiently and at the same time, most importantly, they can be expressed descriptively.

Considering the fact that the design relations are comprised of basic elements with a limited number of types, such as equations, tables and so on, it is quite natural to reach the idea that a design relation may be expressed by a group of "basic description elements", such as the calculation of a value from an equation, the retrieval of a value from a table, and the like.

By expressing design relations through basic description elements, it is convenient to organize one design relation from a pile of basic description elements to achieve one parameter, called a "design slice". Therefore, one design slice can determine one parameter through a procedure or method expressed in a set of basic description elements. This kind of design slice is very flexible to use, and serves the purpose of the descriptive style of knowledge representation. The design slices are the basic units needed to solve parameters, and later on, they will be grouped together to solve a set of parameters.

Based on the fact that design relations are used in groups and that there are many ways to form these groups, the concept of "design procedure" is formed. A design procedure is a pile of design slices. Because there are many ways to organize the design slices and this organization is an essential part of design knowledge, it is necessary to devise the design procedure as one level of knowledge organization. For instance, there are two groups of design slices, both of which can determine a group of parameters. It is necessary to make both of them available for selection according to the current circumstances. Moreover, it may be possible that one of them may apply to other groups of parameters.

Since the parameters are organized under configurations and the solving process of parameters advances by configurations, it is natural to organize the design procedures under configurations. Each configuration is attached to a group of design procedures with their application conditions. Moreover, the variation in the decomposition of configurations results in various sets of parameters. Therefore, it is quite reasonable to set up a configuration organization level. These configurations are termed "knowledgeable configuration units", meaning they are attached to design knowledge.

Thus far, four basic knowledge representation levels have been created. Figure 5.6 shows the relations between them, where the lower levels support the upper levels.

$$\begin{tikzcd}
\text{Knowledgeable Configuration Units} \\
\text{Design Procedures} \arrow[u] \\
\text{Design Slices} \arrow[u] \\
\text{Basic Description Elements} \arrow[u]
\end{tikzcd}$$

Figure 5.6: Organization of analysis knowledge representation formats

Thus far, the basic ideas of the implementation of the analysis stage have been described. The entire representation scheme is descriptive, down to a standard set of basic description elements. These standard basic description elements can be organized easily to form various design slices, design procedures and knowledgeable configuration units. The standard basic description elements can be general to all design domains, or they can be organized according to specific domains. In this case, each domain has its own set of basic description elements. Furthermore, standard methods to develop these basic description elements should be provided, so that the set of these elements can be enlarged easily to cover possible new cases in some design activities. These new cases could be new design knowledge types, which are not covered so far. This way, the entire design knowledge scheme is not only descriptive for facilitating the spreading of knowledge, but is also flexible for further development.

The above sections handled only the knowledge representation formats, which can be developed quite simply based on the above ideas. However, the reasoning machine needs some organization, since the whole reasoning process is quite complicated. An efficient way to deal with this kind of problem is to decentralize the control or reasoning process into a number of reasoning machines, called "interpreters". Each interpreter deals with "a local problem", under the guidance of a general interpreter. For instance, there are interpreters for various types of basic description elements, an interpreter for design procedures and so on.

So far, the basic ideas about the knowledge representation scheme in analysis have been explained. The following is a description of the detailed implementation under the two basic aspects of knowledge representation scheme: reasoning machine and knowledge representation formats.

o Reasoning Machine

The reasoning process control is fulfilled by a set of multiple level, distributed reasoning mechanisms, called "interpreters".

. The "general interpreter" performs the basic iterative cycle of choosing a configuration and determining its parameters;

. The "configuration selection interpreter" chooses a configuration for design by considering the unsolved configurations in the configuration tree, and matching a knowledgeable configuration unit in the analysis knowledge base. A knowledgeable configuration unit contains design procedures and possible patterns of decomposition;

. Since the solving process of parameters is performed by design relations which are expressed in design slices and organized in design procedures, it is necessary to devise a "design procedure selection interpreter" to select a suitable design procedure according to the "known" and "unknown" situations in the parameter set and up–to–date design status;

. There are two types of design slices: "solving slice" and "checking slice". A solving slice is used to determine a parameter, while a checking slice verifies parameters according to the dependency / iteration model. Correspondingly, there are two interpreters for them: the "solving interpreter" and the "checking interpreter";

. Each basic description element has an interpreter. For example, a "calculation" interpreter handles the "calculation" element for determining a value from an equation, a "rule reasoning" interpreter deals with the "rule" element. The whole set of elements is described later in Knowledge Representation Formats section;

. When a checking fails, the "modification interpreter" is called on to change some parameters, which is in fact a dependency/ iteration process. It follows a "failure design procedure" which consists of a number of design slices to modify some parameters to satisfy the checking, and to change dependent parameters as well.

This set of multilevel, distributed control mechanisms reflects the complexity of design analysis process. Thus, it is powerful enough to handle the whole analysis process, and at the same time, it incorporates the flexibility for further development.

○ Knowledge Representation Formats

As described above and shown in Figure 5.6, there are basically four levels of knowledge representation formats:

▪ The Basic Description Elements so far have the following types with their standard formats, (key words are underlined):

◇ search a solution from an equation (or calculation)

(cal goal expression)

◇ look up a table for a parameter

(look-up-table table-name known goal)

◇ choose a parameter according to some premises

(rule goal known rule-set)

while in the rule sets, there are a number of rules in the form of :

if : premises

then : conclusions

◇ determine a parameter by experience

(default goal value)

◇ round off a number using some strategies, e.g. round a calculated rise program "pri = 108.43" into an integer 108^o .

(round goal round-strategy)

◇ solve a parameter using a constraint

(by.constraint goal constraint.name)

while constraints are in the form of :

(constraint.name goal known relation)

◇ determine a parameter from an initial value by an increase

(inc parameter initial.value increase)

◇ find a parameter's two nearest values from a data table

(interpolate table.name object.parameter.list the.parameter key.item conditional.items)

For example, a calculated cam factor 3.246 with cycloidal motion type determines two nearest cam factors from Table 6.10 as 3.46 and 2.86 at 30° and 35° pressure angles, through:

(interpolate 'cam.factors '(cam.factor) 3.246 press.angle '(motion.type cycloidal))

This Basic Description Element set can be easily extended. At each expansion, a new format is created and a corresponding interpreter is developed accordingly. No change to the other parts is required. This gives great flexibility for further development.

Some examples of Basic Description Elements are:

```
(cal (rise.c.f / (rise.c.f + return.c.f)) x remaining.prog)
(look-up-table 'cam.factors '((pressure.angle . φ) (motion.type . rise.motion)) 'cam.factor)
(round pri (strategy . nil))
(by.constrain re.l1 constraint.d.re)
     (constraint.d.re (re.l1 re.h1) (dw1.l2 dw1.h2) ((dw1.l2=re.l1) (dw1.h2=re.h1)))
(rule pd1 (nil) rule-set1)
     in "rule-set1", there is one rule:
          if : (a small dwell is required), and
               (this is at the beginning);
          then : give it a (10)° program
(inc offset.e 0.5 0.1)
```

▪ Each Design Slice defines a "goal" parameter and a design relation to solve it. There are two types of design slices: "solving" slices determine parameters and "checking" slices verify some goal parameters to ensure that these parameters meet certain relations. If the checking succeeds, the design continues. Otherwise, a failure design procedure is called in to modify the previous design, including dependent parameters. Their formats are:

◇ Solving Slice :

Name : name of this slice

Goal : the parameter to be designed

Known : circumstances under which this slice works

How–to–get–it : a sequential collection of basic elements to achieve the goal.

For instance,

Name : solving–ri.ss

Goal : pri (program of rise part in a cam)

Known : (ϕ rise.motion, return.motion, remaining.prog)

How–to–get–it : 1. *rise.c.f= (for–table 'cam.factors '((pressure.angle . ϕ) (motion.type . rise.motion)) 'cam.factor)*

2. *return.c.f= (for–table 'cam.factors '((pressure.angle . ϕ) (motion.type . return.motion)) 'cam.factor)*

3. *pri = (cal (rise.c.f / (rise.c.f+return.c.f)) x remaining.prog)*

4. *pri = (round pri (strategy . nil))*

◇ Checking Slice :

Name : name of this slice

Check.goal : the parameter to be verified

Known : circumstances under which this slice works

Checking relation : a design relation the parameter is supposed to comply with

Failure design procedure : the design procedure called in when failure happens

For example,

Name : checking–ϕ.cs

Check.goal : ϕ (pressure.angle)

Known : nil

Checking.relation : $\phi < 30.0^o$

Failure.design.procedure : check.ϕ.dp

▪ The "Design Procedure" defines the order of design slices used. Since there may be a number of design procedures to solve the same set of parameters, or the same design procedure might be used for different configurations and their parameters, it is devised as a representation level instead of being attached to the "knowledgeable configuration units". Its format is designed as:

Name : name of the procedure

Domain : (which design domain it belongs)

Text–of–slice : a collection of design slices

For instance, a design procedure in the cam design case :

Name : program.dp

Domain : (K.B.: cam1)

Text–of–procedure : (solving–d1.ss solving–d2.ss solving–ri.ss solving–re.ss solving–r.p.ss solving–q.ss checking–q.cs)

▪ The "knowledgeable Configuration Unit" is the uppermost level in this representation. It includes a configuration and its possible patterns of decomposition, the parameter set and a number of design procedures. The following structure is devised:

Name : the name of the knowledgeable configuration unit

Configuration : configuration name

Sub–configurations : list of sub–configurations

Parameters : parameter set

Design procedure : the name of the design procedure

An example of a knowledgeable configuration unit:

Name: cam

Configuration: (cam)

Sub–configurations: (dwell1 conn.d.ri rise conn.ri.d dwell2 conn.d.re return conn.re.d)

Parameters: (pd1 pd2 pri pre offset.e r.p)

Design–procedure: (((angular.velocity dwell.time rise.motion.type return.motion.type radius.base d.rise radius.follower) . program.dp) . . .)

The configuration tree of the analysis inherits the structure from the synthesis stage. It stores the values of all parameters achieved in the analysis. Thus, the transformation from an "empty" tree to a "full" tree is completed. This tree is the output from the analysis stage. Final printout and graphics output are extracted from this configuration tree.

The iterative process characterizes the design process at the analysis stage. This occurs when parameters of some configurations cannot be determined at once. It occurs more frequently when some parameters are verified and the dependency / iteration model is called in. In this case, if a checking is not satisfied, the failure design procedure modifies the earlier design and dependent parameters. Each modification may cost several iterative processes.

The inheritance is intensively used in the achievement of parameters. When determining the "known" situation for solving a configuration in a certain level, the system uses parameters in its upper level configurations by inheritance.

The leaf nodes or the basic functional design units in the configuration tree are organized in a library, and supported by basic geometry features in a feature–based modelling system. Once the analysis process and the values of parameters are completed, graphic display of the product can be easily generated.

o Summary Of Analysis

The reasoning machines and knowledge representation formats presented so far form a multilevel representation method which is powerful enough to handle the large and diverse amount of analysis knowledge. The descriptive way to represent knowledge and develop specific design knowledge bases has been stressed and realized. The extendibility of the types of design knowledge formats is obtained by adding new interpreters and new formats in this distributed representation scheme. Therefore, this descriptive representation approach to design knowledge is effective in handling the complicated analysis process, and at the same time, is flexible enough to allow for further development.

5.2 MACHINE IMPLEMENTATION

First, the machine environment is briefly introduced. Then, the implementation at the machine level is described, covering issues like the concrete representation formats, feature modelling and the user interface, which were skipped in the conceptual implementation. Finally, the procedure to develop specific design systems from the shell is presented.

5.2.1 The Software And Hardware Environment

In this work , the Knowledge Engineering Environment shell -- KEE and Common LISP on a SUN workstation were chosen as the software and hardware environment. KEE is an object-oriented programming tool with rule reasoning facilities. It is a very powerful supporting environment for knowledge engineering research. However, this does not mean that KEE can be used solely for developing expert systems.

Before further description proceeds, a brief introduction of KEE is necessary to lay some ground for the machine implementation description. Detailed KEE introduction can be found in the KEE manuals [39].

<u>A Brief Introduction To KEE</u>

KEE is an object-oriented expert system development tool. It is an integrated set of programming facilities (Figure 5.7) [71] that helps software developers to build knowledge

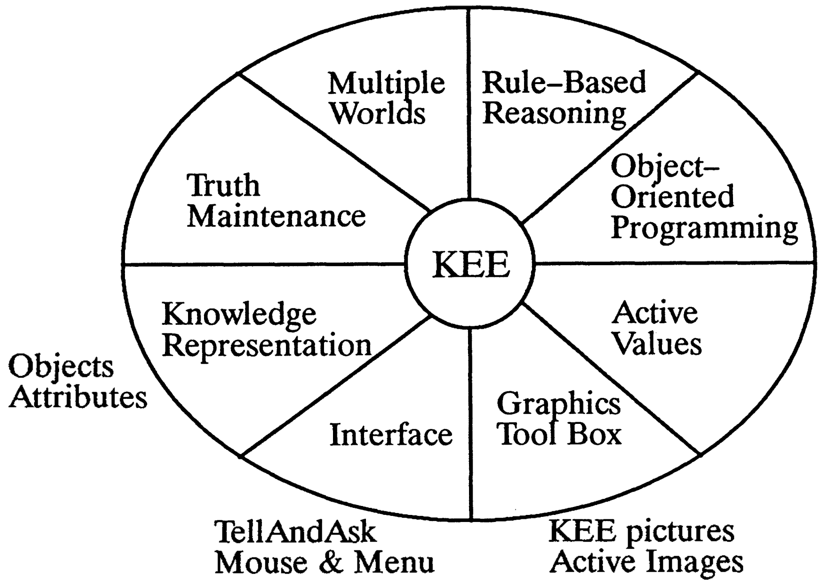


Figure 5.7: The KEE programming facilities

based systems in many application domains. Of this variety of programming facilities, the knowledge representation is the core. The object–oriented programming and rule–based reasoning are two major tools for applications. The KEE software is written in Common LISP programming language which is a conventional Artificial Intelligence and Expert System development language.

o Knowledge Representation:

Information from any application is organized into objects called "units" in KEE. There are two kinds of connections among units: "class–subclass" and "class–member". Each unit has a number of attributes called "slots" which store values for these attributes. A slot has a number of properties called "facets", such as an inheritance facet, a value class facet, and so on. This knowledge representation structure is the core of KEE. It is proven to be convenient for knowledge base development.

o Object–Oriented Programming:

Object–oriented programming is an effective programming technique for application problems. It is fully supported by KEE. The "Method" is a piece of program about the behavior of a unit, and is attached as a slot. By passing "Messages" among units, methods can be invoked to achieve problem solving. These messages ultimately control the behavior of objects and, thus, two types of control are possible: "centralized" and "distributed" control.

o Rule–Based Reasoning:

Another important tool in KEE is the rule–based reasoning. Rules offer a mode of reasoning in knowledge–based systems, which is typical of an expert system shell. They are effective in handling dependencies among facts. Rules are represented as IF — THEN format, meaning certain actions take place under specified conditions, and are organized under "rule classes" in KEE. There are a number of reasoning methods. KEE offers basic forward–chaining and backward–chaining, and further development of reasoning is possible.

o Other Features:

There are three ways of interacting with KEE : "the mouse–and–menu interface" which is commonly used at the development and debugging stage; "programmatic KEE user functions" which are similar in form and style of use to the commands in Common LISP, and thus can be integrated and executed in programs; "TellAndAsk language" provides an English–like syntax language to interact with KEE.

KEE provides a variety of tools for user interface development. The "Window System" is the basic ground of all user interface tools. “Active Images” provide a convenient and vivid means of indicating the reasoning status of knowledge bases.

“Active Values” are another way of giving objects behavior, commonly referred to as “Watchdogs” or “Demons”. When using an active value, a behavior is made an inherent part of the slot and this behavior executes automatically under pre–specified conditions.

“KEE Worlds” are a set of facilities provided by KEE for modelling and exploring different hypothetical situations that might arise in a knowledge base. A world represents an alternative state of a knowledge base or knowledge bases.

The “Truth Maintenance System (TMS)” is a facility provided by KEE for setting up and maintaining dependencies between facts. It is also called the “Assumption–based Truth Maintenance System (ATMS)”.

Summary

From the machine environment point of view, the organization of this implementation can be illustrated as Figure 5.8. KEE supports the shell system, which further supports the specific design systems.

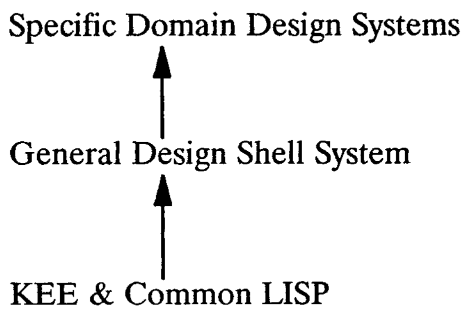


Figure 5.8 : Organization of the machine implementation

One issue that needs to be addressed here is that the machine implementation does not have to be based on KEE or any other expert system development tools. The conceptual implementation could be realized by any programming facility, such as C language. Nevertheless, KEE provides some features that facilitate the expert system development. Following are some of them which have been used in this work:

o KEE has the rule format and corresponding reasoning facilities;

o KEE provides the object-oriented programming style, which organizes the knowledge base in objects and connects them by sending messages;

o Despite the above two major features, KEE also supports:

- Active Value, which is a convenient tool for implementing the "watchdogs" functions, which are usually used in the expert system development;

- Active Images and graphics plus the window system, which helps the user interface design.

5.2.2 Description Of Machine Implementation

The machine implementation basically follows the approach presented in the conceptual implementation. Nevertheless, it is adapted to the specific KEE and LISP environment. Changes due to the machine level issues will be mentioned, together with the KEE and LISP features which cause these changes.

The machine implementation description is organized in the same way as the conceptual implementation. Only those parts concerned with how to implement the proposed conceptual version on KEE are briefly described, while the repeating parts already explained in the conceptual version are skipped. Detailed implementation examples such as specifications, three basic synthesis relations, various design procedures and slices can be found in the Appendix.

The Whole Design Procedure and Specification Development

All specification items are stored in the slots of a special unit — "specification". The synthesis accepts this specification unit, and outputs the configuration tree which is represented in a number of units with unit–subclass structure. This configuration tree is transferred to the analysis stage. Then, the design proceeds to the end, until the analysis stage outputs its configuration tree which can be directed to a file and used for graphics output.

Synthesis

In both synthesis and analysis, the machine implementation of the knowledge representation scheme follows the scheme proposed in the conceptual implementation, consisting of two parts: the reasoning process and the knowledge representation formats.

o The Reasoning Process is implemented in LISP using methods, which roughly follows the algorithm presented in the section 5.1.3: the C.D.A.

o Knowledge Representation Formats are devised in similar formats presented in the last chapter.

▪ Function–to–configuration relations are organized into units with slots described in the section 5.1.3.

▪ Configuration–decomposition relations are organized into rules supported by KEE rule facilities:

Name–of–rule : ID of the rule

Domain : defined by rule class

Premises : conditions + configuration

Consequences : constraints + sub–configurations

▪ Function–checking relations are organized into rules:

Name–of–rule : ID of the rule

Domain : defined by rule class

Premises : conditions + sub–functions

Consequences : constraints + function

▪ The function and configuration dictionary is organized into a unit with sub–units as many as configurations and functions can have. Each unit has slots accommodating items such as name, type, explanation and some optional properties (refer to the section 5.1.3).

▪ The configuration tree and the function tree are represented by a unit–subclass structure. Inheritance is realized by defining the proper inheritance properties.

▪ The basic functional design units are organized in a pool of units. Each unit has its name, and its unique slots. (Sometimes there might be common slots among units). These slots represent their parameters.

Analysis

○ The Reasoning Process:

Interpreters are programmed in LISP following the requirements for every interpreter listed in the section 5.1.4. Methods and active values are extensively used to realize the distribution of the control process, in order to achieve the multiple level distributed control mechanisms in the analysis.

○ Knowledge Representation Formats:

▪ "Knowledgeable Configuration Units", "Design Procedures" and two types of design slices are represented in units with slots for the items described in the section 5.1.4.

▪ "Basic Description Elements" are implemented directly from those described in the section 5.1.4.

Basic Functional Design Units and "IPDM"

The basic design units are organized in a library and supported by a feature–based modelling system. This feature–based modelling system is a part of a system called IPDM which has been developed by McMaster University FMRD (Flexible Manufacturing Research and Development) Centre. IPDM stands for Intelligent Product Design and Manufacture. It intends to facilitate the integration of mechanical product design and manufacturing, design for manufacturing, generative task planning, assembly planning and so on. A detailed introduction can be referred to in references [26, 34 & 43]. Here, only parts relevant to this project are described.

The feature-based modelling system provides a geometry features library and a Product Description Language (PDL). A product is described by PDL which calls geometry features in the library to model the product. PDL provides a three level description: product, component and feature; and a number of feature attributes, such as specifications, constraints, material, operation and so on. The geometry feature library has a number of geometry features extracted from various mechanical product design domains. This feature library can be easily extended to cover new domains. For instance, it has been expanded, for the purpose of this project, to incorporate wedges used for various motion types in the cam design domain.

The basic functional design units are organized in a library with slots for default frames. For each basic design unit, the IPDM system provides a corresponding geometry feature. For example, a cycloidal wedge in the basic functional design unit library has a corresponding geometry feature in the IPDM with the same or similar structure.

The interaction between the basic design units library and the IPDM is achieved by an interface program. Therefore, when a basic design unit is generated with its parameters in the design system, its corresponding geometry feature is produced by the interface program. This is illustrated as Figure 5.9. The output file of all basic functional design units from the design system is transferred to an input file for IPDM in PDL format (Product Description Language) by the interface program.

For example, two files from the cam design system are shown in Figure 5.10 (Only parts of the files are illustrated, and the complete ones are attached in the Appendix). It should

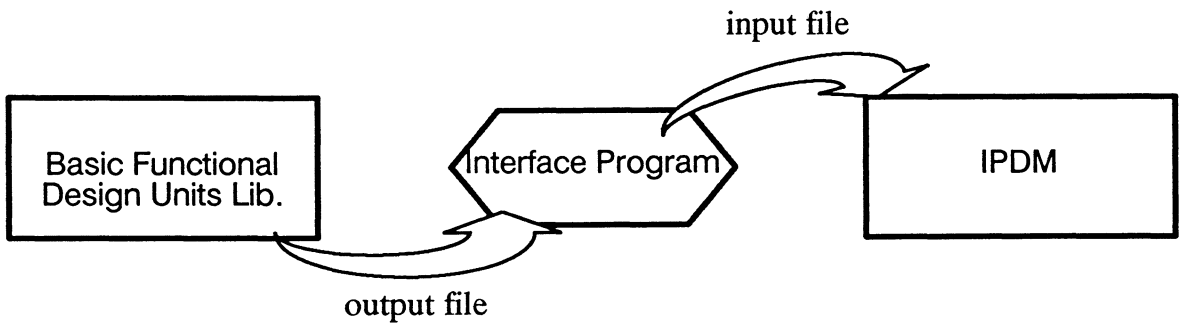


Figure 5.9 : Basic Functional Design Units Library and IPDM

be noted that non–geometric parameters are not transferred to the PDL file, and the hierarchy of configurations in the PDL file is derived from the configuration tree in the analysis.

User Interface

User interface of the shell system is developed by using active images and the window system. The screen is split into several parts with windows showing the design status. Figure 6.4 and 6.11 show a synthesis window and an analysis window from the cam design system. Besides some general icons, the synthesis window mainly includes the "current function", "current configuration", "sub–functions" and "sub–configurations", illustrating the basic four–step synthesis cycle; and the analysis window consists of "current configuration", "current configuration parameters", "current design procedure", "current design slice" and checking–related information, exposing the current ongoing analysis details.

Summary

A prototype shell system for the routine mechanical design activities has been developed based on the general design model and the conceptual implementation. It does not contain any specific domain knowledge. Features of this shell system can be summarized as:

. *The shell system is mechanical design process–oriented;*

. *Design knowledge bases can be developed descriptively;*

. The whole design process is characterized by three stages, loosely linked by file transfer;

. The shell system is based on KEE and Common LISP, and supported by a feature–based modelling system.

File output by the basic functional design units library

```
configuration: CAM-SYSTEM
   pressure.angle: 28.9
   angular.velocity: 125.66


configuration: CAM
   pd1: 10.0
   pd2: 108.0
   pri: 135.0
   pre: 107.0
   offset.e: 0.5


configuration: SHM.return
   s : 3.25
   beta : 108.0
   d : -2.0
   h : 5.0
   x0 : 0.0
   y0 : 0.0
   z0 : 0.0
   a : 0.0
   b : 0.0
   c : 243.0
```

PDL file

```
PRODUCT cam-system
{
  component: cam;
  component: follower;
  . . . . . .
}

COMONENT cam
{
  feature: dwell1;
  feature: rise;
  feature: dwell2;
  feature: return;
  operation: (dwell1 + rise + dwell2
             + return)
}

FEATURE return
{
   type: SHM
   orientation: {
                 movx = 0;
                 movy = 0;
                 movz = 0;
                 rotx = 0;
                 roty = 0;
                 rotz = 243;
                }
   parameter:  {
                 l1 = -2.0;
                 l2 = 5.0;
                 l3 = 3.25;
                 a1 = 108.0;
                }
}
              . . . . . .
```

Figure 5.10 : Comparison of a basic design unit file from the design system and IPDM

5.2.3 How To Develop A Specific Design System

The procedure to develop a specific design system can be illustrated in the following points:

$\boxed{1}$☞ Obtain an original “designer’s version” of design knowledge in a specific domain in which a specific design system is to be developed. It is a good idea to get this version based on some design examples. This version should be obtained from or verified by experienced designers, based on standards, codes and other documents.

$\boxed{2}$☞ Re–organize and rearrange the knowledge in the designer’s version according to the shell knowledge representation scheme to make it easier to express. This results in the “shell scheme version” of design knowledge.

$\boxed{3}$☞ Make copies of the “synthesis shell” and the “analysis shell”. Based on these two shells, the specific synthesis knowledge base and the analysis knowledge base are developed.

$\boxed{4}$☞ Create a “list of specification items and principal functions” based on the synthesis and the analysis knowledge bases.

$\boxed{5}$☞ According to the synthesis knowledge representation formats, develop the synthesis knowledge base from the “shell scheme version”.

$\boxed{6}$☞ According to the analysis knowledge representation formats, develop the analysis knowledge base from the “shell scheme version”.

$\boxed{7}$☞ Debug the two knowledge bases, by loading the “shell knowledge base”, which contains some general facilities for conducting the running process.

This procedure shows rough steps in developing a specific design system from the shell system. (In Figure 5.11, the numbers together with the arrows show the sequential steps in the development procedure listed above.)

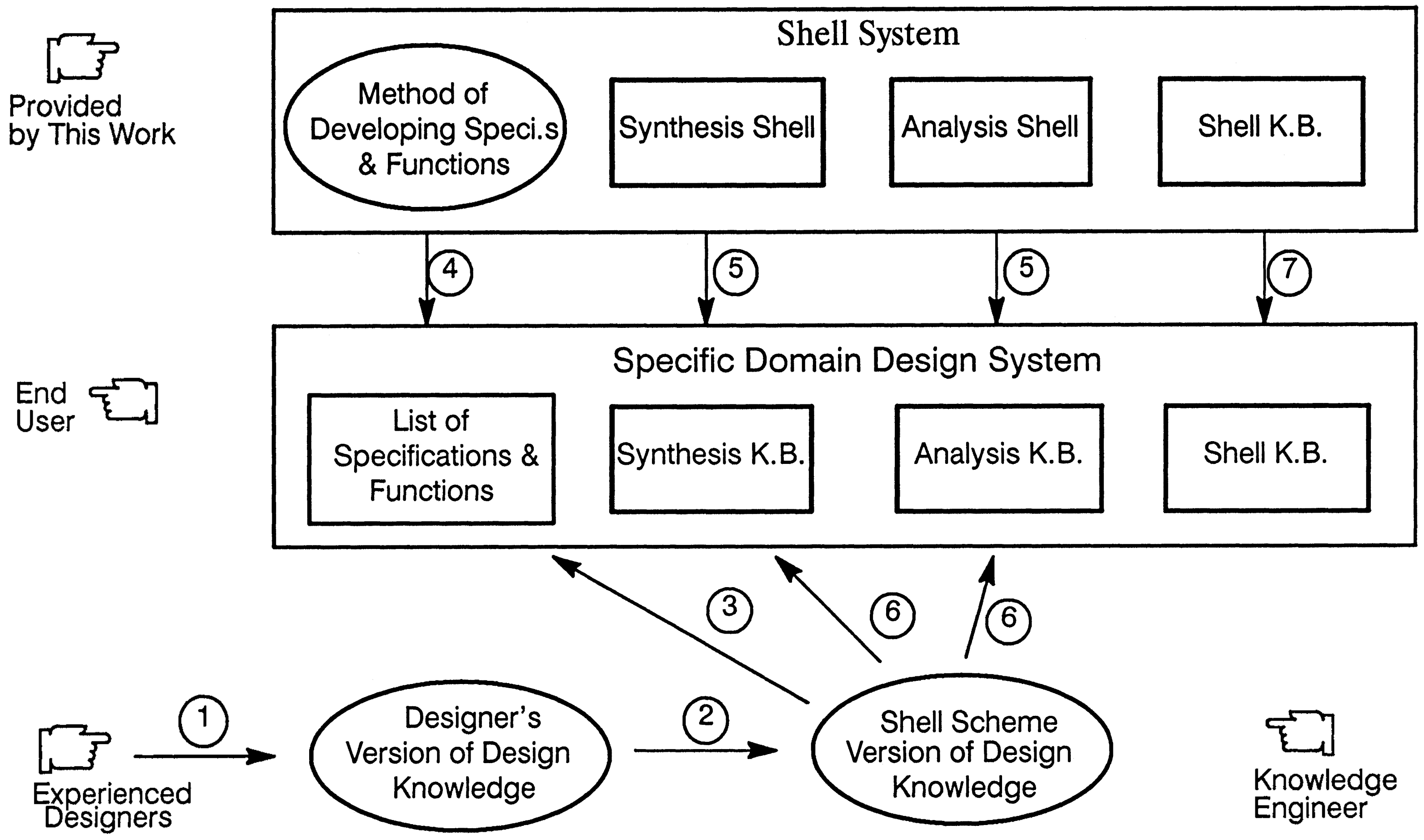

Shell System
Method of Developing Speci.s & Functions
Synthesis Shell
Analysis Shell
Shell K.B.
Provided by This Work
4
5
5
7
Specific Domain Design System
End User
List of Specifications & Functions
Synthesis K.B.
Analysis K.B.
Shell K.B.
3
6
6
1
2
Experienced Designers
Designer's Version of Design Knowledge
Shell Scheme Version of Design Knowledge
Knowledge Engineer

Figure 5.11 :Procedure to develop a specific design system from shell system

CHAPTER 6: SPECIFIC DESIGN SYSTEMS

In this chapter, two domain specific design systems developed based on the shell, the cam system and the bolted flange system, are described. The purpose is, of course, not only to develop two design packages, but also to show that the shell does help to construct domain specific systems, by implementing the above two examples in totally different design domains.

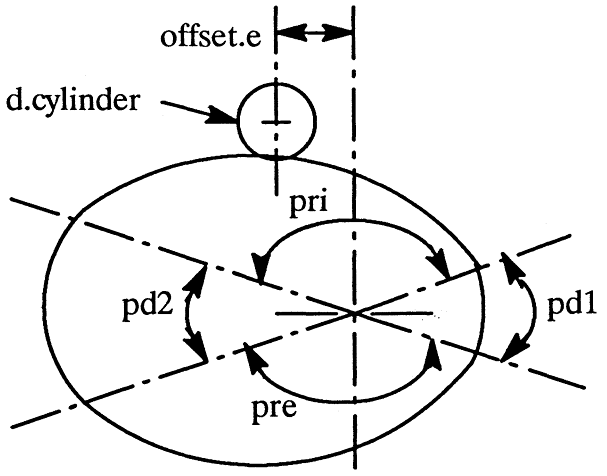
To compromise between proving the feasibility of the shell approach by real design cases and avoiding the time consuming details at this experimental stage, the scope of these design processes is imposed (Table 6.1 and 6.11 illustrate the scopes of both systems). It should be noted that this knowledge base scope does not mean the representation scheme is limited. Once the typical features in the design process have been included, further development of the system is a simple knowledge accumulation process without any change in the representation scheme.

6.1 CAM DESIGN SYSTEM

6.1.1 Development

Objective and Scope

A typical routine mechanical design process, the cam design, is chosen as the first example (Figure 6.1). The decision is made based on the fact that the cam design process is not too involved in details, but has relatively complete features which characterize the overall mechanical design process. (Here, a cam is regarded as consisting of several sections: rise, dwell, return and the like, and a special connection between them comprised of line partial configurations. Each section is selected based on a variety of motion types). Nevertheless,



PARAMETERS :

pd1	dwell1 program
pri	rise program
pd2	dwell2 program
pre	return program
offset.e	offset
R.P	radius of pitch circle
x0,y0,z0	translation of x, y, z
A,B,C	rotation of x, y, z
press.angle	pressure angle
S	starting length of one part
beta	total degree of one part
d	total rise of one part
start.lxy	connection edge length e.g. start.l12 is part2 length of connection1(between dwell1&rise)
H	cam width
R.F	radius of follower
H.F	follower width

CONFIGURATIONS :

cam–system, consists of
- follower
- cam, consists of
  - rise section
  - dwell1 section
  - return section
  - dwell2 section
  - connection between dwell1 & rise
  - connection between rise & dwell2
  - connection between dwell2 & return
  - connection between return & dwell1

Figure 6.1 : A cam sketch

there is a broad variation in specifications, motion types, possible configurations, selection from data tables and so on, even in a small piece of the design process. Further restriction of this knowledge abundance is necessary. Table 6.1 shows the current system scope. Beyond this capacity, the system will display: “knowledge base incomplete”.

The design knowledge is based on references [55, 65 & 74]. From now on, the development follows step by step the procedure proposed in the Chapter 5.

Table 6.1 : Scope of Cam Design System

bold -- implemented
italic -- not implemented

motion types :

cycloidal, simple harmonic, parabolic and uniform

modified trapezoidal, polinomial, double harmonic and the like

one part :

consisting of single motion type

consisting of sub–parts of various motion types

follower type :

roller type, not detailed

other types; no standard follower is selected

data tables & multiple choices :

a portion of them

not complete

For example: only a few rows of the cam factor table (Table 6.10)are implemented

"Designer's Version".

An efficient way to describe the design knowledge is to follow some examples and write down all "steps" that designers go through and "means" that designers use to achieve the results. The cam "designer's version" has been listed in the section 3.6 (from "problem statement" to "checking undercutting"), which is summarized from an example given in [65].

"Scheme Version".

The above "designer's version" is the written form of the human designer's working process. Arrangements are necessary to fit it into the knowledge bases formatted by the proposed representation scheme. It can be better understood as "trying to think in the representation scheme way". The three stages are as follows:

o Specification Development

The following specification list is created to incorporate all items encountered so far. Nevertheless, it is expandable (Table 6.2).

Table 6.2 : Cam Specification List

item : CYCLE.TIME *(the time for a cycle)* ;
item : R.B *(radius of base circle)* ;
item : ANGULAR.VELOCITY *(angular velocity)* ;
item : RISE.SUGG.MOTION *(suggested rise motion)* ;
item : R.FOLLOWER *(radius of the follower)* ;
item : RISE.TIME *(the time for rise motion)* ;
item : RISE.REQ *(the requirements for rise motion)* ;
item : RISE *(the rise, e.g. 2")* ;
item : DWELL2.SUGG.MOTION *(suggested motion for upper dwell)* ;
item : DWELL1.SUGG.MOTION *(suggested motion for lower dwell)* ;
item : DWELL1.TIME *(the time for lower dwell)* ;
item : DWELL1.REQ *(the requirement for lower dwell)* ;
item : DWELL2.REQ *(the requirement for upper dwell)* ;
item : DWELL2.TIME *(the time for upper dwell)* ;
item : RETURN.TIME *(the time for return)* ;
item : RETURN.REQ *(the requirement for return)* ;
item : RETURN.SUGG.MOTION *(suggested motion for return)* ;
item : PRESS.ANGLE *(pressure angle)* ;
item : MOTION.PATTERN *(motion pattern, e.g. DRD)* .

o Synthesis

The principal function can be defined as converting motion from "rotation to translation".

A cam system can serve the purpose of converting rotation to translation, which consists of a cam and a follower.

The motion pattern can be decided as "DRD" or Dwell–Rise–Dwell–return–dwell from the specifications. There are four parts in a cam with "DRD" pattern: dwell1, rise, dwell2, return.

Since no steps in acceleration are allowed in the rise, but steps in jerk are acceptable, the cycloidal motion can be chosen for the rise motion. Since there are no specific requirements for the return, SHM (Simple Harmonic Motion) can be selected for the return.

A disc follower is selected by the designer; this is the type commonly used if there are no special requirements.

○ Analysis

Unlike synthesis, the analysis part (from "☞ continue" to the end) in the "designer's version" basically follows the design process step by step, which can be represented naturally by the representation scheme. Therefore, no further arrangements are necessary.

Synthesis Knowledge Base Development

○ Make a copy of the "synthesis shell" and rename it as "cam synthesis knowledge base". This shell includes some general structures, such as the three basic relations. These structures will be illustrated by later examples; a list of them is given in Table 6.3.

○ Define the terms for configurations and functions, and create the tree structures for them. This is a key part in the knowledge base development. These terms are the basis of the knowledge representation, while the trees set the configuration decomposition pattern and help organize the parameters.

It should be noted that the trees in Figure 6.2 only describe one configuration aggregation pattern. Other patterns can be added to the knowledge base, as the implementation continues and the system scope is broadened.

Configuration and function terms are saved in the "configuration dictionary" and "function dictionary", which may be referred to in later development. Figure 6.6 illustrates the complete configuration dictionary.

○ Develop three basic relations: "f–to–c", "c–subc" and "subf–f". Sample pieces are shown in Figure 6.3 (left portion), Table 6.4, Table 6.5 and Table 6.6.

Table 6.3 : Synthesis shell general units

c–suc.rules
configuration.dictionary
configuration.tree
current–synthesis–cycle
error.message
f–c.pairs
function.dictionary
function.required
function.tree
keepicture.instances
specification
stack –––– configuration.stack
          –––– possible.configuration.stack
sub–f.rules
syn–cycle–display
syn.running.memory

Attention should be paid to the constraints used in the connection design. Constraints usually come with the "c–subc" rules in the synthesis process. Table 6.6 illustrates an example, which brings a constraint: "cons.1", stating "start.l12 = start.l11" between lower dwell part and rise part.

After the above steps, the synthesis knowledge base should be complete. Figure 6.4 shows its various units.

o Load the shell knowledge base, which contains some general control structures to conduct the running of specific knowledge bases. There are two modes to be selected to debug the synthesis knowledge base (Figure 6.5). One is simply going through all the design process without any intervention, the other proceeds with stops at every basic synthesis cycle.

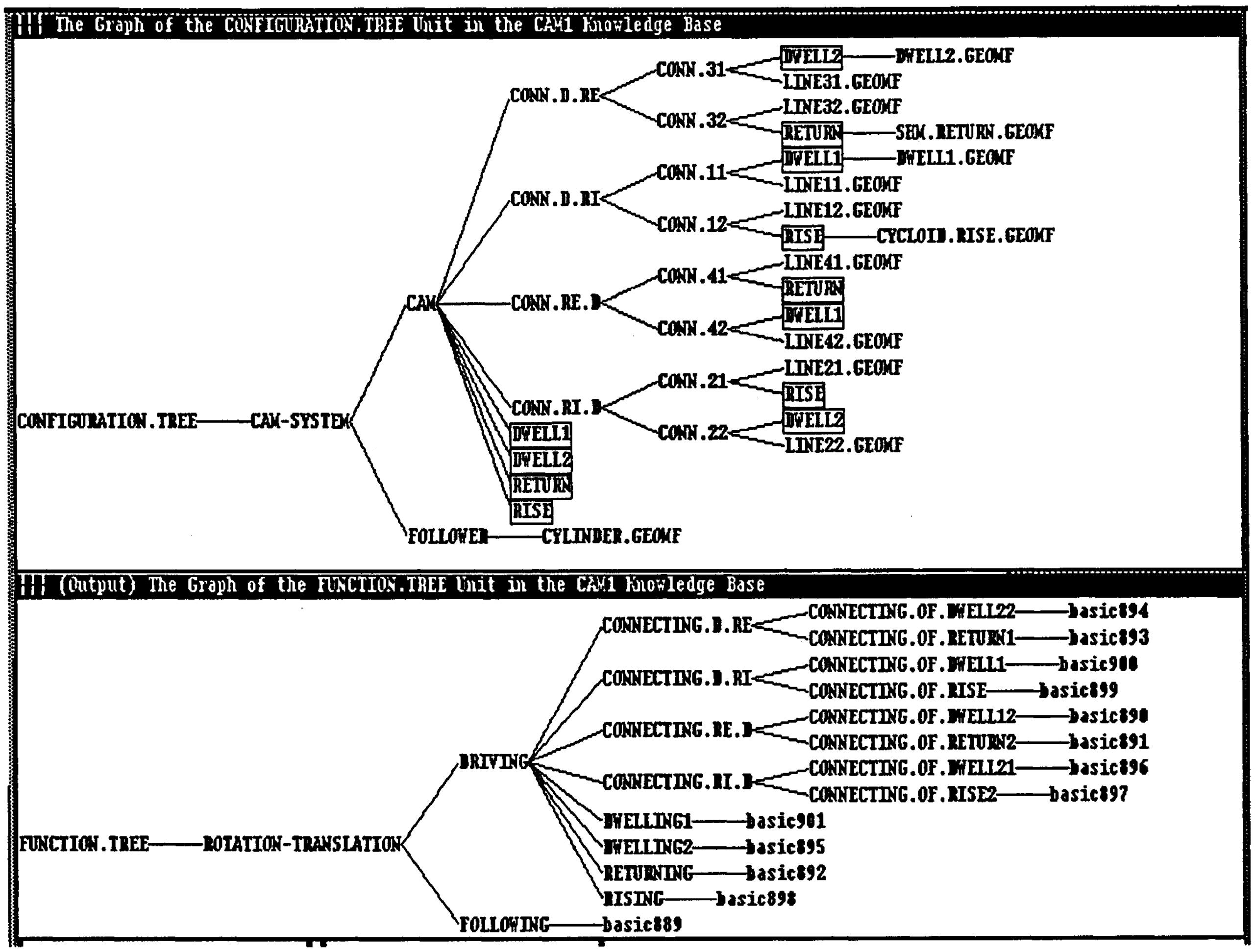
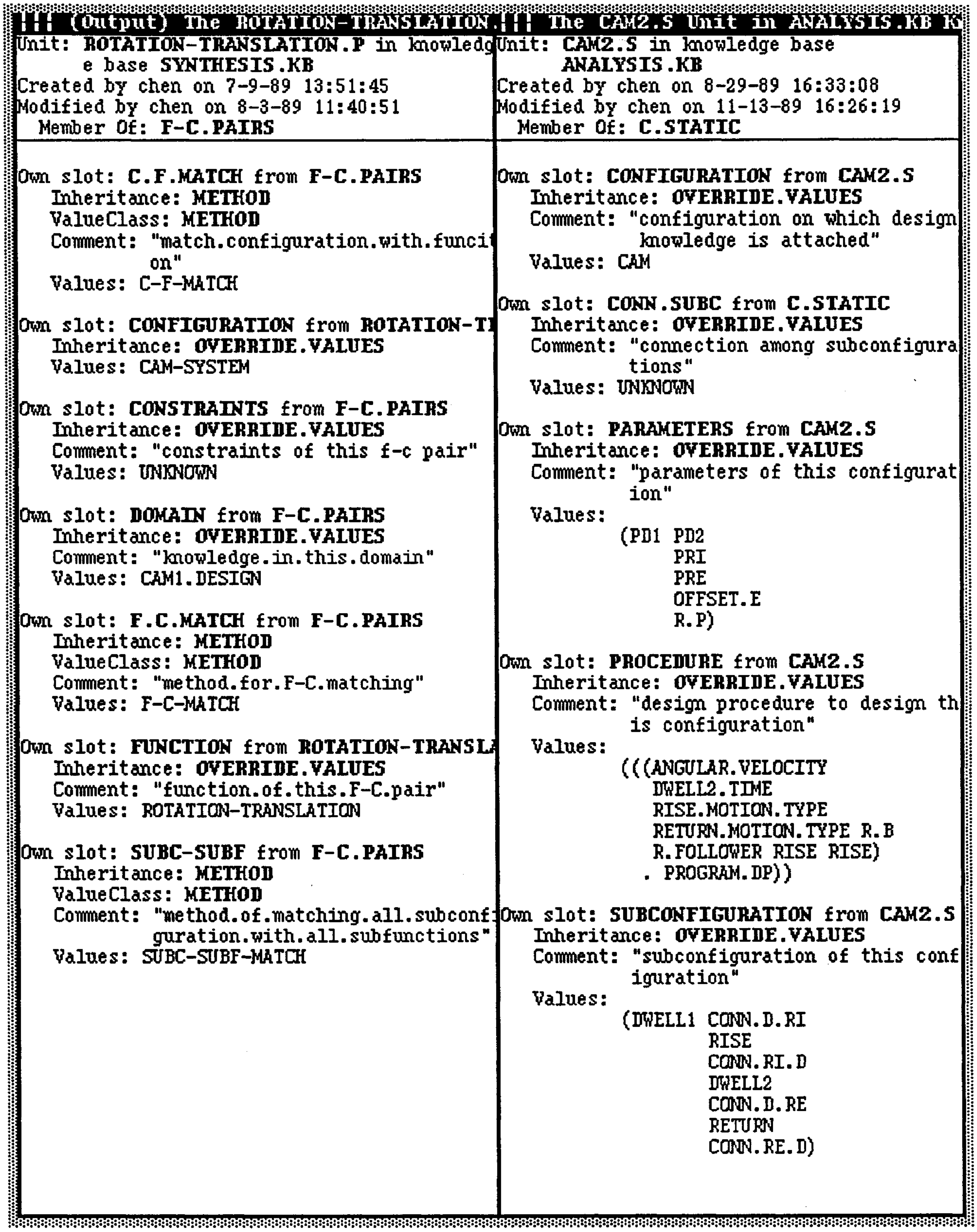


Figure 6.2 : The configuration tree and the function tree

It is suggested that the "step by cycle" mode be selected at this moment, so that any failure can be easily located.

The output of synthesis consists of two trees: a configuration tree and a function tree. They should be identical to the ones created "manually" at the very beginning of the implementation (Figure 6.2).

It should be explained that the above development process solely depends on the KEE environment. No effort is made to improve the user interface.



```
||| (Output) The ROTATION-TRANSLATION.
Unit: ROTATION-TRANSLATION.P in knowledg
      e base SYNTHESIS.KB
Created by chen on 7-9-89 13:51:45
Modified by chen on 8-3-89 11:40:51
  Member Of: F-C.PAIRS

Own slot: C.F.MATCH from F-C.PAIRS
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "match.configuration.with.funci
            on"
   Values: C-F-MATCH

Own slot: CONFIGURATION from ROTATION-T
   Inheritance: OVERRIDE.VALUES
   Values: CAM-SYSTEM

Own slot: CONSTRAINTS from F-C.PAIRS
   Inheritance: OVERRIDE.VALUES
   Comment: "constraints of this f-c pair"
   Values: UNKNOWN

Own slot: DOMAIN from F-C.PAIRS
   Inheritance: OVERRIDE.VALUES
   Comment: "knowledge.in.this.domain"
   Values: CAM1.DESIGN

Own slot: F.C.MATCH from F-C.PAIRS
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "method.for.F-C.matching"
   Values: F-C-MATCH

Own slot: FUNCTION from ROTATION-TRANSL
   Inheritance: OVERRIDE.VALUES
   Comment: "function.of.this.F-C.pair"
   Values: ROTATION-TRANSLATION

Own slot: SUBC-SUBF from F-C.PAIRS
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "method.of.matching.all.subconf
            guration.with.all.subfunctions"
   Values: SUBC-SUBF-MATCH
```

```
||| The CAM2.S Unit in ANALYSIS.KB K
Unit: CAM2.S in knowledge base
      ANALYSIS.KB
Created by chen on 8-29-89 16:33:08
Modified by chen on 11-13-89 16:26:19
  Member Of: C.STATIC

Own slot: CONFIGURATION from CAM2.S
   Inheritance: OVERRIDE.VALUES
   Comment: "configuration on which design
            knowledge is attached"
   Values: CAM

Own slot: CONN.SUBC from C.STATIC
   Inheritance: OVERRIDE.VALUES
   Comment: "connection among subconfigura
            tions"
   Values: UNKNOWN

Own slot: PARAMETERS from CAM2.S
   Inheritance: OVERRIDE.VALUES
   Comment: "parameters of this configurat
            ion"
   Values:
          (PD1 PD2
               PRI
               PRE
               OFFSET.E
               R.P)

Own slot: PROCEDURE from CAM2.S
   Inheritance: OVERRIDE.VALUES
   Comment: "design procedure to design th
            is configuration"
   Values:
          (((ANGULAR.VELOCITY
             DWELL2.TIME
             RISE.MOTION.TYPE
             RETURN.MOTION.TYPE R.B
             R.FOLLOWER RISE RISE)
           . PROGRAM.DP))

Own slot: SUBCONFIGURATION from CAM2.S
   Inheritance: OVERRIDE.VALUES
   Comment: "subconfiguration of this conf
            iguration"
   Values:
          (DWELL1 CONN.D.RI
                  RISE
                  CONN.RI.D
                  DWELL2
                  CONN.D.RE
                  RETURN
                  CONN.RE.D)
```

Figure 6.3 : A “f–to–c” relation (left) and a knowledgeable configuration unit (right)

Table 6.4 : A "c–subc" rule

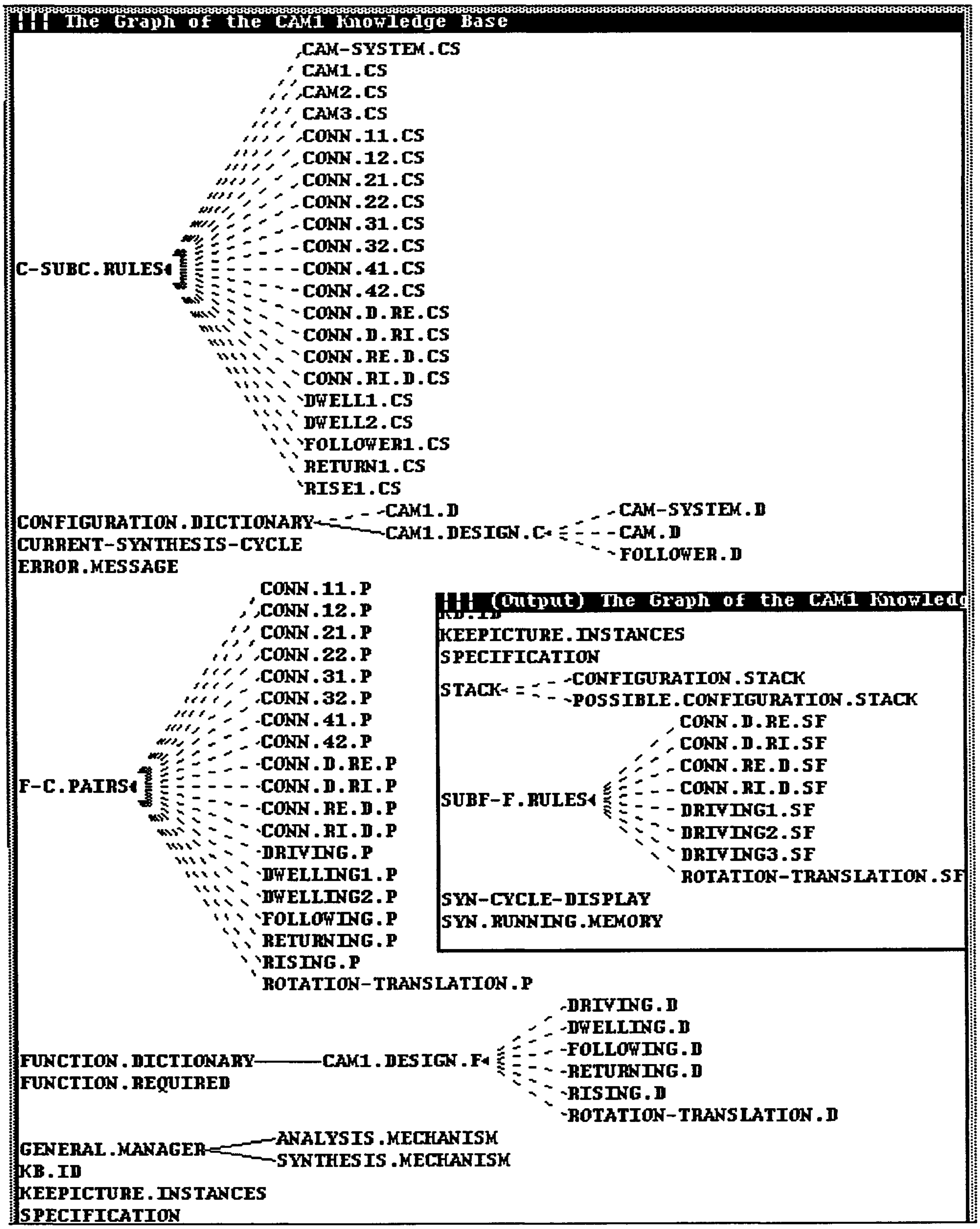
```
(rise1.cs
      (if  (the rise.req of all specification is (no steps in acceleration steps in jerk))
         (the configuration of current–synthesis–cycle is rise)
      then
      (the subconfiguration of current–synthesis–cycle is (cycloid.rise.geomf))
      (the constraints of current–synthesis–cycle is ((rise.motion.type . cycloidal))))).
```

Table 6.5 : A "f–subf" rule

```
(driving3.sf
      (if  (the subfunction of current–synthesis–cycle is (dwelling1 connecting.d.ri rising
         connecting.ri.d dwelling2 connecting.d.re returning connecting.re.d))
         (the motion.pattern of all specification is DRD)
      then
      (the function of current–synthesis–cycle is driving)))
```

Table 6.6 : A "c–subc" rule with constraints

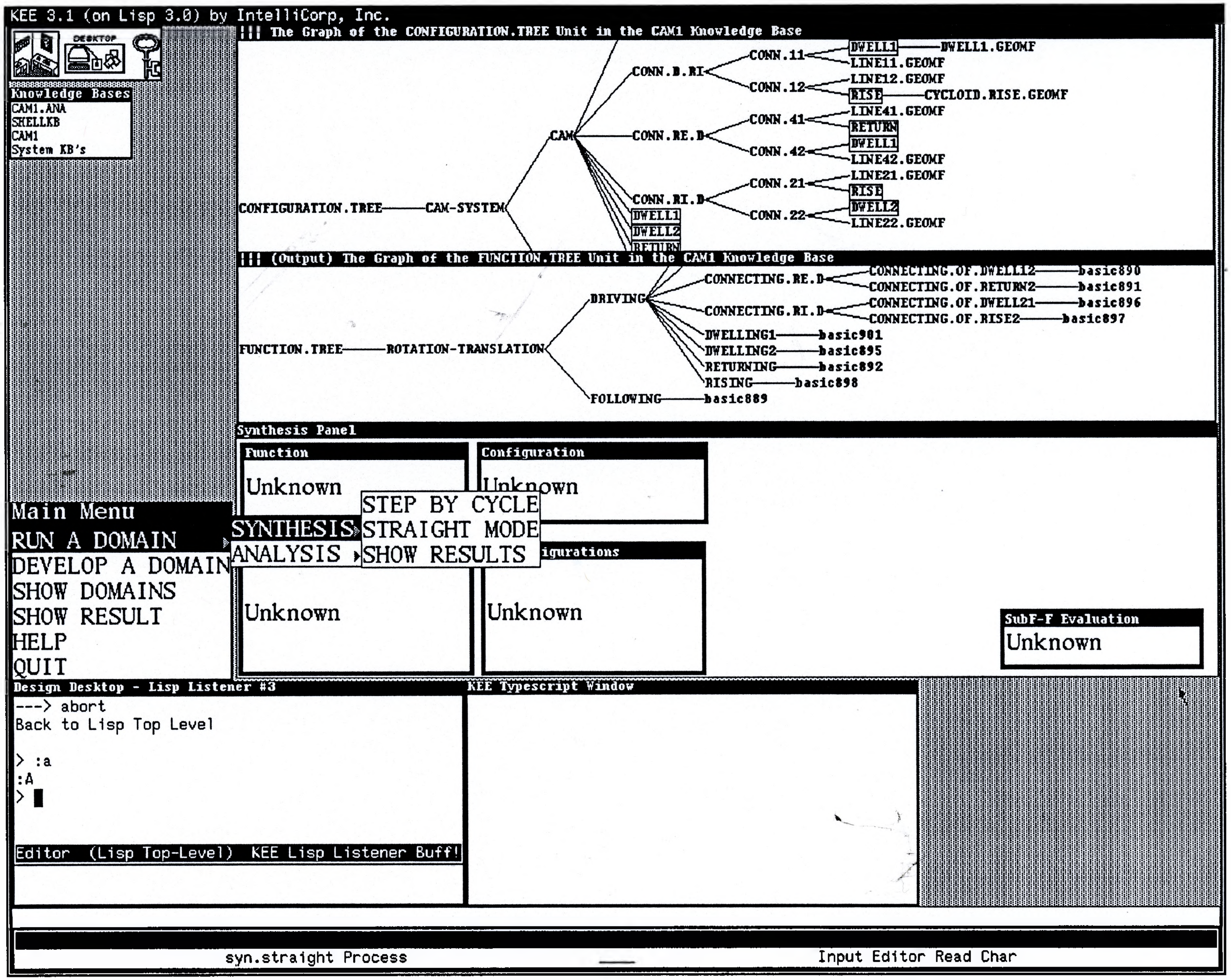
```
(conn.d.ri.cs
      (if  (the configuration of current–synthesis–cycle is conn.d.ri)
      then
      (the subconfiguration of current–synthesis–cycle is '((conn.11 partial dwell1)
            (conn.12 partial rise)))
      (the constraints of current–synthesis–cycle is
            '((cons.1 (start.l12) (start.l11) start.l12 start.l11 equal)))))
```



```
The Graph of the CAM1 Knowledge Base

                    CAM-SYSTEM.CS
                    CAM1.CS
                    CAM2.CS
                    CAM3.CS
                    CONN.11.CS
                    CONN.12.CS
                    CONN.21.CS
                    CONN.22.CS
                    CONN.31.CS
                    CONN.32.CS
C-SUBC.RULES        CONN.41.CS
                    CONN.42.CS
                    CONN.D.RE.CS
                    CONN.D.RI.CS
                    CONN.RE.D.CS
                    CONN.RI.D.CS
                    DWELL1.CS
                    DWELL2.CS
                    FOLLOWER1.CS
                    RETURN1.CS
                    RISE1.CS
                          CAM1.D                         CAM-SYSTEM.D
CONFIGURATION.DICTIONARY  CAM1.DESIGN.C                  CAM.D
CURRENT-SYNTHESIS-CYCLE                                  FOLLOWER.D
ERROR.MESSAGE
                 CONN.11.P
                 CONN.12.P
                 CONN.21.P
                 CONN.22.P
                 CONN.31.P
                 CONN.32.P
                 CONN.41.P
                 CONN.42.P
                 CONN.D.RE.P
F-C.PAIRS        CONN.D.RI.P
                 CONN.RE.D.P
                 CONN.RI.D.P
                 DRIVING.P
                 DWELLING1.P
                 DWELLING2.P
                 FOLLOWING.P
                 RETURNING.P
                 RISING.P
                 ROTATION-TRANSLATION.P
                                           DRIVING.D
                                           DWELLING.D
FUNCTION.DICTIONARY    CAM1.DESIGN.F       FOLLOWING.D
FUNCTION.REQUIRED                          RETURNING.D
                                           RISING.D
                                           ROTATION-TRANSLATION.D
                    ANALYSIS.MECHANISM
GENERAL.MANAGER     SYNTHESIS.MECHANISM
KB.ID
KEEPICTURE.INSTANCES
SPECIFICATION
```

```
(Output) The Graph of the CAM1 Knowled
KB.ID
KEEPICTURE.INSTANCES
SPECIFICATION
         CONFIGURATION.STACK
STACK    POSSIBLE.CONFIGURATION.STACK
                 CONN.D.RE.SF
                 CONN.D.RI.SF
                 CONN.RE.D.SF
SUBF-F.RULES     CONN.RI.D.SF
                 DRIVING1.SF
                 DRIVING2.SF
                 DRIVING3.SF
                 ROTATION-TRANSLATION.SF
SYN-CYCLE-DISPLAY
SYN.RUNNING.MEMORY
```

Figure 6.4 : Portion of cam synthesis knowledge base



```
---> abort
Back to Lisp Top Level

> :a
:A
>
```

Figure 6.5 : Synthesis running windows

<u>Analysis Knowledge Base Development</u>

o Make a copy of the "analysis shell" and rename it as "cam analysis knowledge base", which consists of general structures shown in Table 6.7. These structures will be illustrated by later examples.

Table 6.7 : Analysis shell general units

c.static
checking.slice
configuration.tree
design.procedures
display.av
keepicture.instances
monitor.para.av
parameter.dictionary
parameter.nomenclature
r.m
solving.slice

o Define parameters and organize them according to configurations. This is an important step in the knowledge base development. Once the parameters are defined in "parameter nomenclature", confusion is unlikely for later implementation (Table 6.8).

Table 6.8 : Parameter nomenclature

Parameter In "Designer's Version"	Parameter in K.B.	Brief Explanation
top–dwell	pd1	program of dwell at the top of rise
ϕ	press.angle	pressure angle

A configuration is attached to a number of parameters conveniently and naturally. This attachment is more or less subjective, depending on the knowledge engineer's understanding and organizational style. This is also the content of the knowledgeable configuration units.

o Knowledgeable configuration units (called "c.static"s in the knowledge base)

There is a one–to–one relation between a configuration and its "c.static". One configuration must have at least one "c.static". Figure 6.3 (right portion) shows a cam "c.static" example with six parameters, the decomposition pattern and a design procedure, "program.dp" attached to it.

o Design procedures in knowledgeable configuration units

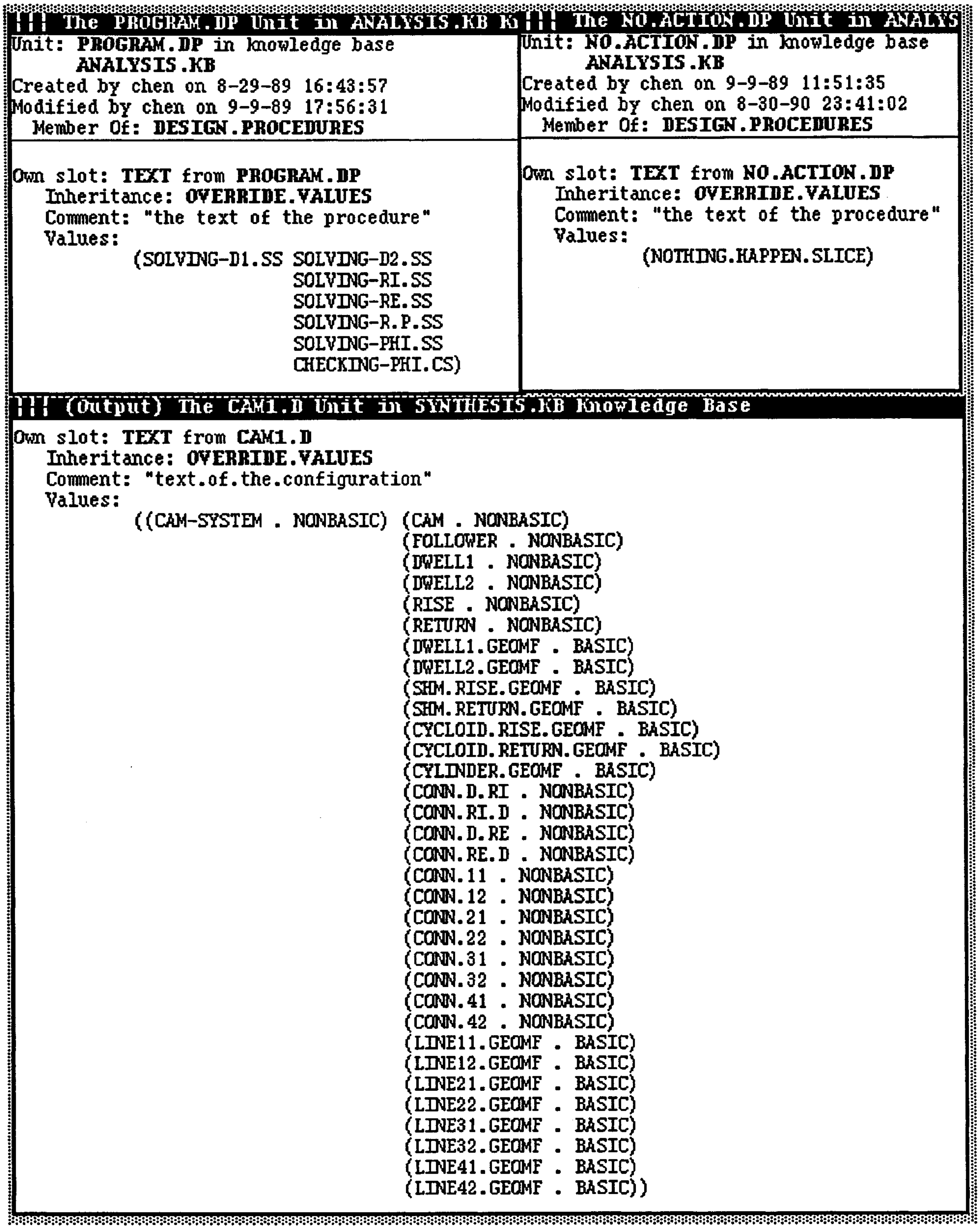
These procedures are created by a program in the form of *((conditions . procedure) | more)*, once the content of a procedure is available. Every "c.static" has at least one design procedure for its parameters. Otherwise, a default design procedure "no.action" should be attached to it, which omits this configuration. Figure 6.6 shows two examples.

As the implementation has shown so far, the solving and checking slices should be created by the time the design procedure is being developed. Furthermore, both should be available when a "c.static" is generated. However, this procedure is not exactly sequential. All of them may need to be considered at the same time. Therefore, the actual development procedure is an iterative process, starting from slices to procedures to "c.static"s.

o Solving and checking slices

Corresponding to each parameter, there will be at least one solving slice. Solving slices are written by using the basic description elements and following the "scheme version". No special organizational effort is needed. Figure 6.7 shows a solving slice.

Whenever there is a checking relation in the design process, there is a checking slice. In Figure 6.8, the "failure.procedure" is activated, when the "checking.criteria" fails. The "fire.status" indicates whether or not the dependent parameters should be modified, since sometimes, the designer may wish them not to be modified. This item simply gives one more option. The failure iteration process can be best illustrated by an example shown in Table 6.9.

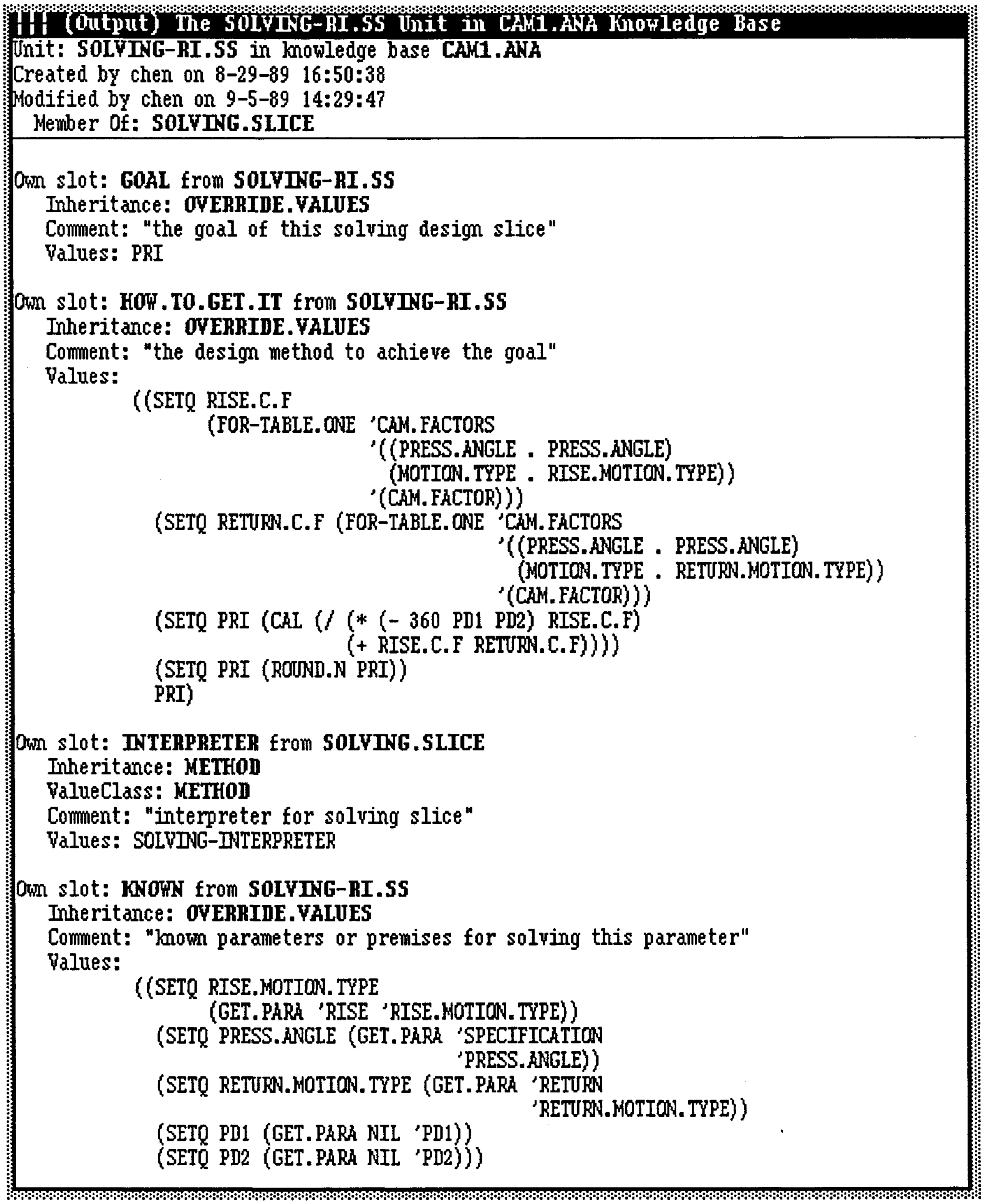


```
||| The PROGRAM.DP Unit in ANALYSIS.KB K
Unit: PROGRAM.DP in knowledge base
      ANALYSIS.KB
Created by chen on 8-29-89 16:43:57
Modified by chen on 9-9-89 17:56:31
  Member Of: DESIGN.PROCEDURES

Own slot: TEXT from PROGRAM.DP
   Inheritance: OVERRIDE.VALUES
   Comment: "the text of the procedure"
   Values:
           (SOLVING-D1.SS SOLVING-D2.SS
                          SOLVING-RI.SS
                          SOLVING-RE.SS
                          SOLVING-R.P.SS
                          SOLVING-PHI.SS
                          CHECKING-PHI.CS)
```

```
||| The NO.ACTION.DP Unit in ANALYS
Unit: NO.ACTION.DP in knowledge base
      ANALYSIS.KB
Created by chen on 9-9-89 11:51:35
Modified by chen on 8-30-90 23:41:02
  Member Of: DESIGN.PROCEDURES

Own slot: TEXT from NO.ACTION.DP
   Inheritance: OVERRIDE.VALUES
   Comment: "the text of the procedure"
   Values:
           (NOTHING.HAPPEN.SLICE)
```

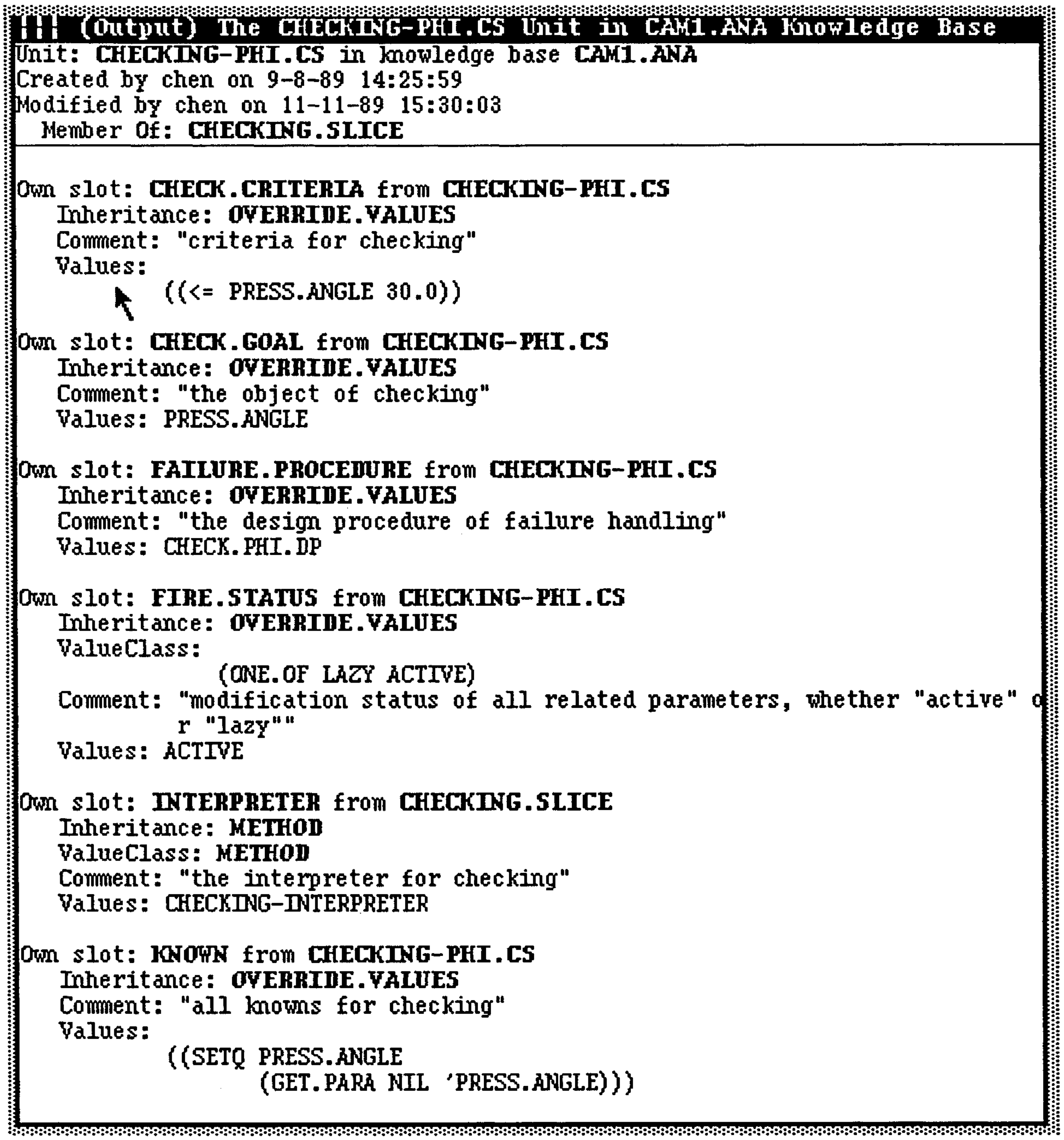
```
||| (Output) The CAM1.D Unit in SYNTHESIS.KB Knowledge Base
Own slot: TEXT from CAM1.D
   Inheritance: OVERRIDE.VALUES
   Comment: "text.of.the.configuration"
   Values:
           ((CAM-SYSTEM . NONBASIC) (CAM . NONBASIC)
                                    (FOLLOWER . NONBASIC)
                                    (DWELL1 . NONBASIC)
                                    (DWELL2 . NONBASIC)
                                    (RISE . NONBASIC)
                                    (RETURN . NONBASIC)
                                    (DWELL1.GEOMF . BASIC)
                                    (DWELL2.GEOMF . BASIC)
                                    (SHM.RISE.GEOMF . BASIC)
                                    (SHM.RETURN.GEOMF . BASIC)
                                    (CYCLOID.RISE.GEOMF . BASIC)
                                    (CYCLOID.RETURN.GEOMF . BASIC)
                                    (CYLINDER.GEOMF . BASIC)
                                    (CONN.D.RI . NONBASIC)
                                    (CONN.RI.D . NONBASIC)
                                    (CONN.D.RE . NONBASIC)
                                    (CONN.RE.D . NONBASIC)
                                    (CONN.11 . NONBASIC)
                                    (CONN.12 . NONBASIC)
                                    (CONN.21 . NONBASIC)
                                    (CONN.22 . NONBASIC)
                                    (CONN.31 . NONBASIC)
                                    (CONN.32 . NONBASIC)
                                    (CONN.41 . NONBASIC)
                                    (CONN.42 . NONBASIC)
                                    (LINE11.GEOMF . BASIC)
                                    (LINE12.GEOMF . BASIC)
                                    (LINE21.GEOMF . BASIC)
                                    (LINE22.GEOMF . BASIC)
                                    (LINE31.GEOMF . BASIC)
                                    (LINE32.GEOMF . BASIC)
                                    (LINE41.GEOMF . BASIC)
                                    (LINE42.GEOMF . BASIC))
```

Figure 6.6 : A normal design procedure (upper left), a “no.action” procedure (upper right) and the configuration dictionary (lower)



```
||| (Output) The SOLVING-RI.SS Unit in CAM1.ANA Knowledge Base
Unit: SOLVING-RI.SS in knowledge base CAM1.ANA
Created by chen on 8-29-89 16:50:38
Modified by chen on 9-5-89 14:29:47
  Member Of: SOLVING.SLICE

Own slot: GOAL from SOLVING-RI.SS
   Inheritance: OVERRIDE.VALUES
   Comment: "the goal of this solving design slice"
   Values: PRI

Own slot: HOW.TO.GET.IT from SOLVING-RI.SS
   Inheritance: OVERRIDE.VALUES
   Comment: "the design method to achieve the goal"
   Values:
           ((SETQ RISE.C.F
                  (FOR-TABLE.ONE 'CAM.FACTORS
                                 '((PRESS.ANGLE . PRESS.ANGLE)
                                   (MOTION.TYPE . RISE.MOTION.TYPE))
                                 '(CAM.FACTOR)))
             (SETQ RETURN.C.F (FOR-TABLE.ONE 'CAM.FACTORS
                                             '((PRESS.ANGLE . PRESS.ANGLE)
                                               (MOTION.TYPE . RETURN.MOTION.TYPE))
                                             '(CAM.FACTOR)))
             (SETQ PRI (CAL (/ (* (- 360 PD1 PD2) RISE.C.F)
                               (+ RISE.C.F RETURN.C.F))))
             (SETQ PRI (ROUND.N PRI))
             PRI)

Own slot: INTERPRETER from SOLVING.SLICE
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "interpreter for solving slice"
   Values: SOLVING-INTERPRETER

Own slot: KNOWN from SOLVING-RI.SS
   Inheritance: OVERRIDE.VALUES
   Comment: "known parameters or premises for solving this parameter"
   Values:
           ((SETQ RISE.MOTION.TYPE
                  (GET.PARA 'RISE 'RISE.MOTION.TYPE))
             (SETQ PRESS.ANGLE (GET.PARA 'SPECIFICATION
                                         'PRESS.ANGLE))
             (SETQ RETURN.MOTION.TYPE (GET.PARA 'RETURN
                                                'RETURN.MOTION.TYPE))
             (SETQ PD1 (GET.PARA NIL 'PD1))
             (SETQ PD2 (GET.PARA NIL 'PD2)))
```

Figure 6.7 : A solving slice



```
(Output) The CHECKING-PHI.CS Unit in CAM1.ANA Knowledge Base
Unit: CHECKING-PHI.CS in knowledge base CAM1.ANA
Created by chen on 9-8-89 14:25:59
Modified by chen on 11-11-89 15:30:03
  Member Of: CHECKING.SLICE

Own slot: CHECK.CRITERIA from CHECKING-PHI.CS
   Inheritance: OVERRIDE.VALUES
   Comment: "criteria for checking"
   Values:
            ((<= PRESS.ANGLE 30.0))

Own slot: CHECK.GOAL from CHECKING-PHI.CS
   Inheritance: OVERRIDE.VALUES
   Comment: "the object of checking"
   Values: PRESS.ANGLE

Own slot: FAILURE.PROCEDURE from CHECKING-PHI.CS
   Inheritance: OVERRIDE.VALUES
   Comment: "the design procedure of failure handling"
   Values: CHECK.PHI.DP

Own slot: FIRE.STATUS from CHECKING-PHI.CS
   Inheritance: OVERRIDE.VALUES
   ValueClass:
               (ONE.OF LAZY ACTIVE)
   Comment: "modification status of all related parameters, whether "active" o
            r "lazy""
   Values: ACTIVE

Own slot: INTERPRETER from CHECKING.SLICE
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "the interpreter for checking"
   Values: CHECKING-INTERPRETER

Own slot: KNOWN from CHECKING-PHI.CS
   Inheritance: OVERRIDE.VALUES
   Comment: "all knowns for checking"
   Values:
           ((SETQ PRESS.ANGLE
                  (GET.PARA NIL 'PRESS.ANGLE)))
```

Figure 6.8 : A checking slice

Table 6.9 : *ILLUSTRATION OF RUNNING PROCESS*

The following frames are a part of a cam design process with step-by-step information (one frame is a design step), which mainly illustrates the iterative process due to the pressure angle checking relation: *(pressure angle < given limit)*. There are three columns in the following piece: "kb.items" and "parameter.status" are obtained from the analysis running windows shown in Figure 6.11 which include *current.configuration, current.procedure, current.design.slice, current.design.goal, checking.slice, current.parameters*; and "explanation" gives comments on the running process.

KB.ITEMS	PARA.STATUS	EXPLANATION
current.configuration: *cam*	*(pd1 nil)* *(pd2 nil)* *(pri nil)* *(pre nil)* *(offset.e nil)* *(r.p nil)*	; a configuration is selected ; its parameters set is shown automatically, which includes programs for cam portions: dwell1, dwell2, rise, return and offset, pitch circle radius

KB.ITEMS	PARA.STATUS	EXPLANATION
current.procedure: *program.dp* *.solving-d1.ss* *.solving-d2.ss* *.solving-ri.ss* *.solving-re.ss* *.solving-r.p.ss* *.solving-phi.ss* *.checking-phi.cs*		; a design procedure is selected

KB.ITEMS	PARA.STATUS	EXPLANATION
current.design.slice: *solving-d1.ss* *.goal: PD1* *.how-to:* *1. rule reasoning:* *if: a small dwell is required;* *then: assign it 10°*	*(pd1 10)* *(pd2 nil)* *(pri nil)* *(pre nil)* *(offset.e nil)* *(r.p nil)*	; the slice searches rules. this rule is selected due to the "small dwell" in specification ; notice the parameter status change

KB.ITEMS	PARA.STATUS	EXPLANATION
current.design.slice:	*(pd1 10)*	; the slice first calculates pd2,
solving-d2.ss	*(pd2 108)*	then rounds it up.
.goal: PD2	*(pri nil)*	
.how-to:	*(pre nil)*	
*1.pd2=angular.v **	*(offset.e nil)*	
dwell2.time	*(r.p nil)*	
2.round pd2		

KB.ITEMS	PARA.STATUS	EXPLANATION
(*current.design.slice:*	*(pd1 10)*	; slices: solving-ri.ss,
solving-ri.ss	*(pd2 108)*	solving-re.ss,
. . .	*(pri 135)*	solving-r.p.ss
	(pre 107)	are not detailed here
solving-re.ss	*(offset.e nil)*	; the new parameter status
. . .	*(r.p 2.75)*	is shown
solving-r.p.ss		
. . .)		

KB.ITEMS	PARA.STATUS	EXPLANATION
current.design.slice:	*(pd1 10)*	; the second step obtains
solving-phi.ss	*(pd2 108)*	two nearest cam factors
.goal: press.angle	*(pri 135)*	from the cam factor
.how-to:	*(pre 107)*	table according to f.rise
*1. f.rise=r.p*pri / rise*	*(offset.e nil)*	
2.((press.a1 cam.f1)	*(r.p 2.75)*	
(press.a2 cam.f2))=		
interpolate('cam.factors	OTHER.PARA	; press.angle is a cam-system
f.rise 'press.angle		parameter. Previously, it was
'motion.type)	*(press.angle 33.16)*	assumed according to limit
3. press.angle=press.a1+		given in specification (30).
*(press.a2 - press.a1) **		Now, it's changed to 33.16.
(f.rise-cam.f1) / (cam.f2 -		
cam.f1)		

KB.ITEMS	PARA.STATUS	EXPLANATION
current.design.slice:	*(pd1 10)*	; this checking relation fails,
checking-phi.ss	*(pd2 108)*	because 33.16 not < 30.
.goal: press.angle	*(pri 135)*	The failure procedure is
.criteria:	*(pre 107)*	invoked.
press.angle <	*(offset.e nil)*	
press.angle.limit	*(r.p 2.75)*	; the "checking.interpreter"
.failure.procedure:		is called, which executes
check-phi.dp		the failure procedure and
	OTHER.PARA	modifies dependent parameters,
	(press.angle 33.16)	then the interpreter verifies
		the checking relation again.
		This iteration continues, until
		the checking criteria are met.

KB.ITEMS	PARA.STATUS	EXPLANATION
current.procedure:		; a failure design procedure
check-phi.dp		is invoked.
.solving-e.ss		
.solving-e.phi.ss		

KB.ITEMS	PARA.STATUS	EXPLANATION
current.design.slice:	*(pd1 10)*	; set offset.e at 0.5,
solving-e.ss	*(pd2 108)*	if later iteration occurs,
.goal: offset.e	*(pri 135)*	offset increases by 0.1
.how-to:	*(pre 107)*	each time
1. increase offset.e	*(offset.e 0.5)*	
from 0.5 by 0.1	*(r.p 2.75)*	
each time.		
	OTHER.PARA	
	(press.angle 33.16)	

KB.ITEMS	PARA.STATUS	EXPLANATION
current.design.slice:	*(pd1 10)*	; after offset.e, this slice obtains
solving-e.phi.ss	*(pd2 108)*	press.angle 28.09.
.goal: press.angle	*(pri 135)*	; parameters Vfmax, w, r.b, y_p
.how-to:	*(pre 107)*	are already determined in
1. $y = (r.b^2 - e^2)^{1/2} + y_p$	*(offset.e 0.5)*	earlier stage.
2. *press.angle =*	*(r.p 2.75)*	
(Vfmax-ew)/yw		
	OTHER.PARA	
	(press.angle 28.09)	

KB.ITEMS	PARA.STATUS	EXPLANATION
current.design.slice: *checking-phi.ss* *.goal: press.angle* *.criteria:* *press.angle <* *press.angle.limit* *.failure.procedure:* *check-phi.dp*	*(pd1 10)* *(pd2 108)* *(pri 135)* *(pre 107)* *(offset.e 0.5)* *(r.p 2.75)* OTHER.PARA *(press.angle 28.09)*	; the press.angle is rechecked, and criteria are met. Thus, the failure procedure has successfully modified the design. ; all dependent parameters have to be checked. The dependent parameters are: pri, pre. (In this case, pri and pre are obtained through the pressure angle limit (according to the designer's knowledge version) so even these two parameters are recalculated, the same results remain.)

KB.ITEMS	PARA.STATUS	EXPLANATION
(*current.design.slice:* *solving-ri.ss* . . . *solving-re.ss* . . .)	*(pd1 10)* *(pd2 108)* *(pri 135)* *(pre 107)* *(offset.e 0.5)* *(r.p 2.75)* OTHER.PARA *(press.angle 28.09)*	; slices for parameters: pri & pre are reinvoked. But as explained results remain the same.

The design process for the cam using procedure "program.dp" has ended.

Two basic description types: data tables and rules, are given some detailed explanation here. The rule facility in KEE is used for the data tables, which basically represents one row in a rule with all columns as rule items. Two special units need to be created for each table. One is the "table header", which includes all items (table columns) appearing in the table, the other is the rule class, which clusters all rules or rows in the table. One example about the cam factor table is given in Table 6.10.

Table 6.10 : The cam factor table

Pressure.Angle	Uniform	Modified.Uniform	SHM	Parabolic/Cycloidal
10	5.67	5.84	8.91	11.34
15	3.73	3.99	5.85	7.46
. . .	. . .	. . .	. . .	. . .
30	1.73	2.27	2.72	3.46
35	1.43	2.06	2.24	2.86
. . .	. . .	. . .	. . .	. . .

Shown in Figure 6.9 are units for the table header and the rule class.

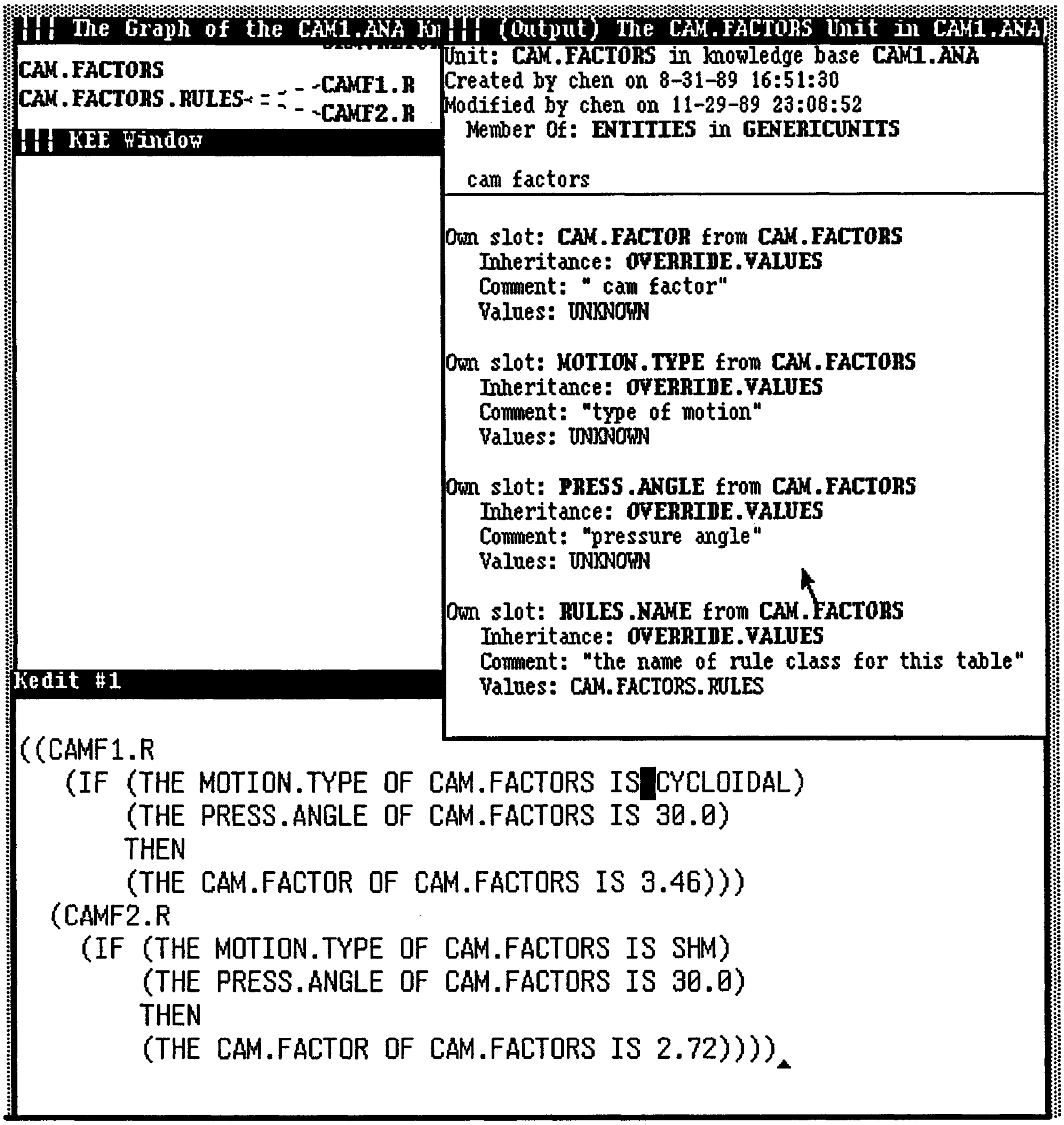
The rules can also be represented in this manner: a rule header and a rule class, with the same basic element format. However, it is also possible to write rules in a free format within rule classes.

After the above steps, the analysis knowledge base has been completed with units shown in Figure 6.10.

o Load the shell knowledge base (if it has not been loaded), run and debug the analysis knowledge base. As is the case in the synthesis, there are several modes (shown in Figure 6.11). "By configuration" is recommended at this stage.

The user interface of the above analysis development solely depends on the KEE facilities, as is the case in the synthesis process.

So far, the knowledge bases (both synthesis and analysis) for the cam design system have been developed.

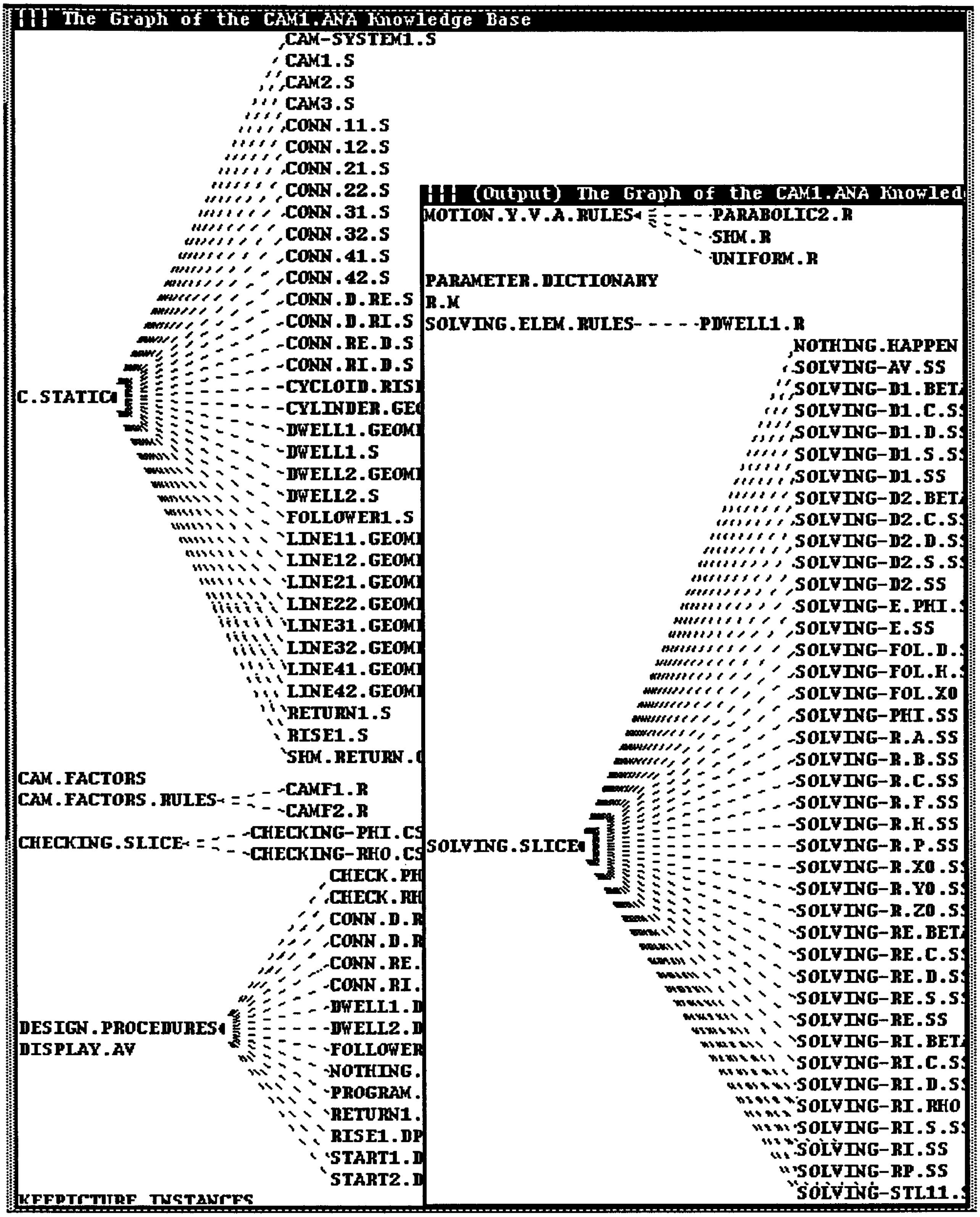

The Graph of the CAM1.ANA Kn
(Output) The CAM.FACTORS Unit in CAM1.ANA
KEE Window
Kedit #1

```
CAM.FACTORS
CAM.FACTORS.RULES - - -CAMF1.R
                  - - -CAMF2.R
```

```
Unit: CAM.FACTORS in knowledge base CAM1.ANA
Created by chen on 8-31-89 16:51:30
Modified by chen on 11-29-89 23:08:52
  Member Of: ENTITIES in GENERICUNITS

  cam factors

Own slot: CAM.FACTOR from CAM.FACTORS
   Inheritance: OVERRIDE.VALUES
   Comment: " cam factor"
   Values: UNKNOWN

Own slot: MOTION.TYPE from CAM.FACTORS
   Inheritance: OVERRIDE.VALUES
   Comment: "type of motion"
   Values: UNKNOWN

Own slot: PRESS.ANGLE from CAM.FACTORS
   Inheritance: OVERRIDE.VALUES
   Comment: "pressure angle"
   Values: UNKNOWN

Own slot: RULES.NAME from CAM.FACTORS
   Inheritance: OVERRIDE.VALUES
   Comment: "the name of rule class for this table"
   Values: CAM.FACTORS.RULES
```

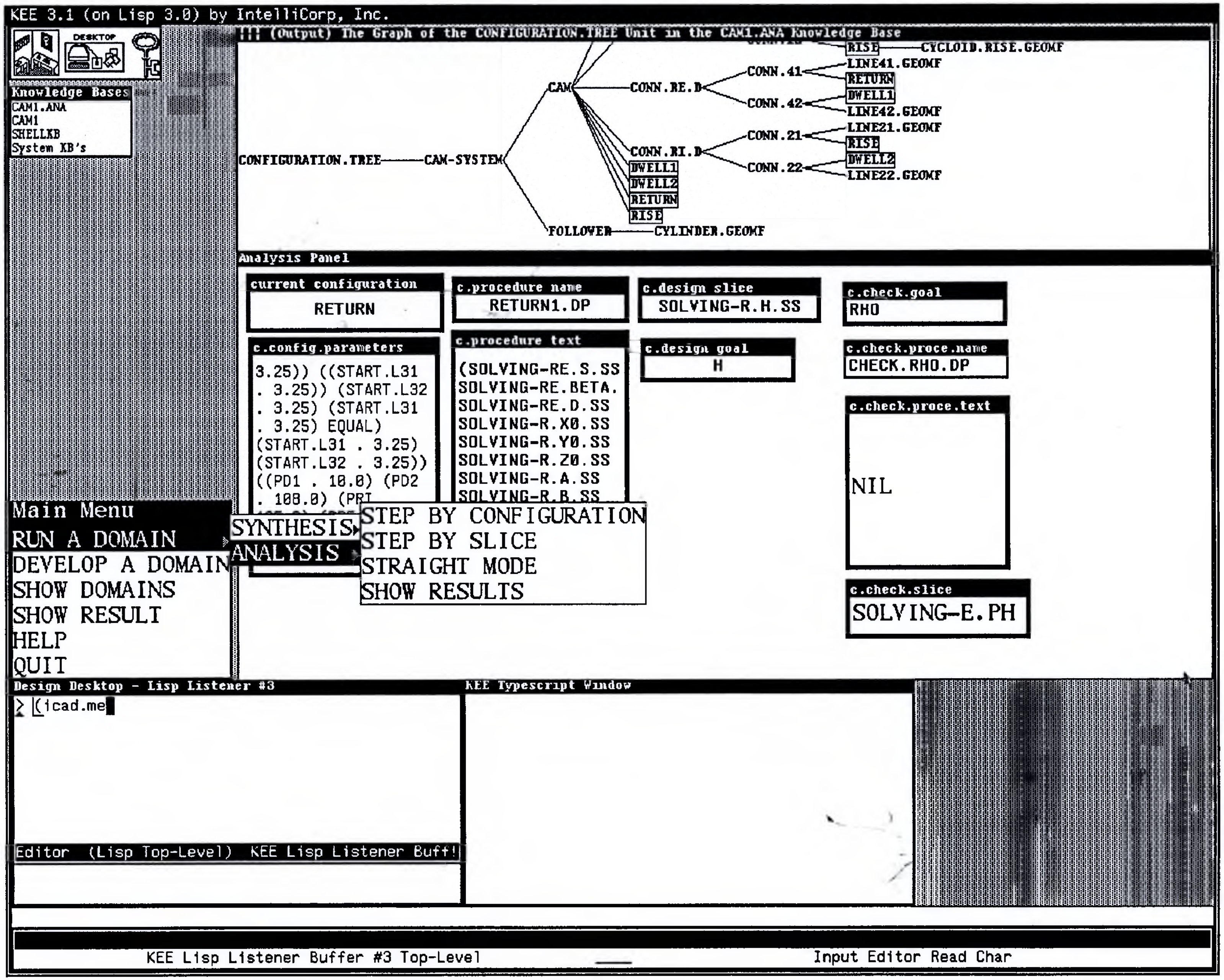
```
((CAMF1.R
   (IF (THE MOTION.TYPE OF CAM.FACTORS IS CYCLOIDAL)
       (THE PRESS.ANGLE OF CAM.FACTORS IS 30.0)
       THEN
       (THE CAM.FACTOR OF CAM.FACTORS IS 3.46)))
  (CAMF2.R
    (IF (THE MOTION.TYPE OF CAM.FACTORS IS SHM)
        (THE PRESS.ANGLE OF CAM.FACTORS IS 30.0)
        THEN
        (THE CAM.FACTOR OF CAM.FACTORS IS 2.72))))
```

Figure 6.9 :Cam factor table, table header structure and rule structure



```
The Graph of the CAM1.ANA Knowledge Base

                CAM-SYSTEM1.S
                CAM1.S
                CAM2.S
                CAM3.S
                CONN.11.S
                CONN.12.S
                CONN.21.S
                CONN.22.S
                CONN.31.S
                CONN.32.S
                CONN.41.S
                CONN.42.S
                CONN.D.RE.S
                CONN.D.RI.S
                CONN.RE.D.S
                CONN.RI.D.S
C.STATIC        CYCLOID.RIS
                CYLINDER.GE
                DWELL1.GEOM
                DWELL1.S
                DWELL2.GEOM
                DWELL2.S
                FOLLOWER1.S
                LINE11.GEOM
                LINE12.GEOM
                LINE21.GEOM
                LINE22.GEOM
                LINE31.GEOM
                LINE32.GEOM
                LINE41.GEOM
                LINE42.GEOM
                RETURN1.S
                RISE1.S
                SHM.RETURN.
CAM.FACTORS
CAM.FACTORS.RULES    CAMF1.R
                     CAMF2.R
CHECKING.SLICE       CHECKING-PHI.CS
                     CHECKING-RHO.CS
                     CHECK.PH
                     CHECK.RH
                     CONN.D.R
                     CONN.D.R
                     CONN.RE.
                     CONN.RI.
                     DWELL1.D
DESIGN.PROCEDURES    DWELL2.D
DISPLAY.AV           FOLLOWER
                     NOTHING.
                     PROGRAM.
                     RETURN1.
                     RISE1.DP
                     START1.D
                     START2.D
KEEPICTURE INSTANCES

(Output) The Graph of the CAM1.ANA Knowled

MOTION.Y.V.A.RULES    PARABOLIC2.R
                      SHM.R
                      UNIFORM.R
PARAMETER.DICTIONARY
R.M
SOLVING.ELEM.RULES    PDWELL1.R
                      NOTHING.HAPPEN
                      SOLVING-AV.SS
                      SOLVING-D1.BET
                      SOLVING-D1.C.S
                      SOLVING-D1.D.S
                      SOLVING-D1.S.S
                      SOLVING-D1.SS
                      SOLVING-D2.BET
                      SOLVING-D2.C.S
                      SOLVING-D2.D.S
                      SOLVING-D2.S.S
                      SOLVING-D2.SS
                      SOLVING-E.PHI.
                      SOLVING-E.SS
                      SOLVING-FOL.D.
                      SOLVING-FOL.H.
                      SOLVING-FOL.X0
                      SOLVING-PHI.SS
                      SOLVING-R.A.SS
                      SOLVING-R.B.SS
                      SOLVING-R.C.SS
                      SOLVING-R.F.SS
                      SOLVING-R.H.SS
SOLVING.SLICE         SOLVING-R.P.SS
                      SOLVING-R.X0.S
                      SOLVING-R.Y0.S
                      SOLVING-R.Z0.S
                      SOLVING-RE.BET
                      SOLVING-RE.C.S
                      SOLVING-RE.D.S
                      SOLVING-RE.S.S
                      SOLVING-RE.SS
                      SOLVING-RI.BET
                      SOLVING-RI.C.S
                      SOLVING-RI.D.S
                      SOLVING-RI.RHO
                      SOLVING-RI.S.S
                      SOLVING-RI.SS
                      SOLVING-RP.SS
                      SOLVING-STL11.
```

Figure 6.10 : Cam analysis knowledge base



current configuration: RETURN

c.procedure name: RETURN1.DP

c.design slice: SOLVING-R.H.SS

c.check.goal: RHO

c.config.parameters:

```
3.25)) ((START.L31
. 3.25)) (START.L32
. 3.25) (START.L31
. 3.25) EQUAL)
(START.L31 . 3.25)
(START.L32 . 3.25))
((PD1 . 10.0) (PD2
. 100.0) (PRI
```

c.procedure text:

```
(SOLVING-RE.S.SS
SOLVING-RE.BETA.
SOLVING-RE.D.SS
SOLVING-R.X0.SS
SOLVING-R.Y0.SS
SOLVING-R.Z0.SS
SOLVING-R.A.SS
SOLVING-R.B.SS
```

c.design goal: H

c.check.proce.name: CHECK.RHO.DP

c.check.proce.text: NIL

c.check.slice: SOLVING-E.PH

```
> (icad.me
```

```
Editor   (Lisp Top-Level)   KEE Lisp Listener Buff!
```

```
KEE Lisp Listener Buffer #3 Top-Level          Input Editor Read Char
```

Figure 6.11 : Cam analysis running windows

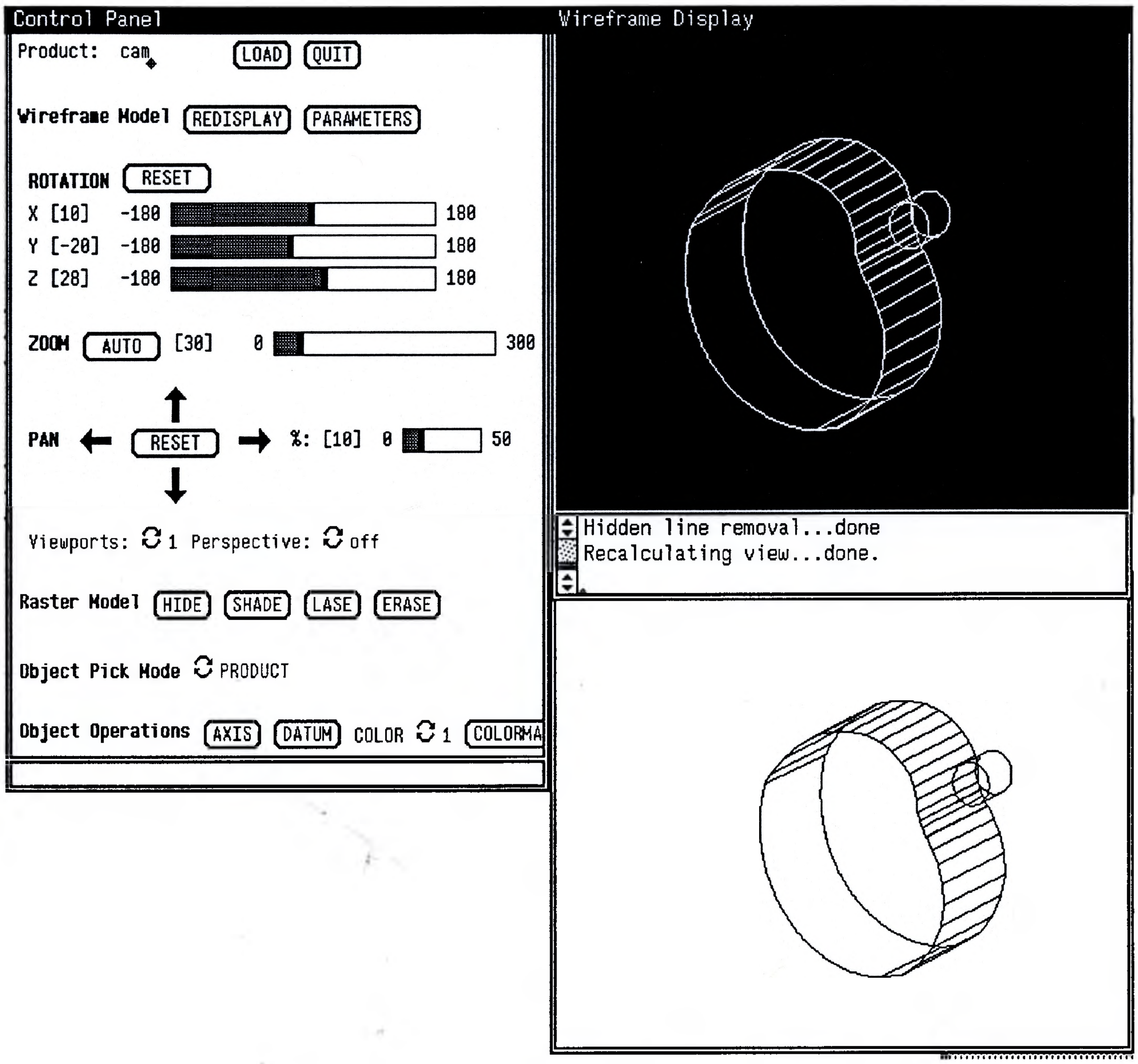
6.1.2 A Cam Design Example

To show the performance of the developed system, one design example is given with the following specifications (an excerpt). Two examples are given in the Appendix with the complete listings.

```
item  : CYCLE.TIME;                 value : 0.05;
item  : R.B;                        value : 1.25;
item  : R.FOLLOWER;                 value : 0.5;
item  : RISE.REQ;                   value : (NO STEPS IN ACCELERATION STEPS IN
                                               JERK);
item  : RISE;                       value : 2.0;
item  : DWELL1.REQ;                 value : (A SMALL DWELL);
item  : DWELL2.TIME;                value : 0.015;
item  : RETURN.SUGG.MOTION;         value : SHM;
item  : PRESS.ANGLE;                value : 30;
item  : MOTION.PATTERN;             value : DRD;
```

A part of parameters and a graphics output (Figure 6.12) are shown below (the complete parameters are given in the Appendix) :

```
Configuration : CAM-SYSTEM
   ANGULAR.VELOCITY :  125.663605
   PRESS.ANGLE :  28.904034
Configuration : CAM
   PD1 :  10.0
   PD2 :  108.0
   PRI :  135.0
   PRE :  107.0
   R.P :  2.75
   OFFSET.E :  0.5
```



Hidden line removal...done
Recalculating view...done.

Figure 6.12 : Cam geometry model produced by the IPDM display interface

6.1.3 Comments

The knowledge base size is hard to quantify, since there are various types of representation, not only rules. However, if each item of these types is roughly considered as one rule, the current system has about 159 rules (48 in synthesis and 111 in analysis). New knowledge can be added to the knowledge base quite simply, due to the explicit descriptive representation scheme. Some knowledge items have been shown in the earlier figures and tables.

The result from the above design example is the same as the one from the human expert. This happens when the coded design process is obtained from a single designer. This may not be the case if multiple experts' knowledge from manufacturing, maintenance, and so on, is provided. The fact that the result is a "copy" of human designers' reflects the nature of the basic idea of this thesis -- the explicit descriptive style of expressing design knowledge.

The implementation of the cam system occurred along with the development of the shell; therefore, it is difficult to say how long it takes to create the cam knowledge bases. In the next example, the bolted flange system, the time can be specified, since at that stage, the shell has been completed and the development of the knowledge bases is a separate process.

Thus far, a domain specific design system, the cam system, has been developed based on the shell.

6.2 BOLTED FLANGE DESIGN SYSTEM

6.2.1 Development

Objective and Scope

The purpose of developing the bolted flange system is to demonstrate, along with the cam system, that the shell system does incorporate the common characteristics of a certain number of design processes, and that it is quite simple to develop a very different design system based on the same shell (Figure 6.13 illustrates a flange example). For this purpose, this design system tries to incorporate the knowledge as simple as possible. The scope is defined in Table 6.11.

The following design knowledge is summarized based on references [14, 56, 92 & 93] and on the consultation with an expert in this area [50].

"<u>Designer's Version</u>"

The "designer's version" is listed below by following an example:

☞ Problem Statement

A pipe connecting device,
Pipe pressure: 6 bar (87 lb/in^2),
Temperature: 60^{o}c,
Pipe Diameter: 5″,
Process Fluid: hot water,
Economic Consideration: minimizing cost,
Working Environment: used for pipe work, service condition is general, not severe.

☞ Select Flange Type

Because it is pipe work, cost is to be minimized, and the service condition is general, not severe, a slip–on flange is chosen.

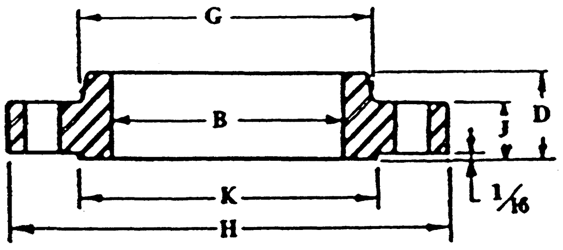
☞ Choose Gasket Material

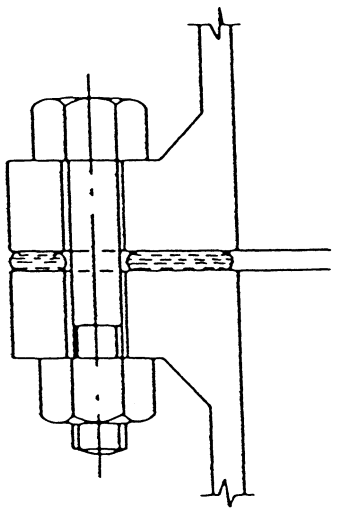
Because the pressure is less than 20 bar and the temperature is lower than 100^{o}c, a gasket made of vegetable fibre serves the purpose, with gasket factor, minimum design seating stress and minimum gasket width as follows: 1.75, 7.6 N/mm^2 and 10 mm.

☞ Select Flange Face Type

Because the flange is inexpensive, simple and used at low pressure ($<$ 10 bar), full–faced type is selected.

☞ Choose a standard flange





PARAMETERS :

B: bore diameter
D: length through hub
G: hub diameter at base
H: flange outside diameter
J : flange thickness
K: outside diameter of raised face
length of raised face
diameter of bolt circle
diameter of bolts
number of bolt holes
length of bolts
nominal pipe size
flange material
flange type
gasket material
nominal pressure
temperature

CONFIGURATIONS :

bolted flange, consists of
- gasket
- two flanges:
  - flange1
  - flange2
- bolt connection, consists of
  - bolt & nut
  - holes on flanges

Figure 6.13 :A flange sketch (figures from [14])

Table 6.11 : Flange System Scope (figures from [14])

bold -- implemented
italic -- not implemented

bolted flange cases :

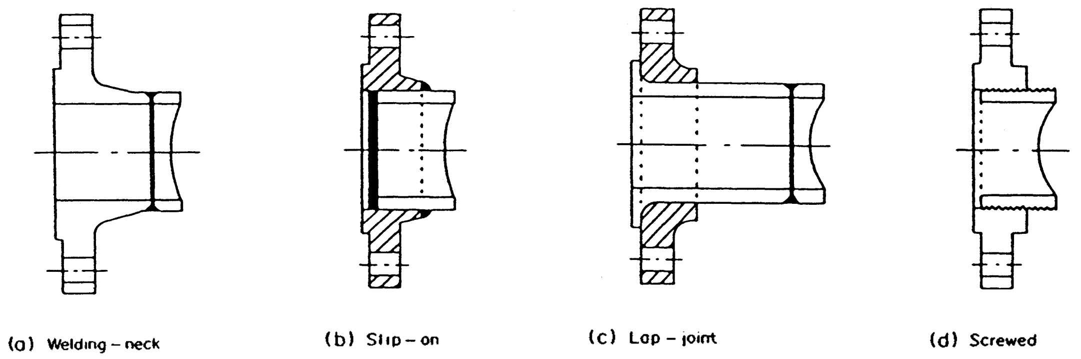
standard flanges

non-standard flanges

flange types :

slip-on

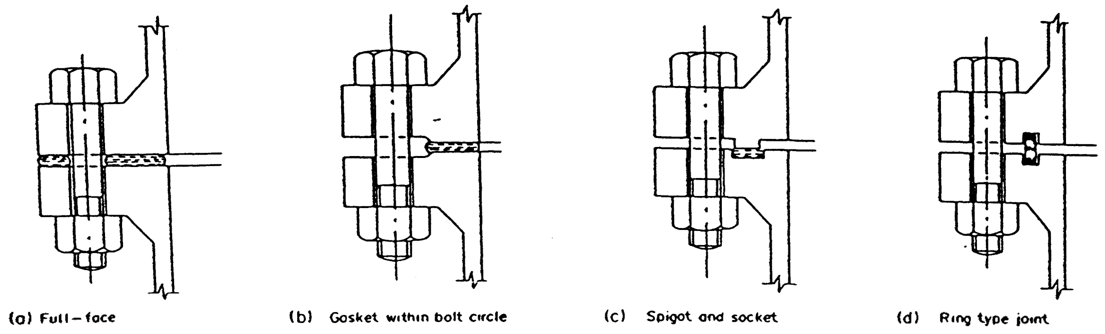
welding-neck, lap-joint and screwed



flange face types :

full-face

gasket within bolt circle, spigot and socket, and ring type joint



pressure & temperature

low pressure, temperature from low to medium

other ranges

data tables & multiple choices :

a portion of them

not complete

For example: only a few cases from the large number of gasket choices; only a few flanges from the standard flange handbook are implemented

☞ Choose a standard flange

According to the pipe diameter (5″), the pressure (87 lb/in^2) < 150, and the slip–on flange type with full–faced gasket, a standard flange is selected from the handbook [56] with the following parameters: B=5.66; D=1 $^7/_{16}$; G=6 $^7/_{16}$; H=10; J=$^{15}/_{16}$; K=7 $^5/_{16}$; No.of holes=8; Dia.of bolts=$^3/_4$; Bolt circle dia.=8 $^1/_2$; Length of bolts=3 $^3/_4$.

☞ Strength Calculation (only an excerpt)

Formulas: (from [14])

longitudinal hub stress: $\sigma_{hb}=F_1M$ (6.1)

radial flange stress: $\sigma_{rd}=F_2M$ (6.2)

tangential flange stress: $\sigma_{tg}=F_3M-F_4\sigma_{rd}$ (6.3)

M is the bigger one of:

$$M_{op}=H_dh_d+H_th_t+H_gh_g \tag{6.4}$$

$$M_{atm}=W_{m2}h_g=y\pi G_dbh_g \tag{6.5}$$

where Hg = gasket reaction,

H_t = pressure force on the flange face,

H_d = pressure force on the area inside the flange,

h_d, h_t, h_g as shown in Figure 6.14,

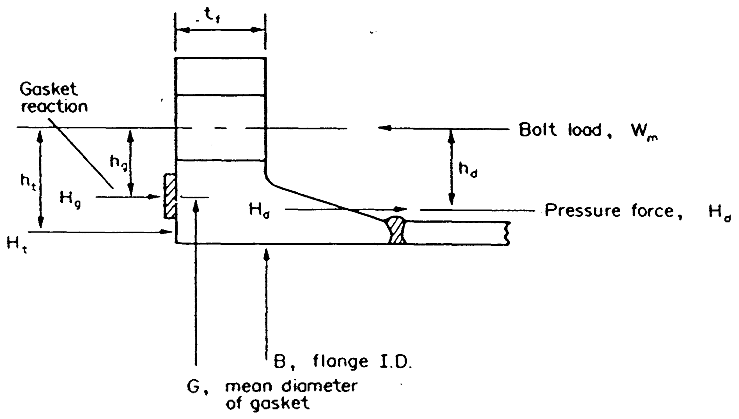


Figure 6.14 : Forces acting on an integral flange (from [14])

y= minimum design seating stress,

2b = effective gasket pressure width,

G_d = mean diameter of the gasket.

F_1, F_2, F_3 and F_4 are factors (refer to [93]) :

$$F_1 = fX/g_1^2 \qquad (6.6)$$
$$F_2 = [1 + 1.33F\frac{J/g_0}{\sqrt{B/g_0}}]X/J^2 \qquad (6.7)$$
$$F_3 = Y/J^2 \qquad (6.8)$$
$$F_4 = Z \qquad (6.9)$$

where f = stress correction factor,

g_0 = shell thickness,

g_1 = maximum hub thickness,

X, Y, Z and F are factors.

All these factors are determined by retrieving all parameter values from tables, figures, multiple choices with criteria, formulas, and the like.

Finally, check the above stresses with maximum allowable design stress f_{fo}:

$$\sigma_{hb} < 1.5\ f_{fo} \qquad (6.10)$$
$$\sigma_{rd} < f_{fo} \qquad (6.11)$$
$$0.5(\sigma_{hb} + \sigma_{rd}) < f_{fo} \qquad (6.12)$$
$$0.5(\sigma_{hb} + \sigma_{tg}) < f_{fo} \qquad (6.13)$$

If strength checking fails, try a larger flange from the handbook.

"Scheme Version"

o Specification Development

The specification list in Table 6.12 is believed to be a basic collection of the design requirements from the user.

o Synthesis

The principal function can be defined as "connecting pipes".

Table 6.12 :Flange specification list

item : Pipe pressure *(the pressure under which the flange operates, e.g. 6 bar),*

item : Temperature *(the temperature at which the flange works, e.g. 60°c),*

item : Pipe Diameter *(the diameter of pipes the flange connects),*

item : Process Fluid *(the substance transmitted in the pipe line),*

item : Economic Consideration *(whether the flange should be designed economically),*

item : Working Environment *(some general description about the environment under which the flange works, e.g. corrosiveness of the process fluid, whether the repeated assembly and disassembly of the flange is required).*

A bolted flange is chosen as the device to connect pipes.

Generally, a bolted flange is comprised of two pieces of the flange, a gasket and a number of bolts and nuts.

Since it is used for pipe work, cost is to be minimized, and the service condition is general, not severe, a slip–on type of flange is selected.

The full–faced type is also determined, due to low pressure and minimum cost.

o Analysis

Since the "steps" and "means" in the "designer's version" are quite clear, no further arrangement is necessary and only the major steps are listed: gasket material selection; choosing a standard flange; strength calculation.

Synthesis Knowledge Base Development

o Make a copy of the "synthesis shell", and rename it as "flange synthesis knowledge base".

o Define the terms for configurations and functions, and create tree structures for configurations and functions (Figure 6.15).

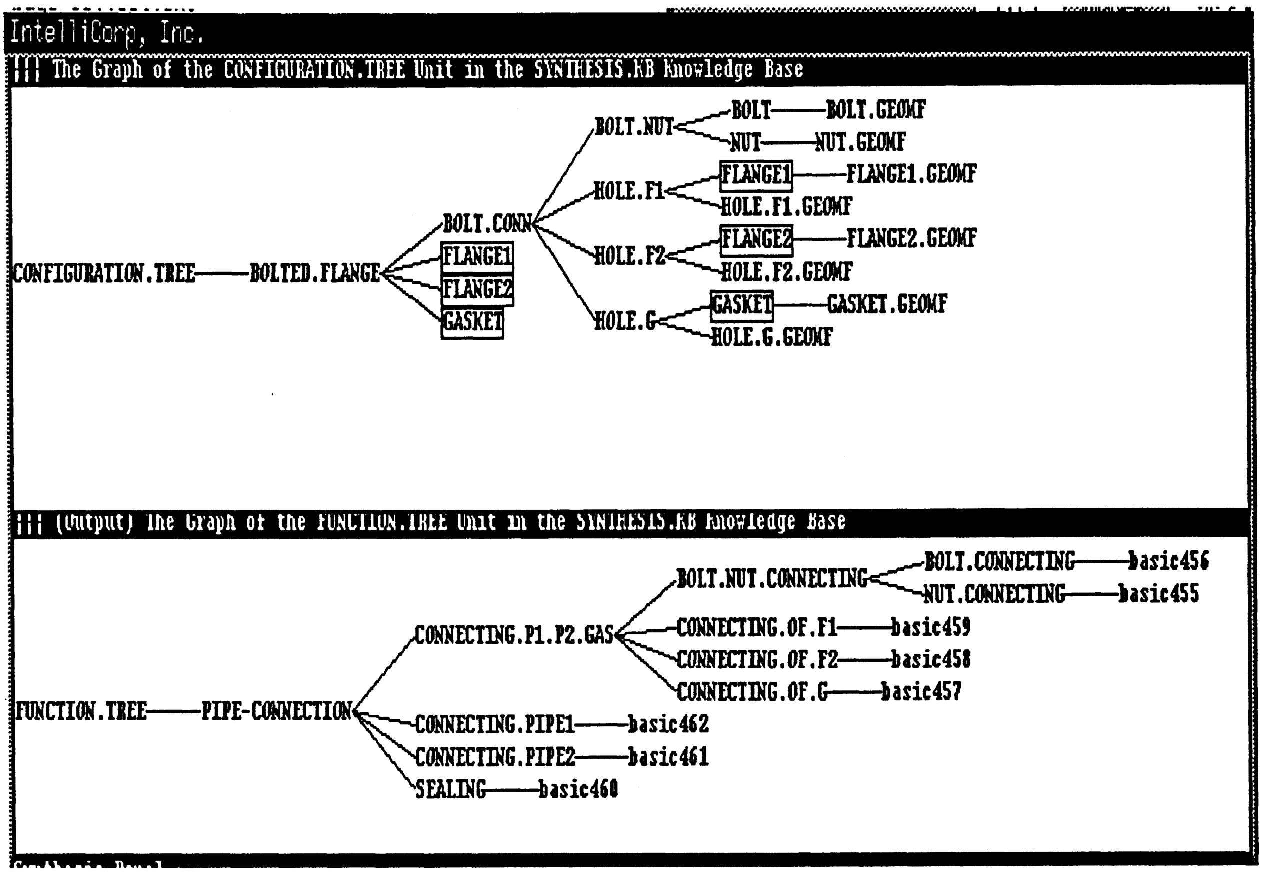


Figure 6.15 : Configuration tree and function tree

o Develop “f–to–c”, “c–subc” and “subf–f” relations. Figure 6.16 shows three examples. The “c–subc” rule is a connection configuration decomposition relation, to which a constraint is attached.

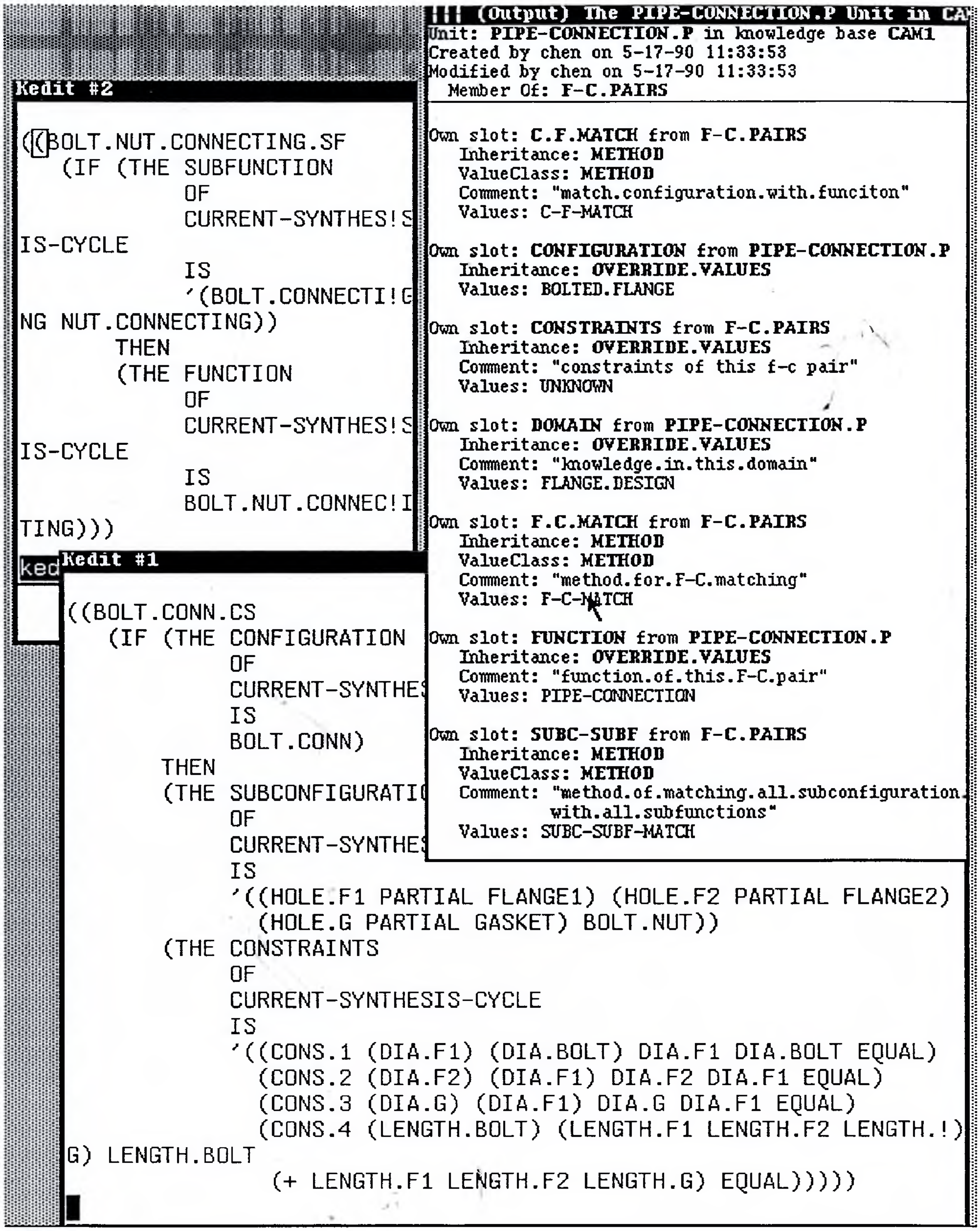
o Load the shell knowledge base, run and debug the synthesis knowledge base.

Thus far, the synthesis knowledge base has been implemented (Figure 6.17). The running windows are the same as Figure 6.5 with the two trees shown in Figure 6.15.

<u>Analysis Knowledge Base Development</u>

o Make a copy of the “analysis shell”, and rename it as “flange analysis knowledge base”.

o Define parameters and organize them according to the configurations.

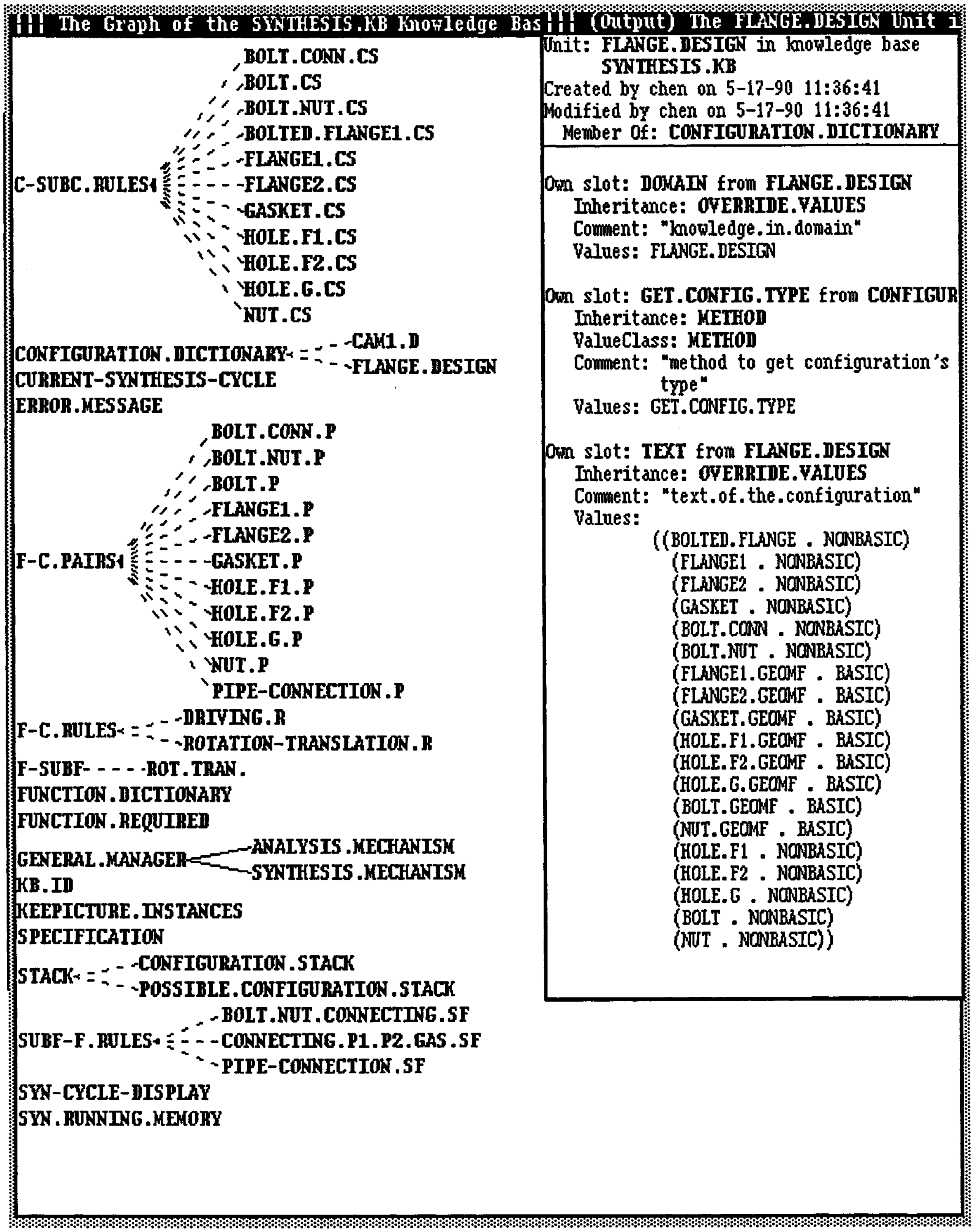


```
(Output) The PIPE-CONNECTION.P Unit in CA
Unit: PIPE-CONNECTION.P in knowledge base CAM1
Created by chen on 5-17-90 11:33:53
Modified by chen on 5-17-90 11:33:53
  Member Of: F-C.PAIRS

Own slot: C.F.MATCH from F-C.PAIRS
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "match.configuration.with.funciton"
   Values: C-F-MATCH

Own slot: CONFIGURATION from PIPE-CONNECTION.P
   Inheritance: OVERRIDE.VALUES
   Values: BOLTED.FLANGE

Own slot: CONSTRAINTS from F-C.PAIRS
   Inheritance: OVERRIDE.VALUES
   Comment: "constraints of this f-c pair"
   Values: UNKNOWN

Own slot: DOMAIN from PIPE-CONNECTION.P
   Inheritance: OVERRIDE.VALUES
   Comment: "knowledge.in.this.domain"
   Values: FLANGE.DESIGN

Own slot: F.C.MATCH from F-C.PAIRS
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "method.for.F-C.matching"
   Values: F-C-MATCH

Own slot: FUNCTION from PIPE-CONNECTION.P
   Inheritance: OVERRIDE.VALUES
   Comment: "function.of.this.F-C.pair"
   Values: PIPE-CONNECTION

Own slot: SUBC-SUBF from F-C.PAIRS
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "method.of.matching.all.subconfiguration.
            with.all.subfunctions"
   Values: SUBC-SUBF-MATCH
```

```
Kedit #2

((BOLT.NUT.CONNECTING.SF
   (IF (THE SUBFUNCTION
            OF
            CURRENT-SYNTHES!S
IS-CYCLE
            IS
            '(BOLT.CONNECTI!G
NG NUT.CONNECTING))
       THEN
       (THE FUNCTION
            OF
            CURRENT-SYNTHES!S
IS-CYCLE
            IS
            BOLT.NUT.CONNEC!I
TING)))
```

```
Kedit #1

((BOLT.CONN.CS
   (IF (THE CONFIGURATION
            OF
            CURRENT-SYNTHE
            IS
            BOLT.CONN)
       THEN
       (THE SUBCONFIGURATI
            OF
            CURRENT-SYNTHE
            IS
            '((HOLE.F1 PARTIAL FLANGE1) (HOLE.F2 PARTIAL FLANGE2)
              (HOLE.G PARTIAL GASKET) BOLT.NUT))
       (THE CONSTRAINTS
            OF
            CURRENT-SYNTHESIS-CYCLE
            IS
            '((CONS.1 (DIA.F1) (DIA.BOLT) DIA.F1 DIA.BOLT EQUAL)
              (CONS.2 (DIA.F2) (DIA.F1) DIA.F2 DIA.F1 EQUAL)
              (CONS.3 (DIA.G) (DIA.F1) DIA.G DIA.F1 EQUAL)
              (CONS.4 (LENGTH.BOLT) (LENGTH.F1 LENGTH.F2 LENGTH.!)
G) LENGTH.BOLT
               (+ LENGTH.F1 LENGTH.F2 LENGTH.G) EQUAL))))
```

Figure 6.16 : A "f–to–c" relation (upper right), a c–subc rule (lower) and a subf–f rule (upper left)



```
||| The Graph of the SYNTHESIS.KB Knowledge Bas

                              BOLT.CONN.CS
                              BOLT.CS
                              BOLT.NUT.CS
                              BOLTED.FLANGE1.CS
                              FLANGE1.CS
C-SUBC.RULES -----------------FLANGE2.CS
                              GASKET.CS
                              HOLE.F1.CS
                              HOLE.F2.CS
                              HOLE.G.CS
                              NUT.CS
                                 CAM1.D
CONFIGURATION.DICTIONARY -------
                                 FLANGE.DESIGN
CURRENT-SYNTHESIS-CYCLE
ERROR.MESSAGE
                         BOLT.CONN.P
                         BOLT.NUT.P
                         BOLT.P
                         FLANGE1.P
                         FLANGE2.P
F-C.PAIRS ---------------GASKET.P
                         HOLE.F1.P
                         HOLE.F2.P
                         HOLE.G.P
                         NUT.P
                         PIPE-CONNECTION.P
              DRIVING.R
F-C.RULES ----
              ROTATION-TRANSLATION.R
F-SUBF----ROT.TRAN.
FUNCTION.DICTIONARY
FUNCTION.REQUIRED
                      ANALYSIS.MECHANISM
GENERAL.MANAGER ------
                      SYNTHESIS.MECHANISM
KB.ID
KEEPICTURE.INSTANCES
SPECIFICATION
          CONFIGURATION.STACK
STACK ----
          POSSIBLE.CONFIGURATION.STACK
                  BOLT.NUT.CONNECTING.SF
SUBF-F.RULES -----CONNECTING.P1.P2.GAS.SF
                  PIPE-CONNECTION.SF
SYN-CYCLE-DISPLAY
SYN.RUNNING.MEMORY
```

```
||| (Output) The FLANGE.DESIGN Unit i
Unit: FLANGE.DESIGN in knowledge base
      SYNTHESIS.KB
Created by chen on 5-17-90 11:36:41
Modified by chen on 5-17-90 11:36:41
  Member Of: CONFIGURATION.DICTIONARY

Own slot: DOMAIN from FLANGE.DESIGN
   Inheritance: OVERRIDE.VALUES
   Comment: "knowledge.in.domain"
   Values: FLANGE.DESIGN

Own slot: GET.CONFIG.TYPE from CONFIGUR
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "method to get configuration's
            type"
   Values: GET.CONFIG.TYPE

Own slot: TEXT from FLANGE.DESIGN
   Inheritance: OVERRIDE.VALUES
   Comment: "text.of.the.configuration"
   Values:
           ((BOLTED.FLANGE . NONBASIC)
             (FLANGE1 . NONBASIC)
             (FLANGE2 . NONBASIC)
             (GASKET . NONBASIC)
             (BOLT.CONN . NONBASIC)
             (BOLT.NUT . NONBASIC)
             (FLANGE1.GEOMF . BASIC)
             (FLANGE2.GEOMF . BASIC)
             (GASKET.GEOMF . BASIC)
             (HOLE.F1.GEOMF . BASIC)
             (HOLE.F2.GEOMF . BASIC)
             (HOLE.G.GEOMF . BASIC)
             (BOLT.GEOMF . BASIC)
             (NUT.GEOMF . BASIC)
             (HOLE.F1 . NONBASIC)
             (HOLE.F2 . NONBASIC)
             (HOLE.G . NONBASIC)
             (BOLT . NONBASIC)
             (NUT . NONBASIC))
```

Figure 6.17 :The synthesis knowledge base (left) and the configuration dictionary (right)

o The solving and checking slices are listed in Figure 6.20. Two examples are given in Figures 6.18 and 6.19 (right portion).

o Design procedures: the list of procedures is shown in Figure 6.20.

o Knowledgeable Configuration Units ("c.static"s) : one example is shown in Figure 6.19 (left portion).

o Load the shell knowledge base (if it has not been loaded), run and debug the analysis knowledge base. Thus, the analysis knowledge base has been developed (Figure 6.20). The running facilities are identical to those in Figure 6.11.

The bolted flange knowledge bases are, therefore, complete.

6.2.2 A Bolted Flange Design Example

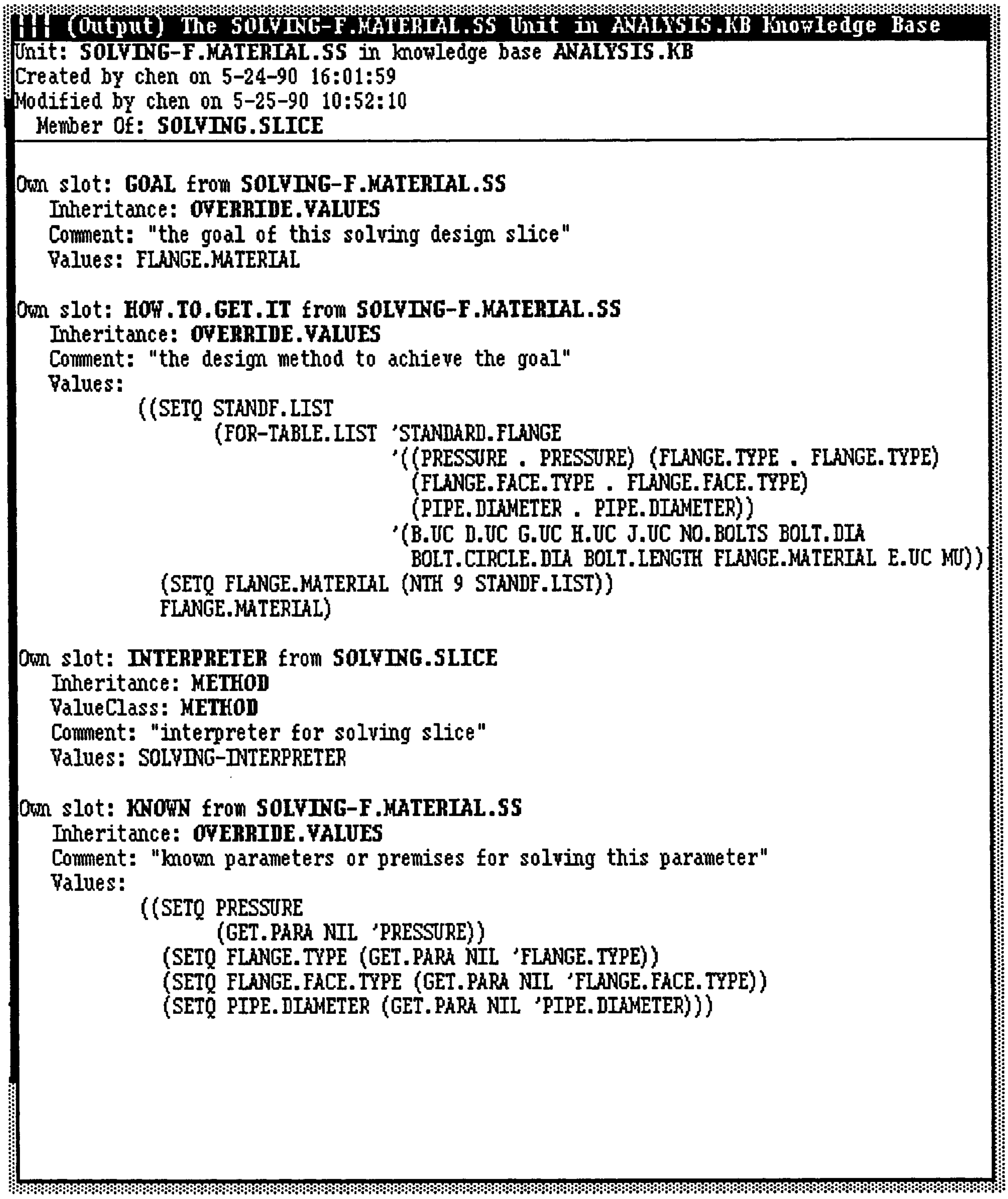
Given specifications:

Pipe pressure: 6 bar;
Temperature: 60^o ;
Pipe Diameter: 5";
Process Fluid: hot water;
Economic Consideration: designed economically;
Working Environment: used for pipe work, minimum cost, service condition general, not severe.

A number of the parameters and a graphics output (Figure 6.21) are shown as follows (the complete parameters are given in the Appendix) :

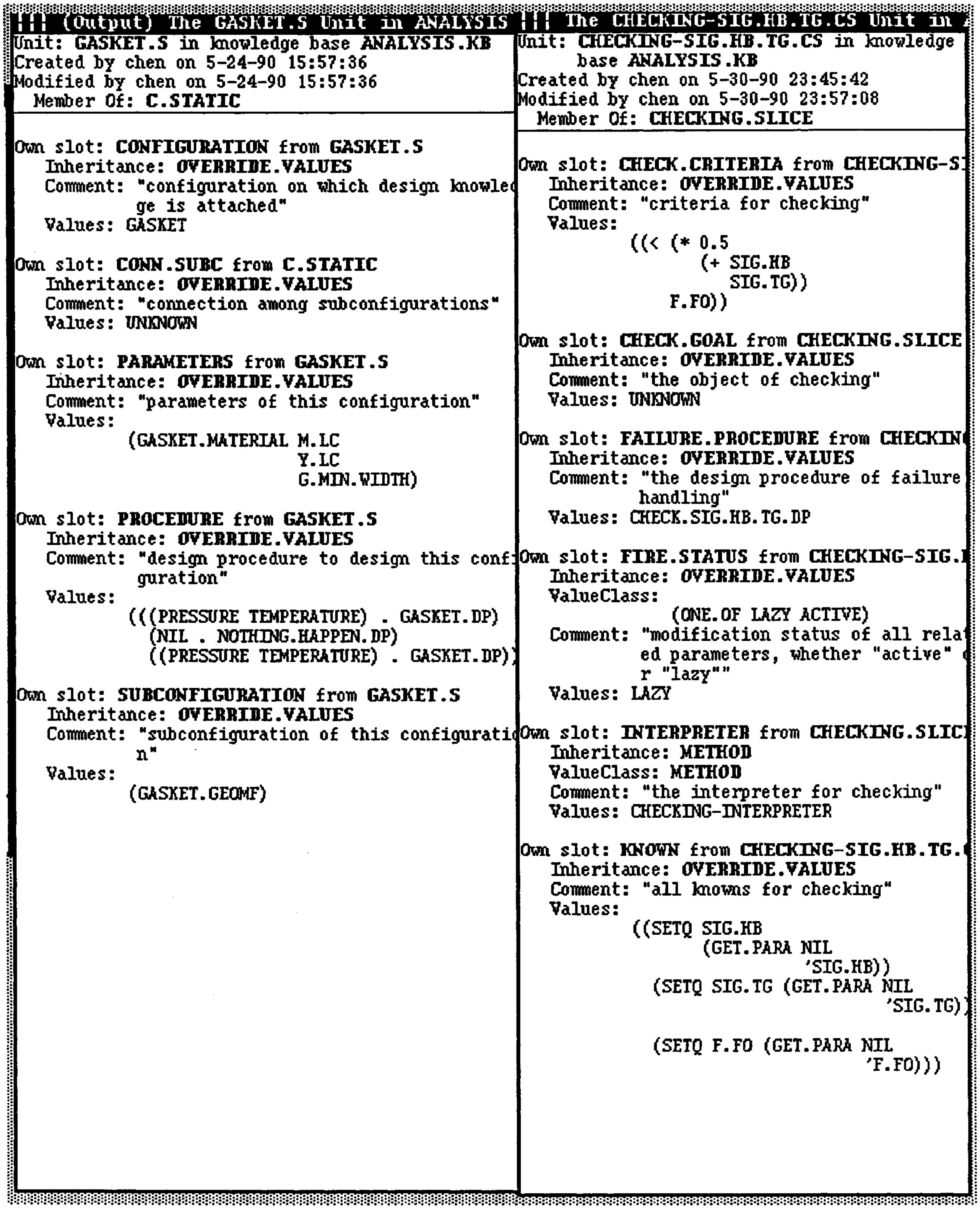
Configuration : BOLTED.FLANGE

FLANGE.FACE.TYPE : FULL–FACED
B.UC : 5.66
D.UC : 1.4375
G.UC : 6.4375
H.UC : 10.0
J.UC : 0.9375



```
(Output) The SOLVING-F.MATERIAL.SS Unit in ANALYSIS.KB Knowledge Base
Unit: SOLVING-F.MATERIAL.SS in knowledge base ANALYSIS.KB
Created by chen on 5-24-90 16:01:59
Modified by chen on 5-25-90 10:52:10
  Member Of: SOLVING.SLICE

Own slot: GOAL from SOLVING-F.MATERIAL.SS
   Inheritance: OVERRIDE.VALUES
   Comment: "the goal of this solving design slice"
   Values: FLANGE.MATERIAL

Own slot: HOW.TO.GET.IT from SOLVING-F.MATERIAL.SS
   Inheritance: OVERRIDE.VALUES
   Comment: "the design method to achieve the goal"
   Values:
           ((SETQ STANDF.LIST
                 (FOR-TABLE.LIST 'STANDARD.FLANGE
                                 '((PRESSURE . PRESSURE) (FLANGE.TYPE . FLANGE.TYPE)
                                   (FLANGE.FACE.TYPE . FLANGE.FACE.TYPE)
                                   (PIPE.DIAMETER . PIPE.DIAMETER))
                                 '(B.UC D.UC G.UC H.UC J.UC NO.BOLTS BOLT.DIA
                                   BOLT.CIRCLE.DIA BOLT.LENGTH FLANGE.MATERIAL E.UC MU)))
             (SETQ FLANGE.MATERIAL (NTH 9 STANDF.LIST))
             FLANGE.MATERIAL)

Own slot: INTERPRETER from SOLVING.SLICE
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "interpreter for solving slice"
   Values: SOLVING-INTERPRETER

Own slot: KNOWN from SOLVING-F.MATERIAL.SS
   Inheritance: OVERRIDE.VALUES
   Comment: "known parameters or premises for solving this parameter"
   Values:
           ((SETQ PRESSURE
                 (GET.PARA NIL 'PRESSURE))
             (SETQ FLANGE.TYPE (GET.PARA NIL 'FLANGE.TYPE))
             (SETQ FLANGE.FACE.TYPE (GET.PARA NIL 'FLANGE.FACE.TYPE))
             (SETQ PIPE.DIAMETER (GET.PARA NIL 'PIPE.DIAMETER)))
```

Figure 6.18 : A solving slice



```
||| (Output) The GASKET.S Unit in ANALYSIS
Unit: GASKET.S in knowledge base ANALYSIS.KB
Created by chen on 5-24-90 15:57:36
Modified by chen on 5-24-90 15:57:36
  Member Of: C.STATIC

Own slot: CONFIGURATION from GASKET.S
   Inheritance: OVERRIDE.VALUES
   Comment: "configuration on which design knowle
            ge is attached"
   Values: GASKET

Own slot: CONN.SUBC from C.STATIC
   Inheritance: OVERRIDE.VALUES
   Comment: "connection among subconfigurations"
   Values: UNKNOWN

Own slot: PARAMETERS from GASKET.S
   Inheritance: OVERRIDE.VALUES
   Comment: "parameters of this configuration"
   Values:
            (GASKET.MATERIAL M.LC
                             Y.LC
                             G.MIN.WIDTH)

Own slot: PROCEDURE from GASKET.S
   Inheritance: OVERRIDE.VALUES
   Comment: "design procedure to design this conf
            guration"
   Values:
            (((PRESSURE TEMPERATURE) . GASKET.DP)
              (NIL . NOTHING.HAPPEN.DP)
              ((PRESSURE TEMPERATURE) . GASKET.DP)

Own slot: SUBCONFIGURATION from GASKET.S
   Inheritance: OVERRIDE.VALUES
   Comment: "subconfiguration of this configurati
            n"
   Values:
            (GASKET.GEOMF)
```

```
||| The CHECKING-SIG.HB.TG.CS Unit in
Unit: CHECKING-SIG.HB.TG.CS in knowledge
      base ANALYSIS.KB
Created by chen on 5-30-90 23:45:42
Modified by chen on 5-30-90 23:57:08
  Member Of: CHECKING.SLICE

Own slot: CHECK.CRITERIA from CHECKING-S
   Inheritance: OVERRIDE.VALUES
   Comment: "criteria for checking"
   Values:
           ((< (* 0.5
                  (+ SIG.HB
                     SIG.TG))
               F.FO))

Own slot: CHECK.GOAL from CHECKING.SLICE
   Inheritance: OVERRIDE.VALUES
   Comment: "the object of checking"
   Values: UNKNOWN

Own slot: FAILURE.PROCEDURE from CHECKIN
   Inheritance: OVERRIDE.VALUES
   Comment: "the design procedure of failure
            handling"
   Values: CHECK.SIG.HB.TG.DP

Own slot: FIRE.STATUS from CHECKING-SIG.
   Inheritance: OVERRIDE.VALUES
   ValueClass:
              (ONE.OF LAZY ACTIVE)
   Comment: "modification status of all rela
            ed parameters, whether "active"
            r "lazy""
   Values: LAZY

Own slot: INTERPRETER from CHECKING.SLIC
   Inheritance: METHOD
   ValueClass: METHOD
   Comment: "the interpreter for checking"
   Values: CHECKING-INTERPRETER

Own slot: KNOWN from CHECKING-SIG.HB.TG.
   Inheritance: OVERRIDE.VALUES
   Comment: "all knowns for checking"
   Values:
           ((SETQ SIG.HB
                  (GET.PARA NIL
                            'SIG.HB))
             (SETQ SIG.TG (GET.PARA NIL
                                    'SIG.TG)

             (SETQ F.FO (GET.PARA NIL
                                  'F.FO)))
```

Figure 6.19 : A knowledgeable configuration unit (left) and a checking slice (right)

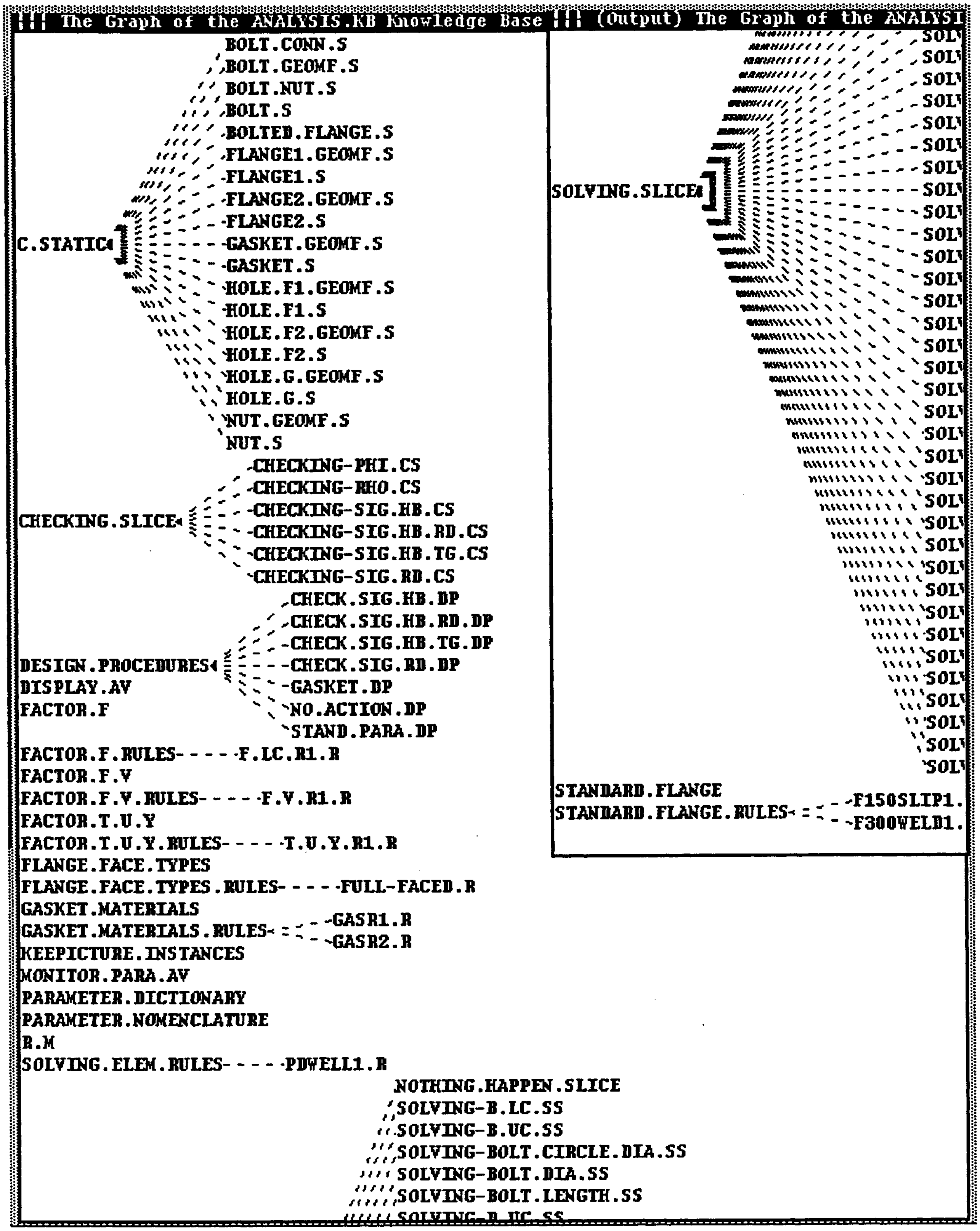
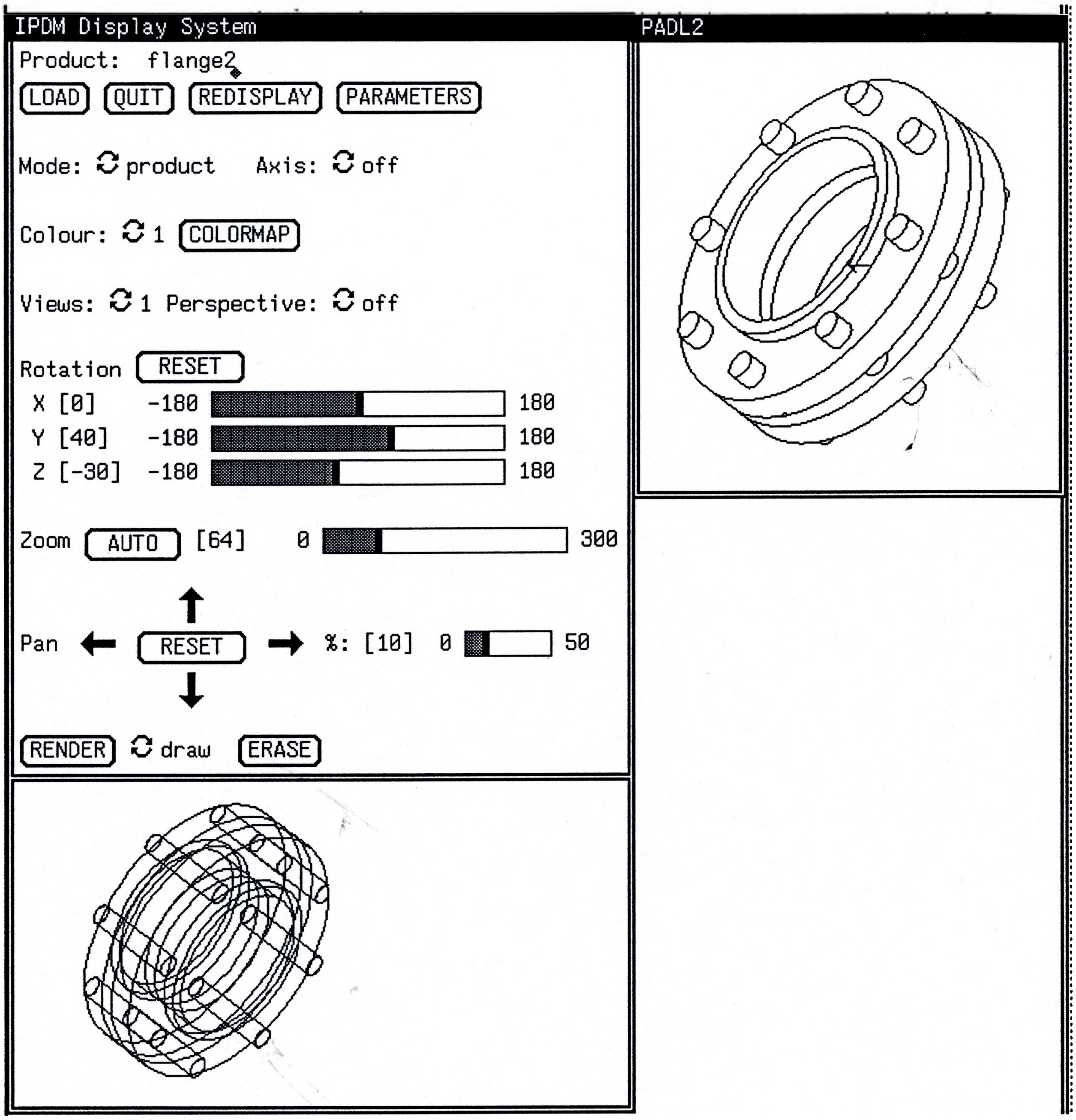


Figure 6.20 : Flange analysis knowledge base



Product: flange2

Mode: product   Axis: off

Colour: 1

Views: 1 Perspective: off

Rotation
X [0] -180 180
Y [40] -180 180
Z [-30] -180 180

Zoom [64] 0 300

Pan %: [10] 0 50

draw

Figure 6.21 : Flange geometry model produced by the IPDM display interface

```
    NO.BOLTS : 8.0
    BOLT.DIA : 0.75
    BOLT.CIRCLE.DIA : 8.5
    BOLT.LENGTH : 3.75
    FLANGE.MATERIAL : FORGED.STEEL.SA181
Configuration : GASKET
   GASKET.MATERIAL : VEGETABLE.FIBRE
   M.LC : 1.75
   Y.LC : 7.6
   G.MIN.WIDTH : 10
```

6.2.3 Comments

The bolted flange design differs considerably from the cam design, yet it is very easy to develop both systems based on the same shell system. This fact proves the feasibility and advantage of this design shell approach.

The system is constructed as simply as possible, but is relatively complete with approximately 124 items (25 in synthesis and 99 in analysis). The knowledge bases can be easily extended to cover more design cases.

As in the case of the cam system, the system performance is as good as the designer(s).

It takes the author approximately one month to implement this flange system. From this point of view, this work is successful.

Besides the simplicity of the knowledge accumulation, it is also flexible enough to extend the representation scheme by creating new basic elements once new knowledge types appear. For example, fetching parameters from a figure emerged as a type during the flange system implementation. A new basic element, perhaps named "figure", should be constructed. However, considering the fact that the "figure" is mainly represented in mathematical formulae and, thus, the task is left to create a user interface for deriving these formulas from the figure inputted, a temporary alternative is utilized by representing a figure in the same way as a data table.

Thus far, another domain specific design system, the bolted flange system, has been developed.

CHAPTER 7 : CONCLUSIONS AND DISCUSSION

7.1 SUMMARY OF THE THESIS WORK

The objective established for this work, a general approach to facilitate design automation in a certain range of routine mechanical design activities, has been achieved. This approach consists of a general model and a shell system. By filling domain knowledge into this shell system, specific design systems can be developed efficiently.

This general approach was achieved by following a clear, logical line of research:

1. Conduct a literature survey of this research area;
2. Understand the problem, the design process, by observing some design cases completed by experienced designers and referring to some prescriptive design theories, in order to get insight into the common characteristics of the design activities;
3. Propose a general model, based on the common characteristics from the understanding, and aim at laying some foundation for further computer implementation;
4. Implement this model without considering any specific machine environment;
5. Develop an experimental shell system based on the above conceptual implementation, using KEE and Common LISP on a SUN workstation;
6. Develop two specific design systems to prove the feasibility of this general approach.

The following evaluation of this work shows the comparison with current research work in this area. The contributions from this work are pointed out, as well as the limitations and possible future developments.

First, a brief review of some conclusions from the Literature Survey chapter is reiterated here.

7.2 REVIEW OF CURRENT RESEARCH WORK

The current research work in this topic is primarily concerned with three issues: 1) parametric design; 2) conceptual and configuration design; and 3) geometric aspect in design. The limitations in current research in these areas are listed as follows:

- The design process has not yet been well understood;
- Processes for the achievement of configurations from functions and the evaluation of configurations without assigning values remain open research topics;
- Design knowledge is large and diverse. The current representation tools are not powerful enough to handle it;
- The description tool for design knowledge is the key issue in the expert system application to this area, because the purpose of this application is to facilitate the spreading of knowledge;
- The geometry and parameters are two inseparable aspects in design processes. The geometric aspect in design has not yet been well studied;
- The design process has not been well studied as an integrated process. The above three issues: parametric design, conceptual and configuration design, and the geometric aspect in design, have been researched primarily on an individual basis. Some new features might be exposed if they are viewed as one process.

These problems partially contributed to the motivation of this thesis work.

7.3 CONTRIBUTIONS

The following thesis contributions can be identified:

- The whole design process has been studied, which is divided into four stages: specification development, synthesis, analysis and non–functional considerations;
- The approach developed in this work, which consists of a model, a shell system and a

procedure to develop domain specific design systems, is domain independent of certain routine mechanical component and mechanism design processes;

- The descriptive style for knowledge representation has been emphasized for the ease of developing domain specific design systems based on the shell system;
- The Configuration Decomposition Approach has been developed for the synthesis stage. It is effective in the routine mechanical design;
- The multilevel control mechanisms and general knowledge representation formats achieve the descriptive approach to implementing of the analysis stage, which is characterized as having diverse and large amounts of design knowledge;
- The geometric aspect is handled parametrically based on a feature–based modelling system;
- Other problems studied include dependency in design, which refers to the relationship among parameters and which causes the iteration process; and connection design, which brings in a basic knowledge type, the constraint;
- Although the work focuses on the routine mechanical design, it certainly has the potential for the expansion of further research work in this area.

7.4 COMPARISON WITH SOME RESEARCHERS' WORK

To further enhance the description of the contributions from this thesis, a comparison of this thesis work with other researchers' work follows. These researchers have been introduced in the Literature Survey chapter in a general sense. Now the difference between this thesis work and their work is to be clearly pointed out.

Dixon and his group focus on the algorithmic determination of design parameters. Their DOMINIC is similar to an optimization program in terms of the algorithm or reasoning process. The designer's knowledge is not obvious in their system. On the contrary, this thesis work is directed toward the descriptive style of design knowledge expression. The design process is described by explicit elements following the experienced human designer's procedure, not "programmed". In other words, the design process is handled in an explicit, "visible" style in this thesis work, while it is dealt with in a less visible manner by DOMINIC.

The "design with features" work from Dixon's group which deals with the interactive design of non–standard components is not addressed in this work. This thesis topic is the automatic implementation of the whole design process from synthesis to analysis, which differs from the issue of "design with features". However, a features modeller — IPDM has been used in this work as a basic support tool for the design process.

The characteristics of Langrana's group at Rutgers are similar to Dixon's group in terms of the manner in which they express the design knowledge. Both DPMED and DOMINIC describe the design process in a more algorithmic style. The difference between their work and this thesis work is obvious and fundamental: one is algorithmic and less "visible", the other is explicit. There are advantages and disadvantages of each style. The algorithmic style achieves the design results according to both designers' experience and optimization guidance. Thus, the design results may be better than the explicit style. The explicit style follows the experienced designers' "steps" in their design process straightforwardly. This explicit description style makes the development and modification of knowledge bases quite simple, but it limits the design quality to be at most as good as those produced by experienced designers (not a single designer, but multiple experts). A better approach could be a combination of both: the explicit descriptive style for the initial solution, and the algorithmic style for optimizing the solution.

Brown and Chandrasekaran's work (DSPL) has great similarity with the work of Mittal, et al. (PRIDE) from the Xerox Research Group (as concluded from the papers listed in the

Bibliography). Although the design problems addressed by these two groups are different, their basic ideas are similar. That is, both of them implement the explicit descriptive style in expressing design knowledge. From this point of view, this thesis work falls into the same category as theirs. All promote the explicit descriptive style in expressing design knowledge. The major difference between this thesis and their work, or the distinct work from this thesis, which is not shown in any previous work, is the idea of creating a group of basic description elements. The design knowledge can be described and represented by this group of basic elements. These elements are general to a number of design activities, and can be easily modified and expanded. Thus, the development and modification of knowledge bases is quite simple.

In addition to the above difference, this thesis addresses the whole design process which begins with specification development, goes through the configuration decomposition synthesis, and ends with solving of all parameters and graphics output of all configurations. In other words, the complete design process is covered in this thesis.

The conceptual and configuration design in this thesis follows a distinct approach from other researchers' work: Configuration Decomposition considering functional requirements. This approach is a combination of function considerations and the decomposition of configurations. It originated from the nature of this research problem: the routine mechanical design. Unlike researchers such as Rinderle, Struss and others, who are trying to develop a "deep" model of function and configuration relations, this work implements the explicit function to configuration relations and places the functions on the secondary level after the "principal function" has been considered.

Regarding the geometry issue, the geometry and spatial reasoning is not dealt with in this thesis, as it is in the work of Fenves and Baker, Nevill and others. The reason for this omission also arises from the nature of the problem. This reasoning process in the routine mechanical design activities is not as complicated as in creative and new design. Thus, a simple approach serves the purpose in this case. Therefore, the geometry issue in this work is handled parametrically based on a feature-based modelling system — the IPDM.

In summary, the work of the conceptual and configuration design in this thesis does not address the "deep" issues, such as function driven design and the spatial and geometry reasoning, as some researchers are trying to do. Because of the routine mechanical design nature, the work of achieving a model is guided by the idea that the model should reflect the characteristics of this kind of design process and achieve satisfactory results in its implementation.

After the above comparison, the distinct features of this thesis can be clearly summarized as follows:

- The design knowledge is represented in the explicit descriptive style, which is based on a group of basic description elements and organized in a four level hierarchy structure;
- The whole design process is covered;
- Because of the nature of the routine mechanical design problem, the conceptual and configuration design is handled by a configuration decomposition approach while the functional requirements are also considered, and the geometry aspect is dealt with parametrically based on a feature-based modelling system.

7.5 FUTURE DEVELOPMENTS

Some limitations of this thesis work and possible further developments are summarized as follows:

- Feedback information from the synthesis or the analysis stages has not been used to guide the re-specification or re-synthesis. The characteristics of this iterative process should be studied and implemented, as it certainly occurs in some kinds of design activities;
- The geometric aspect is handled parametrically based on a feature-based modelling sys-

tem, which characterizes the routine mechanical design, but not the "pure" geometric reasoning process which may occur in non–standard component design activities and others;

- In the conceptual design, the "pure" function driven design process has not been fully studied in the present work.

BIBLIOGRAPHY

[1] Adelson, B. (1989), Cognitive Research: Uncovering How Designers Design; Cognitive Modelling: Explaining and Predicting How designers Design, Research in Engineering Design, Vol.I, No I, pp. 35–42.

[2] Agogino, A.M. and Almgren, A.S. (1987), Symbolic Computation in Computer–aided Optimal Design, Expert Systems in Computer–Aided Design, Gero, J.S., ed., North–Holland, Amsterdam, pp. 267–284.

[3] Allen, R.H., Goarnet, M.G., Culbert, C.J. and Savely, R.T. (1987), The Nature and Evaluation of Expert Systems Tools for Engineering Applications, Proceeding of 1987 ASME Computers in Engineering, New York, pp. 143–154.

[4] Asimow, M. (1962), Introduction to Design, Prentice–Hall, Englewood Cliffs, NJ.

[5] Brown, D.C. (1985), Capturing Mechanical Design Knowledge, Proceeding of 1985 ASME Computers in Engineering, Boston, Massachusetts, August.

[6] Brown, D.C. and Chandrasekaran, B. (1985), Expert Systems for a Class of Mechanical Design Activity, In: Knowledge Engineering in Computer–Aided Design, North–Holland, November, pp. 259–282.

[7] Brown, D.C. (1985), Failure Handling in a Design Expert System, Computer–Aided Design, November, pp. 436–442.

[8] Brown, D.C. and Chandrasekaran, B. (1986), Knowledge and Control for a Mechanical Design Expert Systems, IEEE Computer Magazine, Special Issue on Expert Systems for Engineering Problems, July, pp. 92–100

[9] Brown, D.C. and Breau, R. (1986), Types of Constraints in Routine Design Problem–Solving, The First International Conference on Applications of Artificial Intelligence to Engineering Problems, Southampton University, UK, April, pp. 383–390

[10] Brown, D.C. and Sloan, W.N. (1987), Compilation of Design Knowledge for

Routine Design Expert Systems: an Initial View, Proceeding of 1987 ASME Computers in Engineering, New York, August, pp. 131–136.

[11] Chen, S., Sun, H. and Zhang, G. (1987), An Artificial Intelligent Approach to CAD/CAM, Proceeding of 1987 ASME Computers in Engineering, New York, pp. 173–177.

[12] Chieng, W.H. and Hoeltzel, D.A. (1986), An Interactive Hybrid System Approach to Near Optimal Design of Mechanical Components, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, pp. 149–159.

[13] Choy, J.K. and Agogino, A.M. (1986), SYMON: Automated Symbolic Monotonicity Analysis System for Qualitative Design Optimization, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, August, pp. 207–212.

[14] Coulson, J.M., Richardson, J.F. and Sinnott, R.K. (1983), Chemical Engineering, Volume 6, Design, Pergamon Press Canada Ltd., Willowdale, Canada.

[15] Cunningham, J.J. and Dixon, J.R. (1988), Designing with Features: The Origin of Features, Proceeding of 1988 ASME Computers in Engineering, pp. 237–243.

[16] Cutkosky, M.R., Tenenbaum, J.M. and Muller, D. (1988), Features in Process-Based Design, Proceeding of 1988 ASME Computers in Engineering, pp. 557–562.

[17] Dixon, J.R. (1966), Design Engineering: Inventiveness, Analysis, and Decision Making, McGraw-Hill, New York.

[18] Dixon, J.R., Simmons, M.K. and Cohen, P.R. (1984), An Architecture for Applying Artificial Intelligence to Design, Proceeding of the 21st ACM/IEEE Design Automation conference, Albuquerque, NM, June 25–27.

[19] Dixon, J.R., et al. (1986), Dominic I: Progress Towards Domain Independence in Design by Iterative Redesign, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, pp. 199–206.

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

[21] Dixon, J.R., et al. (1987), Expert Systems for Mechanical Design: Examples of

Symbolic Representations of Design Geometries, Engineering with Computers, vol. 2, New York.

[22] Dixon, J.R., et al. (1988), A Proposed Taxonomy of Mechanical Design Problems, Proceeding of 1988 ASME Computers in Engineering, pp. 41–46.

[23] Dixon, J.R. (1988), Designing with Features: Building Manufacturing Knowledge into More Intelligent CAD Systems, Manufacturing International 88', pp. 51–57.

[24] Duffey, M.R. and Dixon, J.R. (1988), Automating the Design of Extrusions: A Case Study in Geometric and Topological Reasoning for Mechanical Design, Proceeding of 1988 ASME Computers in Engineering, pp. 505–511.

[25] Dyer, M.G., Flowers, M. and Hodges, J. (1986), Edison: An Engineering Design Invention System Operating Naively, Proceeding of the First International Conference on Applications of Artificial Intelligence to Engineering Problems, Southampton University, UK, April, pp. 327–341.

[26] ElMaraghy, H.A. (1990), Intelligent Product Design and Manufacture, to appear in AI in Design, Editors, D.T.Pham, IFS/Springer–Verlag AI in Industry Series

[27] ElMaraghy, H.A. and Chen, S. (1990), A General Model for Mechanical Design, CIRP Annals 1990, Manufacturing Technology, vol. 39/1, pp. 111–116.

[28] ElMaraghy, H.A. and Knoll, L. (1990), IPDM -- Intelligent Product Design and Manufacture, Proceeding of MAPLE 1990, Ottawa, Ontario, Canada, May 14–15, pp. 67–79.

[29] Fenves, S.J. and Baker, N.C. (1987), Spatial and Functional Representation Language for Structural Design, IFIP WG5.2 Expert Systems in Computer–Aided Design, Sydney, Australia, pp. 511–530.

[30] Finger, S. and Dixon, J.R. (1989), A review of research in mechanical engineering design. Part 1: descriptive, prescriptive and computer–based models of design processes, Research in Engineering Design, vol.I, No I, pp. 51–67.

[31] Finger, S. and Rinderle, J.R. (1989), A Transformational Approach to Mechanical Design Using A Bond Graph Grammar, Proceeding of the First International Conference on

Design Theory and Methodology, Montreal, Canada, pp. 107–116.

[32] Freeman, P. and Newell, A. (1971), A Model for Functional Reasoning in Design, Proceeding of The Second IJCAI, London, England, pp. 621–633.

[33] Gero, J.S. and Balachandran, M. (1986), Knowledge and Design Decision Processes, Proceeding of the First International Conference on Applications of Artificial Intelligence to Engineering Problems, Southampton University, UK, April, pp. 343–352.

[34] Gu, P.H., ElMaraghy, H.A. and Hamid, L. (1989), FDDL: A Feature Based Design Description Language, Proceeding of 1989 ASME Conference on Design Automation, Montreal, Canada, September, pp. 53–63.

[35] Gupta, A. (1987), Explanation–Based Failure Recovery, National Conference on Artificial Intelligence, pp. 606–610.

[36] Hirschtick, J.K. and Gossard, D.C. (1986), Geometric Reasoning for Design Advisory Systems, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, pp. 263–270

[37] Howe, A., Cohen, P. and Dixon, J.R. (1987), Dominic: A Domain–Independent Program for Mechanical Engineering Design, The First Conference of A.I. on Engineering Problems, pp. 289–299.

[38] Hodges, J., Flowers, M. and Dyer, M. (1987), Knowledge Representation for Design Creativity, the 1987 Winter Annual Meeting of the ASME, pp. 81–93.

[39] Intellicorp (1987), KEE User Manuals

[40] Ishii, K. and Barkan, P (1987), Rule–Based Sensitivity Analysis, IFIP WG5.2 Expert Systems in Computer–Aided Design, Sydney, Australia, pp. 179–198.

[41] Ishii, K. and Barkan, P. (1987), Design Compatibility Analysis – A Framework for Expert Systems in Mechanical System Design, Proceeding of 1987 ASME Computers in Engineering, New York, pp. 95–102.

[42] Joskowicz, L. (1987), Shape and Function in Mechanical Devices, National Conference on Artificial Intelligence, pp. 611–615.

[43] Knoll, L. (1989), IPDM Status Report, FMRD Center Technical Report IPDM-24, Centre for Flexible Manufacturing Research and Development, McMaster University.

[44] Kott, A.S. and May, J.H. (1989), Decomposition vs. transformation: case studies of two models of the design process, Proceeding of 1989 ASME Computers in Engineering, pp. 1-8.

[45] Krick, E.V. (1967), An Introduction to Engineering and Engineering Design, Wiley, NY.

[46] Kroll, E., Lenz, E. and Wolberg, J.R. (1988), A Knowledge-Based Solution to the Design- for-Assembly Problem, Manufacturing Review, vol 1, no 2, June, pp. 104-108.

[47] Lai, K. and Wilson, W.R.D. (1987), FDL - A Language for Function Description and Rationalization in Mechanical Design, Proceeding of 1987 ASME Computers in Engineering, New York, pp. 87-94.

[48] Langrana, N.A., Mitchell, T.M. and Ramchandran, N. (1986), Progress Toward A Knowledge-Based Aid for Mechanical Design, the 1986 Winter Annual Meeting of the ASME, pp. 45-61.

[49] Latombe, J.C. (1976), Artificial Intelligence in Computer Aided Design: The TROPIC System, Technical Report 125, Stanford Research Institute, February.

[50] Lian, H.Y. (1990), Private Consultation with Professor Lian, TianJin University, China, currently visiting scholar at McMaster University

[51] Lu, S.C-Y., Subramanyam, S. and Thompson, J.B., Klein, M. (1989), A Cooperative Product Development Environment to Realize the Simultaneous Engineering Concept, Proceeding of 1989 ASME Computers in Engineering, pp. 9-18.

[52] Luby, S.C., Dixon, J.R. and Simmons, M.K. (1986), Designing with Features: Creating and Using a Features Data Base for Evaluation of Manufacturability of Castings, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, pp. 285-292.

[53] Maher, M.L. (1985), HI-RISE: A Knowledge-Based Expert System for the Preliminary Structural Design of High Rise Buildings, unpublished Ph.D. Dissertation, Dept of

Civil Engineering, CMU, January, pp. 125–135.

[54] Maher, M.L. and Zhao, F. (1987), Using Experience to Plan the Synthesis of New Designs, IFIP WG5.2 Expert Systems in Computer–Aided Design, Sydney, Australia, pp. 349–369.

[55] Martin, G.H. (1969), Kinematics and Dynamics of Machines, McGraw–Hill, New York.

[56] Megyesy, E.F. (1981), Pressure Vessel Handbook, Fifth Edition, Pressure Vessel Handbook Publishing, Inc., Tulsa, Oklahoma, USA.

[57] Meunier, K.L. and Dixon, J.R. (1988), Iterative Respecification: A Computational Model for Hierarchical Mechanical system Design, Proceeding of 1988 ASME Computers in Engineering, pp. 25–32.

[58] Mittal, S., Dym, C.L. and Morjaria, M. (1986), PRIDE: An Expert System for the Design of Paper Handling System, Computer, July, pp. 102–114.

[59] Mittal, S. and Araya, A. (1986), A Knowledge–Based Framework for Design, Proceeding of AAAI–86, Philadelphia, PA., August, pp. 856–865.

[60] Mittal, S. and Frayman, F. (1987), Making Partial Choices in Constraint Reasoning Problems, National Conference on Artificial Intelligence, pp. 632–636.

[61] Mostow, J. (1985), Toward Better Models of The Design Process, AI Magazine, Spring.

[62] Murthy, S.S. and Addanki, S. (1987), PROMPT: An Innovative Design Tool, National Conference on Artificial Intelligence, pp. 637–642.

[63] Nevill, G.E., Jr. and Paul, G.H., Jr. (1987), Knowledge–Based Spatial Reasoning for Designing Structural Configurations, Proceeding of 1987 ASME Computers in Engineering, New York, pp. 155–160.

[64] Nevill, G.R., Jr., et al. (1989), Creating Abstraction/Feature Sets for Top Down Mechanical Design: Experiences with Automating Preliminary Design of Structures, Proceeding of 1989 ASME Computers in Engineering, pp. 273–279

[65] Newcombe, W.R. (1989), Advanced Kinematics of Machines, Lecture Notes, McMaster University.

[66] Orelup, M.F., Dixon, J.R. and Simmons, M.K. (1987), Dominic II: More Progress Towards Domain Independent Design by Iterative Redesign, The Winter Annual Meeting of the ASME, Boston, December, pp. 67–80.

[67] Pahl, G. and Beitz, W. (1988), Engineering Design — A Systematic Approach, ed. K.Wallace, Springer–Verlag.

[68] Pfaff, T. (1990), IPDM Status Report, Technical Report FMRD–11–03–90/ IPDM–25, Centre for Flexible Manufacturing Research and Development, McMaster University.

[69] Pratt, M.J. (1988), Synthesis of An Optimal Approach to Form Feature Modelling, Proceeding of 1988 ASME Computers in Engineering, pp. 263–274.

[70] Ramchandran, N., Shah, A. and Langrana, N.A. (1988), Expert System Approach in Design of Mechanical Components, Proceeding of 1988 ASME Computers in Engineering, pp. 1–10.

[71] Ravi, T. (1989), KEE Overview, FMRD Centre Technical Report, Centre for Flexible Manufacturing Research and Development, McMaster University.

[72] Rehak, D.R., Howard, H.C. and Sriram, D. (1985), Architecture of an Integrated Knowledge Based Environment for Structural Engineering Applications, IFIP WG5.2 Knowledge Engineering in Computer–Aided Design, J.S.Gero (edited), Hungary, pp. 89–117.

[73] Rindele, J.R. (1986), Implications of Function–Form–Fabrication Relations on Design Decomposition Strategies, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, pp. 193–198.

[74] Rothbart, H.A. (1956), CAMS, John Wiley

[75] Schmekel, H. (1989), Functional Models and Design Solutions, CIRP Annals 1989, Manufacturing Technology, General Assembly, Trondheim, vol. 38/1, pp. 129–132.

[76] Shah, J.J. and Pandit, L. (1986), Deziner – An Expert System for Conceptual Form

Design of Structural Parts, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, pp. 167–172.

[77] Shah, J.J. (1988), Shape Algebra for Structural Configuration Synthesis, Proceeding of 1988 ASME Computers in Engineering, San Francisco.

[78] Shapiro, V. and Voelcker, H. (1989), On the Role of Geometry in Mechanical Design, Research in Engineering Design, vol.I, No I, pp. 69–73.

[79] Simmons, M.K. and Dixon, J.R. (1986), Reasoning about Quantitative Methods in Engineering Design, Coupling Symbolic and Numerical Computing in Expert Systems, North–Holland, pp. 47–57.

[80] Sriram, D. and Maher, M.L. (1987), The Representation and Use of Constraints in Structural Design, The First International Conference of AI on Engineering Problems, pp. 355–368.

[81] Steinberg, L.I. (1986), Design as Refinement Plus Constraint Propagation: The VEXED Experience", Proceeding of the Sixth National Conference on Artificial Intelligence, Seattle, WA, pp. 830–835.

[82] Struss, P. (1987), Multiple Representation of Structure and Function, IFIP WG5.2 Expert Systems in Computer–Aided Design, Sydney, Australia, pp. 57–92.

[83] Suh, N.P. (1990), The Principles of Design, MIT, Oxford University Press.

[84] Tikerpuu, J. and Ullman, D.G. (1988), General Feature–Based Frame Representation for Describing Mechanical Engineering Design Developed From Empirical Data, Proceeding of 1988 ASME Computers in Engineering, pp. 245–253.

[85] Tomiyama, T. and Yoshikawa, H. (1985), Requirements and Principles for Intelligent CAD Systems, IFIP WG5.2 Knowledge Engineering in Computer–Aided Design, J.S.Gero (edited), Hungary, pp. 1–23.

[86] Tomiyama, T. and Hagen, P.J.W.T. (1987), Organization of Design Knowledge in an Intelligent CAD Environment, IFIP WG5.2 Expert Systems in Computer–Aided Design, Sydney, Australia, pp. 119–147.

[87] Turner, G.P. and Anderson, D.C. (1988), An Object-Oriented Approach to Interactive, Feature-Based Design for Quick Turnaround Manufacturing, Proceeding of 1988 ASME Computers in Engineering, pp. 551–555.

[88] Ullman, D.G. and Dietterich, T.A. (1986), Mechanical Design Methodology: Implications on Future Developments of Computer-Aided Design and Knowledge-Based Systems, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, pp. 173–180.

[89] Ullman, D.G. (1989), A Taxonomy of Mechanical Design, Proceeding of the First International Conference on Design Theory and Methodology, Montreal, Canada, pp. 23–36.

[90] Ulrich, K. and Seering, W. (1987), Conceptual Design: Synthesis of Systems of Components, the 1987 Winter Annual Meeting of the ASME, pp. 57–66.

[91] Verrilli, R.J., Meunier, K.L., Dixon, J.R. and Simmons, M.K. (1987), Iterative Respecification Management: A Model for Problem-Solving Networks in Mechanical Design, Proceeding of 1987 ASME Computers in Engineering, New York, pp. 103–112.

[92] Waters, E.O., Wesstrom, D.B. and Williams, F.S.G. (1934), Design of Bolted Flanged Connections, Pressure Vessel and Piping Design, Collected Papters, 1927–1959, The American Society of Mechanical Engineers, New York, pp. 58–61.

[93] Waters, E.O., Wesstrom, D.B., Rossheim, D.B. and Williams, F.S.G. (1937), Formulas for Stresses in Bolted Flanged Connections, Pressure Vessel and Piping Design, Collected Papters, 1927–1959, The American Society of Mechanical Engineers, New York, pp. 62–82.

[94] Welch, R.V. and Dixon, J.R. (1989), Extending the Iterative Redesign Model to Configuration Design: Sheet Metal Brackets as an Example, Proceeding of the First International Conference on Design Theory and Methodology, Montreal, Canada, pp. 81–88.

[95] Zarefar, H., Lawley, T.J. and Etesami, F. (1986), PAGES: A Parallel Axis Gear Drive Expert System, Proceeding of 1986 ASME Computers in Engineering, Chicago, IL, pp. 145–147.

APPENDIX

CAM EXAMPLE1

Specifications 163

Results 164

PDL File 166

CAM EXAMPLE2

Specifications 167

Results 168

FLANGE EXAMPLE1

Specifications 170

Results 171

PDL File 173

FLANGE EXAMPLE2

Specifications 176

Results 177

SPECIFICATIONS

```
item  : CYCLE.TIME;
value : 0.05;

item  : R.B;
value : 1.25;

item  : ANGULAR.VELOCITY;
value : NIL;

item  : RISE.SUGG.MOTION;
value : NIL;

item  : R.FOLLOWER;
value : 0.5;

item  : RISE.TIME;
value : NIL;

item  : RISE.REQ;
value : (NO STEPS IN ACCELERATION STEPS IN JERK);

item  : RISE;
value : 2.0;

item  : DWELL2.SUGG.MOTION;
value : NIL;

item  : DWELL1.SUGG.MOTION;
value : NIL;

item  : DWELL1.TIME;
value : NIL;

item  : DWELL1.REQ;
value : (A SMALL DWELL);

item  : DWELL2.REQ;
value : NIL;

item  : DWELL2.TIME;
value : 0.015;

item  : RETURN.TIME;
value : NIL;

item  : RETURN.REQ;
value : NIL;

item  : RETURN.SUGG.MOTION;
value : SHM;

item  : PRESS.ANGLE;
value : 30;

item  : MOTION.PATTERN;
value : DRD;
```

```
          RESULTS

Configuration : CAM-SYSTEM
   ANGULAR.VELOCITY :  125.663605
   PRESS.ANGLE :  28.904034

Configuration : CAM
   PD1 :  10.0
   PD2 :  108.0
   PRI :  135.0
   PRE :  107.0
   R.P :  2.75
   OFFSET.E :  0.5

Configuration : FOLLOWER
   R.F :  NIL
   D.CYLINDER :  1.0
   H.CYLINDER :  5.0
   X0 :  1.25
   Y0 :  0
   Z0 :  0
   A :  0.0
   B :  0.0
   C :  0.0

Configuration : DWELL1
   S :  1.25
   BETA :  10.0
   D :  0.0
   H :  5.0
   X0 :  0
   Y0 :  0
   Z0 :  0
   A :  0.0
   B :  0.0
   C :  350.0

Configuration : RISE
   S :  1.25
   BETA :  135.0
   D :  2.0
   H :  5.0
   X0 :  0
   Y0 :  0
   Z0 :  0
   A :  0.0
   B :  0.0
   C :  0.0

Configuration : CONN.D.RI
   START.L11 :  1.25
   START.L12 :  1.25
```

```
Configuration : DWELL2
   S :  3.25
   BETA :  108.0
   D :  0.0
   X0 :  0
   Y0 :  0
   Z0 :  0
   H :  5.0
   A :  0.0
   B :  0.0
   C :  135.0

Configuration : RETURN
   S :  3.25
   BETA :  107.0
   D :  -2.0
   X0 :  0
   Y0 :  0
   Z0 :  0
   A :  0.0
   B :  0.0
   C :  243.0
   H :  5.0

Configuration : CONN.RI.D
   START.L21 :  3.25
   START.L22 :  3.25

Configuration : CONN.D.RE
   START.L31 :  3.25
   START.L32 :  3.25

Configuration : CONN.RE.D
   START.L41 :  1.25
   START.L42 :  1.25
```

PDL FILE

```
product camsystem
{
          component cam
          {
                    feature dwell1 "dwell"
                    {
                              l1 = 0.0
                              l2 = 4.0
                              l3 = 1.25
                              a1 = 10.0

                              rotz 350.0
                    }

                    feature return "shm"
                    {
                              l1 = -2.0
                              l2 = 4.0
                              l3 = 3.25
                              a1 = 107.0

                              rotz 243.0
                    }

                    feature dwell2 "dwell"
                    {
                              l1 = 0.0
                              l2 = 4.0
                              l3 = 3.25
                              a1 = 108.0

                              rotz 135.0
                    }

                    feature rise "cycloidal"
                    {
                              l1 = 2.0
                              l2 = 4.0
                              l3 = 1.25
                              a1 = 135.0
                    }

          } $ end component

          component follower
          {
                    feature follower1 "cylinder"
                    {
                              l1 = 1.5
                              d1 = 1.0

                              roty -90
                              movx 1.75
                    }
          } $ end component

} $ end product
```

SPECIFICATIONS

```
item  : CYCLE.TIME;
value : 0.05;

item  : R.B;
value : 1.25;

item  : ANGULAR.VELOCITY;
value : NIL;

item  : RISE.SUGG.MOTION;
value : NIL;

item  : R.FOLLOWER;
value : 0.5;

item  : RISE.TIME;
value : NIL;

item  : RISE.REQ;
value : (NO STEPS IN ACCELERATION STEPS IN JERK);

item  : RISE;
value : 1.5;

item  : DWELL2.SUGG.MOTION;
value : NIL;

item  : DWELL1.SUGG.MOTION;
value : NIL;

item  : DWELL1.TIME;
value : NIL;

item  : DWELL1.REQ;
value : (A SMALL DWELL);

item  : DWELL2.REQ;
value : NIL;

item  : DWELL2.TIME;
value : 0.02;

item  : RETURN.TIME;
value : NIL;

item  : RETURN.REQ;
value : NIL;

item  : RETURN.SUGG.MOTION;
value : SHM;

item  : PRESS.ANGLE;
value : 25;

item  : MOTION.PATTERN;
value : DRD;
```

```
            RESULTS

Configuration : CAM-SYSTEM
   ANGULAR.VELOCITY :  125.66359999999999
   PRESS.ANGLE :  22.79075388899968

Configuration : FOLLOWER
   R.F :  0.5
   D.CYLINDER :  1.0
   H.CYLINDER :  5.0
   X0 :  1.25
   Y0 :  0
   Z0 :  0
   A :  0.0
   B :  0.0
   C :  0.0

Configuration : CAM
   PD1 :  10.0
   PD2 :  144.0
   PRI :  115.0
   PRE :  91.0
   R.P :  2.5
   OFFSET.E :  0.8999999999999999

Configuration : CONN.RE.D
   START.L41 :  1.25
   START.L42 :  1.25

Configuration : CONN.D.RE
   START.L31 :  2.75
   START.L32 :  2.75

Configuration : RETURN
   S :  2.75
   BETA :  91.0
   D :  -1.5
   X0 :  0
   Y0 :  0
   Z0 :  0
   A :  0.0
   B :  0.0
   C :  259.0
   H :  5.0

Configuration : CONN.RI.D
   START.L21 :  2.75
   START.L22 :  2.75

Configuration : DWELL2
   S :  2.75
   BETA :  144.0
   D :  0.0
   X0 :  0
   Y0 :  0
   Z0 :  0
   H :  5.0
   A :  0.0
   B :  0.0
   C :  115.0

Configuration : CONN.D.RI
   START.L11 :  1.25
   START.L12 :  1.25

Configuration : RISE
   S :  1.25
   BETA :  115.0
   D :  1.5
   H :  5.0
   X0 :  0
   Y0 :  0
   Z0 :  0
   A :  0.0
   B :  0.0
   C :  0.0
   RHO :  1.8143702526467154

Configuration : DWELL1
   S :  1.25
   BETA :  10.0
   D :  0.0
   H :  5.0
   X0 :  0
   Y0 :  0
   Z0 :  0
   A :  0.0
   B :  0.0
   C :  350.0
```

SPECIFICATIONS

item : PIPE.DIAMETER;
value : 5;

item : PRESSURE;
value : 6;

item : TEMPERATURE;
value : 60;

item : FLUID;
value : HOT.WATER;

item : WORK.ENVIRONMENT;
value : (USED FOR PIPE WORK SERVICE-CONDITION GENERAL NOT SEVERE);

item : ECONOMIC.CONSIDERATION;
value : ECONOMICAL;

item : VACCUM.SERVICE;
value : NIL;

```
          RESULTS

Configuration : BOLTED.FLANGE
   FLANGE.FACE.TYPE :  FULL-FACED
   B.UC :  5.66
   D.UC :  1.4375
   G.UC :  6.4375
   H.UC :  10.0
   J.UC :  0.9375
   NO.BOLTS :  8.0
   BOLT.DIA :  0.75
   BOLT.CIRCLE.DIA :  8.5
   BOLT.LENGTH :  3.75
   FLANGE.MATERIAL :  FORGED.STEEL.SA181
   E.UC :  3.0E7
   MU :  0.31
   K.UC :  1.7667844522968197
   Z.UC :  1.9427170695246334
   T.UC :  1.7
   U.UC :  4.0
   Y.UC :  3.7
   G0.LC :  0.38874999999999993
   G1.LC :  0.38874999999999993
   G10.LC :  1.0
   H.LC :  0.5
   HBG0 :  0.3370750345632861
   F.LC :  1.0
   F.UC :  0.90892
   V.UC :  0.550103
   T.LC :  0.9375
   X.UC :  0.6984999665249465
   F1 :  4.621953645805625
   F2 :  1.272578096095082
   F3 :  4.209777777777778
   F4 :  1.9427170695246334
   HD.UC :  2188.981022090171
   HD.LC :  0.8975
   G.DIA :  7.83
   B.LC :  1.42
   HG.UC :  10636.219127812958
   HG.LC :  0.8975
   HH.UC :  4189.21507901285
   HT.UC :  2000.234056922679
   HT.LC :  1.42
   M.OP :  14350.94949536826
   M.ATM :  34556.983032166165
   M.BIG :  34556.983032166165
   SIG.HB :  28219.217970594265
   SIG.RD :  7769.6925218841125
   SIG.TG :  10608.374623706612
   F.FO :  38000

 Configuration : GASKET
   GASKET.MATERIAL :  VEGETABLE.FIBRE
   M.LC :  1.75
   Y.LC :  7.6
   G.MIN.WIDTH :  10

(PARAMETER.NOMENCLATURE
          (FLANGE.MATERIAL FLANGE MATERIAL)
```

```
        (F4 STRESS FACTOR F4)
        (M.OP TOTAL MOMENT)
        (HD.UC PRESSURE FORCE ON THE AREA INSIDE THE FLANGE)
        (F.LC RATIO OF SIG.HB)
        (V.UC RATIO OF SIG.HB)
        (X.UC FACTOR IN SIG.HB)
        (Y.UC FACTOR)
        (T.UC FACTOR)
        (U.UC FACTOR)
        (Z.UC FACTOR)
        (K.UC RATIO OF OUTSIDE DIA H AND INSIDE DIA B)
        (MU POISSON RATIO)
        (H.LC HUB LENGTH)
        (M.LC GASKET FACTOR)
        (G1.LC MAXIMUM HUB THICKNESS)
        (G0.LC SHELL THICKNESS OR MINIMUM HUB THICKNESS)
        (G10.LC RATIO OF G1.LC OVER G0.LC)
        (HBG0 RATIO OF H OVER B G0)
        (F.LC RING THICKNESS SHOULD BE THE SAME AS J.UC)
        (F.UC FACTOR IN SIG.RD)
        (HD.LC DISTANCE OF HD.UC)
        (HT.UC PRESSURE FORCE ON THE FLANGE FACE)
        (HT.LC DISTANCE OF HT.UC)
        (HG.UC GASKET REACTION PRESSURE FORCE)
        (HG.LC DISTANCE OF HG.UC)
        (M.ATM MOMENT REQUIRED FOR BOLT LOAD)
        (W.M2 BOLT LOAD REQUIRED TO SEAT THE GASKET)
        (Y.LC GASKET SEATING PRESSURE)
        (G.DIA GASKET MEAN DIAMETER)
        (G.MIN.WIDTH GASKET MINIMUM WIDTH)
        (B.LC 2B IS EFFECTIVE GASKET PRESSURE WIDTH B IS EFFECTIVE
GASKET SEATING WIDTH)
        (F1 STRENGTH FACTOR F1)
        (F2 STRENGTH FACTOR F2)
        (F3 STRENGTH FACTOR F3)
        (SIG.HB LONGITUDINAL NUB STRESS)
        (SIG.RD RADIAL FLANGE STRESS)
        (SIG.TG TANGENTIAL FLANGE STRESS)
        (NO.BOLTS NUMBER OF BOLTS)
        (BOLT.DIA BOLT DIAMETER INCH)
        (BOLT.CIRCLE.DIA BOLT CIRCLE DIAMETER)
        (BOLT.LENGTH BOLT LENGTH INCH)
        (B.UC INSIDE DIAMETER INCH)
        (D.UC FLANGE THICKNESS INCH)
        (G.UC HUB INTERMEDIATE DIAMETER INCH)
        (H.UC OUTSIDE DIAMETER)
        (J.UC RING THICKNESS INCH)
        (KK.UC FLANGE BASE DIAMETER INCH)
        (GAS.MATERIAL GASKET MATERIAL)
        (FLANGE.FACE.TYPE FLANGE FACE TYPE)
        (FLANGE.TYPE FLANGE TYPE)
        (E.UC YOUNG S MODULUS)
        (PIPE.DIAMETER PIPE DIAMETER)
        (F.FO MAXIMUM ALLOWABLE DESIGN STRESS)
        (HH.UC TOTAL PRESSURE FORCE)
        (M.BIG BIGGER ONE OF M.OP AND M.ATM))
```

```
product boltedflange1
{
        component flange1
        {
                feature flange1geomf1 "cylinder"
                {
                        l1 = 0.9375
                        d1 = 10.0

                        feature flange1bore1 "bore"
                        {
                                l1 = 0.9375
                                d1 = 5.66
                        }
                }

                feature flange1geomf2 "cylinder"
                {
                        l1 = 0.5
                        d1 = 6.4375

                        movx 0.9375

                        feature flange1bore2 "bore"
                        {
                                l1 = 0.5
                                d1 = 5.66

                                movx 0.9375
                        }
                }

        } $ end component

        component flange2
        {
                movx -1.5

                feature flange2geom1 "cylinder"
                {
                        l1 = 0.9375
                        d1 = 10.0

                        feature flange2bore1 "bore"
                        {
                                l1 = 0.9375
                                d1 = 5.66
                        }
                }

                feature flange2geomf1 "cylinder"
                {
                        l1 = 0.5
                        d1 = 6.4375

                        movx -0.5
```

```
                    feature flange2bore2 ”bore”
                    {
                                l1 = 0.5
                                d1 = 5.66
                    }

          }

} $ end component

component bolt1
{
          movx -2.1
          movy 4.25

          feature bolt1geomf1 ”cylinder”
          {
                    l1 = 3.75
                    d1 = 0.75
          }
} $ end component

component bolt2
{
          movx -2.1
          movy -4.25

          feature bolt1geomf1 ”cylinder”
          {
                    l1 = 3.75
                    d1 = 0.75
          }
} $ end component

component bolt3
{
          movx -2.1
          movz 4.25

          feature bolt1geomf1 ”cylinder”
          {
                    l1 = 3.75
                    d1 = 0.75
          }
} $ end component

component bolt4
{
          movx -2.1
          movz -4.25

          feature bolt1geomf1 ”cylinder”
          {
                    l1 = 3.75
                    d1 = 0.75
          }
} $ end component
```

```
          component bolt5
          {
                    movx -2.1
                    movy 3.01
                    movz 3.01

                    feature bolt1geomf1 "cylinder"
                    {
                              l1 = 3.75
                              d1 = 0.75
                    }
          } $ end component

          component bolt6
          {
                    movx -2.1
                    movy -3.01
                    movz 3.01

                    feature bolt1geomf1 "cylinder"
                    {
                              l1 = 3.75
                              d1 = 0.75
                    }
          } $ end component

          component bolt7
          {
                    movx -2.1
                    movy 3.01
                    movz -3.01

                    feature bolt1geomf1 "cylinder"
                    {
                              l1 = 3.75
                              d1 = 0.75
                    }
          } $ end component

          component bolt8
          {
                    movx -2.1
                    movy -3.01
                    movz -3.01

                    feature bolt1geomf1 "cylinder"
                    {
                              l1 = 3.75
                              d1 = 0.75
                    }
          } $ end component

} $ end product
```

```
        SPECIFICATIONS

item  : FLUID;
value : STEAM;

item  : PRESSURE;
value : 12;

item  : PIPE.DIAMETER;
value : 4;

item  : TEMPERATURE;
value : 150;

item  : WORK.ENVIRONMENT;
value : (USE PIPE WORK SERVICE-CONDITION GENERAL NOT SEVERE);

item  : ECONOMY.CONSIDERATION;
value : GENERAL;

item  : VACUUM.SERVICE;
value : NIL;
```

```
            RESULTS

Configuration : BOLTED.FLANGE
   FLANGE.FACE.TYPE :  GASKET.WITHIN.BOLT.CIRCLE
   B.UC :  4.57
   D.UC :  1.88
   G.UC :  5.75
   H.UC :  10.0
   J.UC :  1.25
   NO.BOLTS :  8.0
   BOLT.DIA :  0.75
   BOLT.CIRCLE.DIA :  7.88
   BOLT.LENGTH :  4.25
   FLANGE.MATERIAL :  FORGED.STEEL.SA181
   E.UC :  3.0E7
   MU :  0.31
   K.UC :  2.1881838074398248
   Z.UC :  1.5279624243665244
   T.UC :  1.4392131610303625
   U.UC :  2.8910193176013963
   Y.UC :  2.630833671120552
   G0.LC :  0.5899999999999999
   G1.LC :  0.5899999999999999
   G10.LC :  1.0
   H.LC :  0.6299999999999999
   HBG0 :  0.38366876453101706
   F.LC :  1.0
   F.UC :  0.90892
   V.UC :  0.550103
   T.LC :  1.25
   X.UC :  0.5477159132288923
   F1 :  1.5734441632545033
   F2 :  0.9407124061802918
   F3 :  1.6837335495171533
   F4 :  1.5279624243665244
   HD.UC :  2854.11540587665
   HD.LC :  1.0149999999999997
   G.DIA :  7.285
   B.LC :  1.9649999999999999
   HG.UC :  31300.49625366414
   HG.LC :  1.0149999999999997
   HH.UC :  7252.675420099978
   HT.UC :  4398.560014223327
   HT.LC :  1.6549999999999998
   M.OP :  41946.5476579735
   M.ATM :  72826.34295846276
   M.BIG :  72826.34295846276
   SIG.HB :  25074.00093198335
   SIG.RD :  14990.950616579135
   SIG.TG :  3925.934937848444
   F.FO :  38000

Configuration : GASKET
   GASKET.MATERIAL :  COMPRESSED.ASBESTOS
   M.LC :  2.0
   Y.LC :  11.0
   G.MIN.WIDTH :  10
```