INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

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

U·M·I

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

Order Number 1337435

A LISP-based shell for model structuring in system design

Pan, Ning, M.S.

The University of Arizona, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A LISP-BASED SHELL

FOR MODEL STRUCTURING IN SYSTEM DESIGN

by

Ning Pan

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 8 9

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation form or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Ning Pan

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Jerzy W. Rozenblit
Assistant Professor of Electrical
and Computer Engineering Department

5/25/89
Date

ACKNOWLEDGEMENT

I wish to express my gratitude to my advisor and committee chairman, Dr. Jerzy W. Rozenblit, for his valuable guidance, helpful comments and supports during the course of this study, and committee members, Dr. Bernard P. Zeigler and Dr. Francois E. Cellier, for their suggestions and evaluation. I would also wish to acknowledge Raymond G. White for his valuable reviewing of this thesis.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS 6

ABSTRACT 7

CHAPTER 1 MOTIVATION AND OBJECTIVES 8

§1.1 Motivation 8

§1.2 Objectives 10

§1.3 Contribution of the Thesis 11

CHAPTER 2 SYSTEM DESIGN AND STRUCTURE MODELLING 13

§2.1 System Design and Structure Modelling 13

§2.2 System Entity Structure 14

CHAPTER 3 KNOWLEDGE BASE DESIGN 20

§3.1 Knowledge Representation Schemes 20

§3.2 A Qualified Knowledge Representation Scheme -- FRASES 23

§3.2.1 System entity structure in FRASES 23

§3.2.2 Frames in FRASES 24

§3.2.3 Production system and production rule in FRASES 25

§3.2.4 FRASES 27

§3.3 The Shell Rule System 29

§3.3.1 The production rules and the syntax 31

§3.3.2 The rule interpreter 35

CHAPTER 4 INFERENCE ENGINE DESIGN 38

§4.1 Problem State Space 38

§4.2 Inference Engine Paradigm and Implementation 42

§4.2.1 Top-down reasoning 44

§4.2.2 Bottom-up reasoning 51

§4.2.3 Comparisons 54

CHAPTER 5 LISP PROGRAMMING LANGUAGE AND THE SHELL IMPLEMENTATION 58

§5.1 LISP Programming Language 58

§5.2 The Shell System Architecture 59

§5.3 The Shell System Implementation 62

CHAPTER 6 EXAMPLE 68

CHAPTER 7 CONCLUSION 76

LIST OF REFERENCES 78

LIST OF ILLUSTRATIONS

Figure 1.1	Block diagram of the five major steps in completing the thesis	12
Figure 2.2	The concept of hierarchical decomposition of a system	15
Figure 2.2	The system entity structure of personal computer system	19
Figure 3.1	The kinds of knowledge that can go into a knowledge base	21
Figure 3.2	Frame information examples	30
Figure 3.3	The examples of inheritance property and the production rules	33
Figure 4.1	The problem state space	41
Figure 4.2	Example of the problem state space	43
Figure 4.3	An example to present heuristic algorithm	57
Figure 4.4	A special system entity structure	57
Figure 5.1	The shell architecture	60
Figure 5.2	The main menu	63
Figure 6.1	The pruned system entity structure of the personal computer system	75

ABSTRACT

This thesis builds a knowledge-based, computer-aided decision making shell, written in LISP, for assistance in generic engineering system design problems. The theoretical framework presented in the thesis places system design processes in the environment of multifaceted modelling methodology and artificial intelligence techniques. A new reliable and efficient knowledge representation scheme - FRASES is introduced into the knowledge base design. The scheme combines system entity structure and frame and production rule system, and allows us to easily acquire, represent, and infer knowledge and information about the system being designed. In the design of the inference engine, multiple inference algorithms are supported in the shell. They infer a set of desired system configurations with respect to the designer's objectives and requirements. In comparison, top-down reasoning with depth-first offers the most efficient reasoning algorithm when using the FRASES knowledge representation scheme.

CHAPTER 1

MOTIVATION AND OBJECTIVES

§1.1 Motivation

As opposed to system analysis, where we use engineering techniques and knowledge, electronic equipment, and computer software packages to obtain performance measures and input/output relationships of an existing system, system design is the process of synthesis, in which we build a real system based on the constrained input/output relationship and performance objectives required by the customers. Therefore system design is a process of transformation from the customer's objectives and requirements to a set of "blueprints" from which the system will be built, implemented, or deployed.

At this point, the problem is "how is this process to be achieved"? The systems being designed are often of a large scale, and they are subject to a multiplicity of objects. Thus it is easy to conceive that there should be methods for decomposing the systems into subsystems, which are easily comprehensible by the designer. Then the partial subsystems could be generated and integrated using proper aggregation mechanisms. However, sometimes these subsystems may conflict with each other. The attributes of the design should be described in comparative measures and applied by using trade-off techniques.

In order to support this design process, modelling techniques are employed to procure and evaluate a model of the system being designed. Modelling is a creative

act of individuals using the basic problem solving techniques to build conceptual models based on the knowledge and perspective of reality constrained by the requirements and objectives of the modelling project. The process of modelling is a transformation of designer's ideas and requirements to an idealized system. Once the model is created, we can then simulate it to find a satisfactory solution. With this solution the manufacturer can implement the system model which meets all the constraints, objectives, and requirements of the designer.

How to assemble all of the information about a system becomes a crucial problem in the process of system modelling. Artificial intelligence (AI) techniques can be a solution to this problem. A fundamental observation arising from work in AI has been that expertise in a task domain requires substantial knowledge about the domain. The effective representation of domain knowledge is therefore generally considered to be the keystone to the success of AI programs. Domain knowledge typically has many forms, including descriptive definitions of domain-specific terms, descriptions of individual domain objects and their relationships to each other, and criteria for making decisions. Because of this emphasis on representation of domain knowledge, systems that use AI techniques to achieve expertise are often referred to as *knowledge-based systems*, or simply as *knowledge systems*.

Therefore, it is necessary to place system design within a multifaceted, knowledge-based, modelling framework. In 1986, Dr. Rozenblit developed such methodology, which combined artificial intelligence techniques with the system model construction. Based upon this methodology, MODSYN, a PROLOG-written expert system tool for synthesis, was developed [Huang, 1987]. This knowledge-based

environment increased decision making capabilities in both engineering and business environments. Along with the growing complexity of the systems to be designed, there has been increased demand for providing a good knowledge representation scheme and for designing an efficient knowledge inference strategy.

§1.2 Objectives

The objective of this thesis is to create a knowledge-based, computer-aided shell for assistance in generic engineering system. In order to provide a systematic design methodology, the proposed framework will utilize the multifaced modelling concepts [Zeigler, 1984] in system modelling, establish an effective knowledge representation scheme, and apply expert system and artificial intelligence techniques to an efficient inference algorithm. Building this shell will include the work of system modelling, knowledge base design, and inference engine design. The following major steps underlie our methodology:

*) System entity structure can represent the hierarchical decomposition, taxonomic relation, and coupling constraints of a system. It is a basic means of organizing a family of possible configurations of the system being designed.

*) A frame and rule associated system entity structure - FRASES is an efficient and powerful knowledge management scheme. It facilitates the processing of knowledge, including knowledge acquisition, representation, and reasoning.

*) The production rules serve as framework for pruning the alternatives in the system entity structure. The selection rule sets are generated for selection of admissible entities from a specialization, while the synthesis rule sets are

required for identifying the interconnections between individual components, which are to be combined into a desirable model.

*) The inference engine design is based upon two most useful strategies: top-down reasoning and bottom-up reasoning. All satisfactory solutions are obtained with respect to the objectives and requirements of the designer through the use of these techniques.

§1.3 Contribution of the Thesis

This thesis contributes both a conceptual knowledge-based modelling framework and a computer environment implementation for a generic engineering system design. It includes:

*) Modelling a system in hierarchical and taxonomic ways.

*) Design of a flexible knowledge representation scheme.

*) Design of a reliable and efficient inference engine paradigm.

*) Writing model-based source code modules in Common LISP.

*) Implementation of a powerful knowledge-based shell to capture desired system configurations.

As shown in the block diagram of Figure 1.1, there are five major steps that have been accomplished in this thesis.

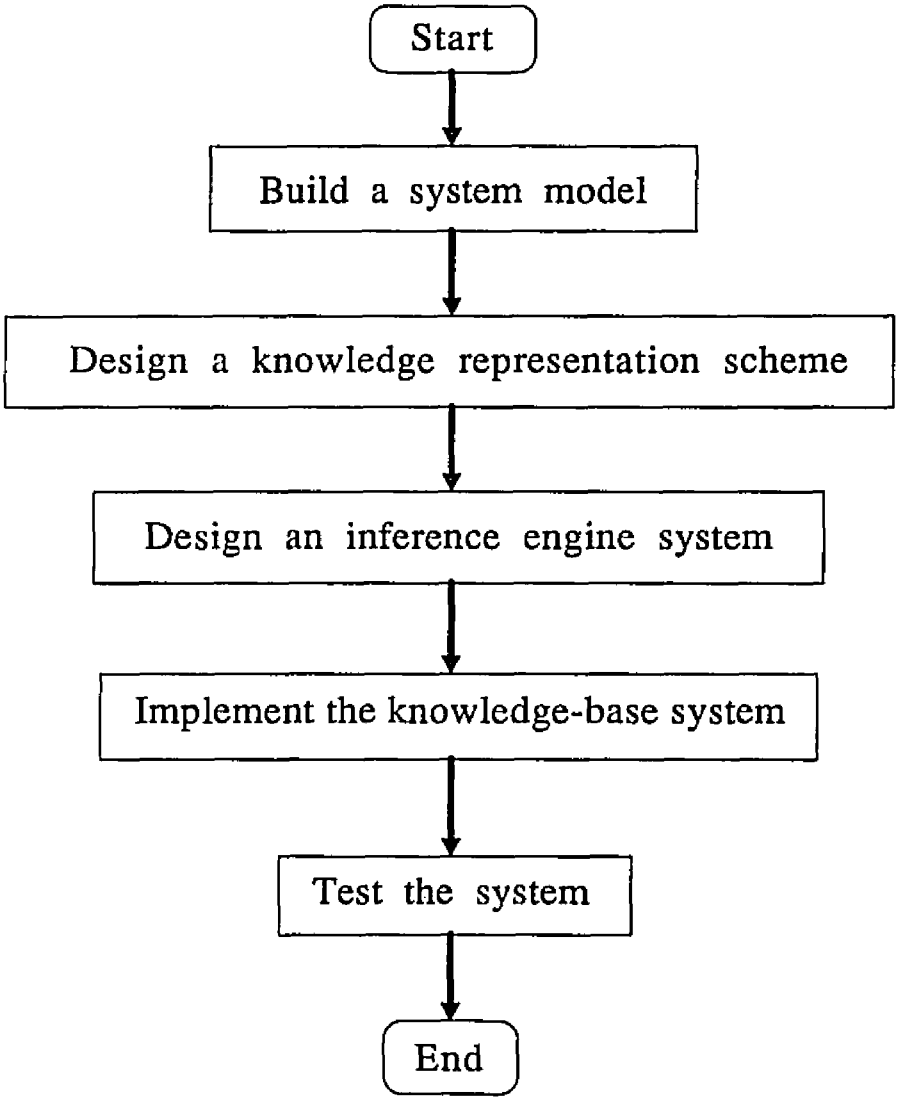


Figure 1.1 Block diagram of the five major steps in completing the thesis

CHAPTER 2

SYSTEM DESIGN AND STRUCTURE MODELLING

The design process is driven by design requirements provided by clients and available technology. The growing complexity of systems being designed has strongly influenced research efforts in constructing computerized support environments for assistance in the design process.

Our primary goal in this chapter is to embed system design within the multifaceted modelling framework and thus provide a systematic design methodology supported by an adequate formal structure. Such an approach is amenable to computerization and direct application of expert system and AI techniques.

§2.1 System Design and Structure Modelling

System design concepts are found in many disciplines. The paradigms of each discipline underlie the methods for design representation and methodology. Most of the design methodologies contain the following major steps:

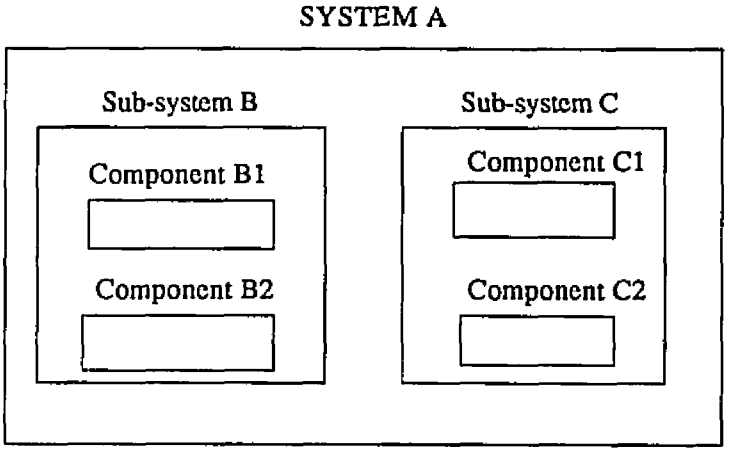
1) State the problem.
2) Identify goals and objectives.
3) Generate alternative solutions.
4) Develop a model.
5) Evaluate the alternatives.
6) Implement the result.

In this thesis, a structure modelling methodology is used in the process of system design. Due to the complexity of many designed systems and the multiplicity of objectives involved in such designs, it is unlikely that a comprehensive model, reflecting all of the requirements and objectives, could be constructed. Even if it were possible to build such a model, its validation, and subsequent verification and simulation would present a paramount degree of complexity. Instead of modelling the whole system, we envision a collection of partial models, each of which reflects a specific objective. In order to simplify a complex system, the specification of design models in a hierarchical manner is concerned. A system is decomposed into several sub-systems, and the successive decomposition to these sub-system is applied until the resulting components do not require further decomposition. Figure 2.1 shows hierarchical decomposition of a system.

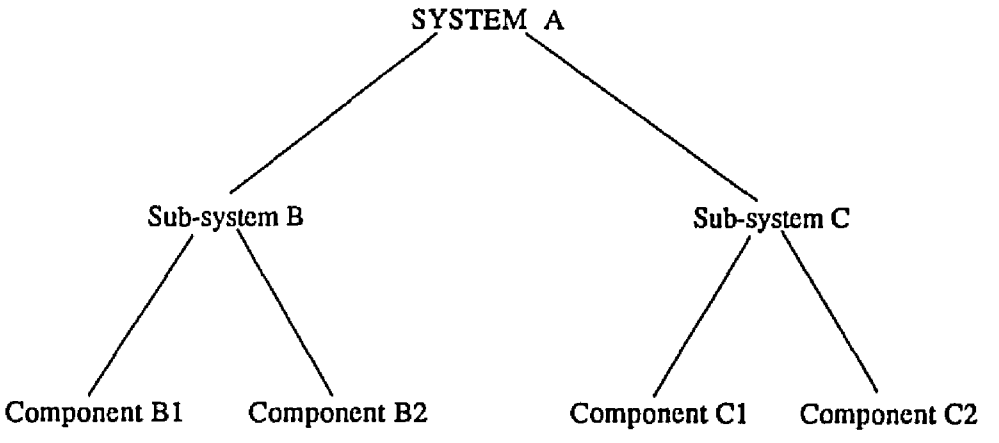
Based upon the above considerations, a multifaceted modeling approach is required that has the ability to decompose a system, to classify components into different variants, and to integrate the sub-components. The fundamental formal concept supporting these activities is an object-oriented modelling methodology -- the system entity structure [Zeigler, 1984]. It enables the user making the model to encompass the boundaries and decompositions conceived for a system.

§2.2 System Entity Structure

The system entity structure is a labeled tree with attached variable types that organizes knowledge in three relationships: *hierarchical decomposition, taxonomy, and coupling constraints*, which have been required by the system modelling in the



(a) The concept of system decomposition



(b) The concept of hierarchical decomposition

Figure 2.1 The concept of hierarchical decomposition of a system

previous section. It is a basic means of modelling and organizing the system structure via entities, aspects, and specializations.

Decomposition knowledge means that the structure has schemes for representing the manner in which an object is decomposed into components. These components can then be further decomposed. Therefore, a system can be expressed as a hierarchy. All the components in a specified level can only "communicate" with those of components in its connecting levels. For example, the consideration at personal computer system level is only its sub-systems, such as memory, CPU, I/O-devices, and O.S. It does not necessary to take care of what kind of monitor to be used. Multiple decomposition is a special type of decomposition. It facilitates flexible representation of multiple entities whose number may vary in the system. The decomposition concept is expressed by the entity-aspect relations of a system entity structure. An entity signifies a conceptual part of the system whose model we aim to build. It is intended to represent a real world object which can either be independently identified or postulated as a component in one or more decompositions of a real world object. An aspect is a mode of decomposition for an entity. Its elements are entities, the components of such a decomposition. Overall, the entities of an aspect represent distinct components of a decomposition, while the aspects of an entity express the different ways to decompose the entity. In this way, a system entity model constructs a system hierarchically.

By taxonomic knowledge, we mean a representation for the kinds of variants that are possible for an object, i.e. how to categorize and subclassify the components of an object, and how to constitute design alternatives. In a system entity structure,

the relationship between entity and specialization represents the taxonomy knowledge. A specialization is a mode of classification for an entity. It facilitates representation of variants for an entity, and allows a designer to collect various design alternatives. Each variant forms an entity, which is a specific type of entity relative to the general entity. Therefore it must inherit properties of the original entity by the linkage of entity-specialization-entities.

The third type of characteristics is synthesis and selection relationships, which are expressed by selection constraints and synthesis constraints. The selection constrains limit choices of variants of objects determined by the taxonomic relations. While the synthesis (coupling) constraints limit the manner in which components identified in decompositions can be coupled together.

By entity, aspect, and specialization, the system entity structure organizes a family of possible design structural configurations. The structural models should satisfy the following axioms [Zeigler, 1984]:

*) *Uniformity:* Any two nodes which have the same labels have identical attached variable types and isomorphic sub-trees.

*) *Strict hierarchy:* No label appears more than once down any path of the tree.

*) *Alternating mode:* Each node has a mode which is either "entity" or "aspect" / "specialization", the mode of a node and the modes of its successors are always opposites. The mode of the root is an entity.

*) *Valid Brothers:* No two brothers have the same label.

*) *Attached Variables:* No two variable types attached to the same item have the same name.

As well, the structural model should also allow the following operations to work on [Rozenblit, 1985]:

*) Naming scheme

*) Generation of distribution / aggregation relations

*) Transformations to taxonomy free form

*) Pruning

*) Attachment of constraints to aspect

Figure 2.2 shows the system entity structure of personal computer systems. In this figure, one vertical bar "|" means decomposition, and double and triple vertical bars ("||" and "|||") represent specialization and multiple decomposition, respectively.

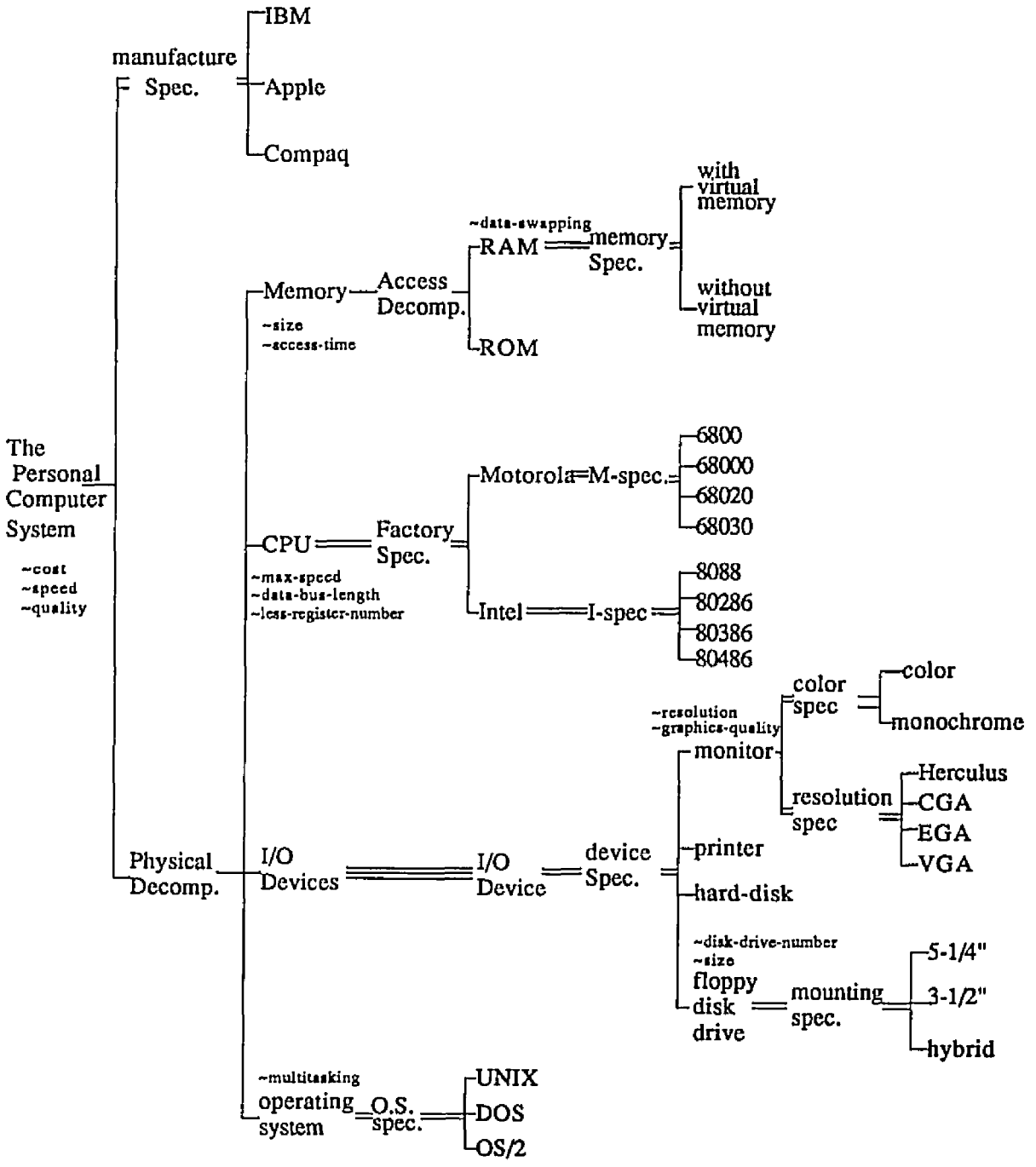


Figure 2.2 The system entity structure of personal computer systems

CHAPTER 3

KNOWLEDGE BASE DESIGN

In order to solve a complex problem through the use of artificial intelligence, we need large amounts of knowledge and some mechanisms for manipulating the knowledge to find the solution to the problem. Figure 3.1 shows what kinds of knowledge should be contained in a knowledge base. Knowledge forms the cornerstone of a knowledge-based system. On the other hand, the knowledge representation scheme, which essentially provides convenient formats for the organization of the knowledge base, determines the efficiency of knowledge acquisition, manipulation, and reasoning. Thus we must be able to abstractly represent knowledge and use the knowledge to support the system's reasoning process. This chapter presents the knowledge representation scheme used in the shell.

§3.1 Knowledge Representation Schemes

Over the past three decades, considerable research has been devoted to develop strategies for knowledge representation. Different schemes such as production rules, frames, structure model, conceptual dependency, semantic networks, AND/OR trees, scripts, predicate logic, and system entity structure, have been defined representing knowledge. Although these representation schemes have some advantages in one or more fields, they have some limitations in the other fields. For example, the predicate logic was appealing because of its very general expressive power and well-defined

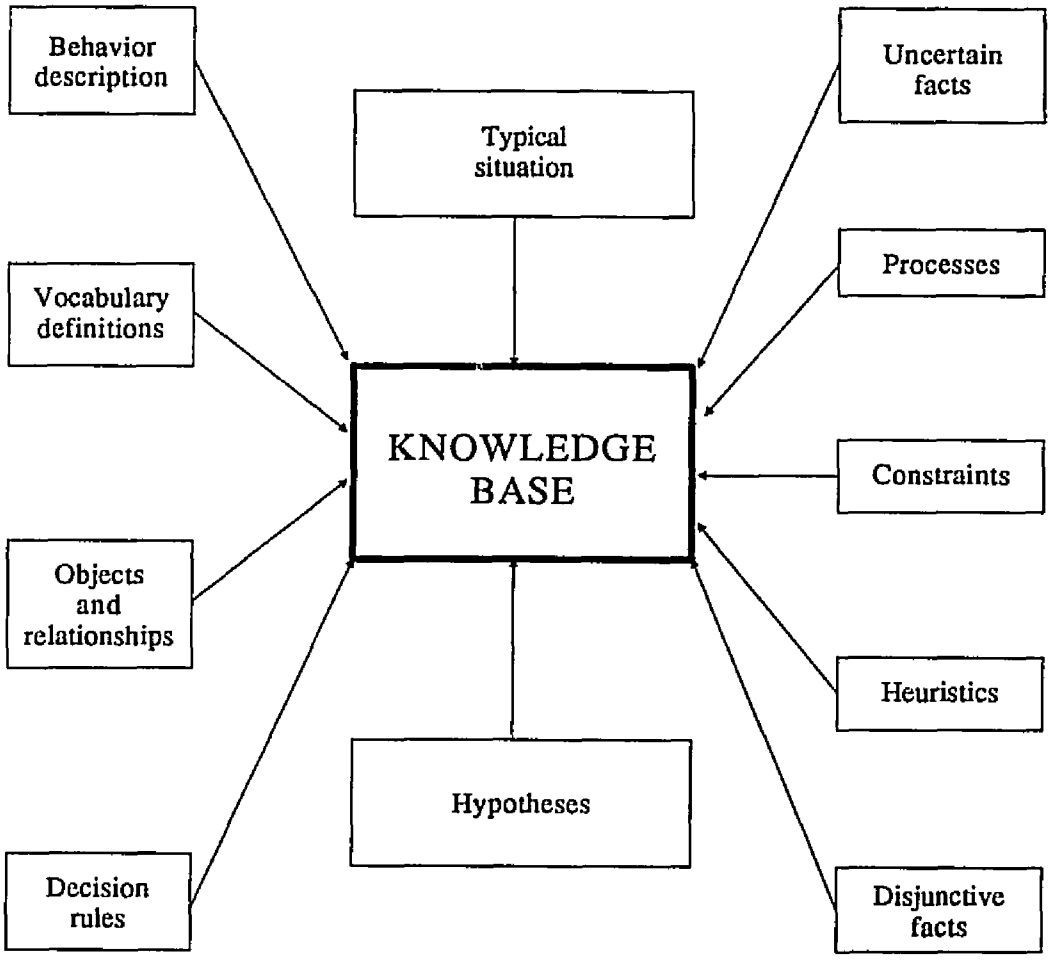


Figure 3.1 The kinds of knowledge that can go into a knowledge base .

semantics. However, the objects in the predicate logic representation are so simple that much of the complex structure found in the real world cannot be described easily.

A good system for the representation of complex structured knowledge in a particular domain should possess the following four properties [Rich, 1983]:

*) Representational Adequacy -- the ability to represent all of the kinds of knowledge that are needed in that domain.

*) Inferential Adequacy -- the ability to manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.

*) Inferential Efficiency -- the ability to incorporate into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions.

*) Acquisition Efficiency -- the ability to acquire new information easily. The simplest case involves direct insertion, by a person, of new knowledge into the database. Ideally, the program itself would be able to control knowledge acquisition.

At the same time, a reasonable knowledge representation scheme must also be flexible enough to express both static and dynamic knowledge representations [Rozenblit *et.al*, 1989], which can be classified as follows:

*) Static Knowledge.

1. structural characteristic of objects.
2. taxonomy / decomposition of objects.

3. constraints and rules for design selection / synthesis.

*) Dynamic Knowledge

1. functional characteristics of objects.
2. procedures for generation of design alternatives.
3. procedures for design verification and evaluation.

In addition, such a scheme should also facilitate knowledge management tasks in a complex design problem. The knowledge reflected by the representation scheme must be transparent to domain experts, knowledge engineers, and system users. The domain experts, knowledge engineers, and system users can communicate their knowledge effectively to the system.

§3.2 A Qualified Knowledge Representation Scheme -- FRASES

Hu [Rozenblit, Hu, *et.al.* 1989] has developed a combination knowledge representation scheme, Frame and Rule Associated System Entity Structure (FRASES), which incorporates the production rules and frames into the system entity structure.

§3.2.1 System entity structure in FRASES

The system entity structure is derived from the multifaceted modelling methodology [Zeigler, 1984], which was presented in detail in the previous chapter. It is a knowledge representation scheme that facilitates expressing the decomposition hierarchy, the taxonomy of the objects it represents, and coupling constraints on the ways in which system components identified in the decomposition hierarchy can be

coupled together. The modelling structure is a basic means of organizing a family of possible design configurations.

§3.2.2 Frames in FRASES

A frame is a collection of facts and data about something or some object. In the system entity structure, each object is associated with a framework within which the information about the object can be interpreted. For this reason, the FRASES is referred to as an object-oriented representation scheme. It provides the knowledge base builder with an easy means of describing the types of domain objects that the system must model. The description of an object type can contain a prototype description of individual objects of that type. The prototype can be used to create a default description of an object when its type becomes known in the model.

A frame consists of a series of *slots* each of which represents a standard property or attribute of the element represented by the frame. A slot gives us a place to systematically store one component of our past experience regarding the class of elements under consideration. Each slot is identified by the name of the corresponding attribute and includes the value, or range of values, that can be associated with the slot. A default value for the slot may also be indicated. For complex domains, slots can be divided into increasingly detailed sub-slots, called *facets*.

Like a system entity structure, a frame system, in which all frames are linked together, not only has the ability to organize facts and information in a hierarchy, but it also has inheritance properties. By hierarchy we mean that an object can be

decomposed into components, which are linked by "apo" (a part of) to the object. By inheritance property we mean that a class of specific objects can inherit the properties of their above abstraction object, which are linked by "ako" (a kind of). For example, in Figure 2.2 CPU is a part of (apo) personal computer system, while DOS is a kind of (ako) operating system.

§3.2.3 Production system and production rule in FRASES

A *production system*, the most commonly used scheme in expert systems, uses rules for knowledge representation. A production system consists of:

*) A set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule, and a right side that describes the action to be performed if the rule is applied.

*) One or more databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem. The information in these databases may be structured in any appropriate way.

*) A control strategy that specifies that order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.

As you see from the above definition, production rules play an important part in the production rule system. The most popular and effective representational form for declarative description of domain-dependent behavior knowledge in the knowledge base system has been pattern/action decision rules, called production rules. The

production rules can also be considered as constraints, which are located in the knowledge base - FRASES.

In a production system, the production inference rules are inference rules that are based upon what is known about the problem in general. For example, it is known that all birds have feathers. If it is known that an animal has no feathers, then it can be inferred that it is not a bird. This simple deduction amounts to a change in the problem state if it is concerned with identifying animal species. That is, what is known about the problem in particular now includes information which narrows the focus of the search and advances search progress beyond the initial state.

Each of the rules consists of a condition part and an action part, and has the following general form:

IF: [antecedent]

.....

[antecedent]

THEN [consequence]

.....

[consequence]

The condition portion of a production rule consists of a series of condition elements that should be satisfied by a value or a set of values for the rule to be applicable. These conditions are described by identifying required global memory patterns: memory element identifiers along with associated attributes and required values.

The action portion of the rule describes the actions to be taken when the rule fires. The actions have general results, such as entering new state descriptions in global memory, modifying existing state descriptions, and performing a user-defined action that is unique to the specific production.

Production rule systems can be implemented for any of the problem-solving approaches. Thus, the "goal-driven" approach may be used, employing the rules to chain backwards from the goal searching for a complete supportive or causal set of rules and data. Or, a "data-driven" approach can be used, employing forward chaining of the rules to search for the goal. These two kinds of strategies will be explained in detail in the next chapter.

§3.2.4 FRASES

A FRASES is a super-class of the system entity structure, frame, and production system knowledge representation schemes. All the axioms and operations for these knowledge representation schemes are valid for the FRASES-based representation. A FRASES representation scheme provides a structured representation of an object or a class of objects. Constructs are available in a FRASES for organizing frames hierarchically and taxonomically.

The following features distinguish FRASES from other representation schemes [Rozenblit, Hu, *et.al.* 1989]:

*) Implicitness of knowledge.

*) Reducing complexity of a problem.

*) Uniformity of the knowledge base.

*) Hierarchical organization of the knowledge base.

*) Flexible refinements of the knowledge base.

*) Verification of the knowledge base.

The advantages of FRASES are considerable: they capture the way experts typically think about much of knowledge, represent an object in a hierarchical way, provide a concise structural representation of useful relations, and support a concise definition by specialization techniques that are easy for most domain experts to use. In addition, pruning algorithms have been developed that exploit the structural characteristics of FRASES to rapidly perform a set of inferences commonly needed in a knowledge system application. This will be explained in detail in the next chapter.

With the FRASES knowledge, each node in the system entity structure is associated with an Entity Information Frame (EIF). Each Entity Information Frame is a frame structure containing the following information:

1) Structural information

The structural information presents the name of the frame by the slot "Name", the mode of the node by the slot "Type", and the relationships of the node with its connecting nodes by the slots "Parents" and "Children". If the node is an entity and it is a sub-component of the above entity, a slot "Apo" (a part of) describes the relationships between these two entities. On the other hand, if the node is a specific entity related to the general entity, their relationship is represented by the slot "Ako" (a kind of).

2) Variable information

EIF: PERSONAL COMPUTER SYSTEM

```
(PERSONAL COMPUTER SYSTEM
  ( Name  ( Value  PERSONAL COMPUTER SYSTEM ))
  ( Type  ( Value  Ent) )
  ( Children  ( Value Function-dec ) )
  ( Variables  ( Types  Cost Speed Quality) )
  ( Cost ( Value high) )
  ( Speed  ( Value  20MHz ) )
  ( Quality ( Value high) )
  ( User-rule  ( R36 ( if  (manufacture-spec = Apple or manufacture-spec = Compaq)
                      then (O.S.-spec != OS/2) ) )
               ( R37 ( if (manufacture-spec = Apple) then (O.S.-spec = UNIX) ) ) )
  ( Rule  ( R36  (syn  (or  (= manufacture-spec  Apple)  (= manufacture-spec Compaq) )
                       (!= O.S.-spec  OS/2) ) ) )
          ( R37 (syn (= manufacture-spec Apple) (= O.S.-spec UNIX) ) ) ) ) )
```

(a) The frame information of personal computer system node

EIF: COLOR-SPEC

```
( COLOR-SPEC
    ( Name  ( Value  Color-spec ) )
    ( Type  ( Value  Spec ) )
    ( Parents  (Value  Monitor-Driver ) )
    ( Children  ( Value  monochrome  color ) )
    ( User-rule  ( R14  ( if  (Graphic-quality = medium-high  and  cost  = medium-high)
                         then  (Select  Color ) )
                   ( R15  ( if  (Graphic-quality = medium  or cost = medium)
                          then ( Select Monochrome) ) ) )
    ( Rule  (R14  (sel  (and (= Graphic-quality medium-high) (= cost  medium-high)) (Color)))
            (R15  (sel  (or  (= Graphic-quality medium ) (= cost medium)) (Monochrome)))))
```

(b) The frame information of color specialization node

Figure 3.2 FRASES knowledge representation examples

and synthesis constraints may be imposed by technical requirements, standards, resource availability, and cost considerations.

§3.3.1 The production rules and the syntax

Selection Rule Set:

Each selection rule stands for a choice of an entity in a specialization. It restricts the way in which a specific entity may replace a general entity in the pruning process. All the selection rules are associated with specialization of an entity.

Rule syntax:

(Rulename IF (f(att1, att2, ...))

THEN (Select sub-entity))

1) The premise part of the rule, f(att1, att2, ...), is a function that may have arithmetic and logical operations. The operations supported here are "not, /, *, +, -, =, >, <, >=, <=, !=, and, or". The precedence of above operations are arranged in decreasing order. The arguments of the function are attached variables.

As has been presented, the system entity structure has inheritance property. By inheritance we mean that all the properties of a specific entity can be inherited from those of the generic entity. However, the properties of an entity cannot inherit to its sub-components. For example in Figure 3.3 (a), E111 and E112 can inherit the values of variables c and d attached on E11, but they cannot inherit the values of variables a and b attached on E, while E221 and E222 can inherit the values of variables a and b, because E221 and

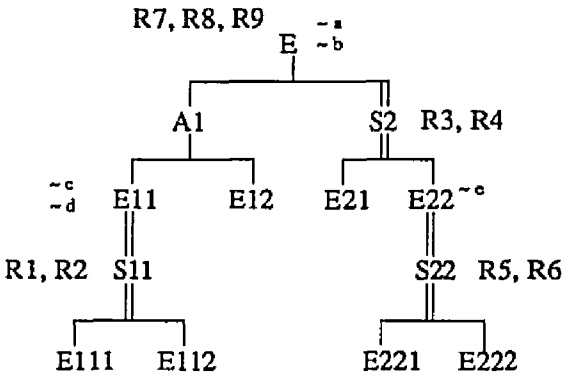
E222 are the specific kinds of E. Based on this concept a inheritance list of E111 is defined as (E111, E11), and a inheritance list of E221 is defined as (E221, E22, E). Figure 3.3 (b) shows the inheritance lists for each entity.

In order to evaluate the condition expression f(att1, att2, ...), the z-inheritance search algorithm is used to get the values [Winston, 1984]. In a frame of a node, each variable may have a default value and a user defined value. For example, we think the default number of floppy disk drive is one. It can also be required as two. So for an entity "floppy disk drive", variable type "disk drive number" is attached. The frame associated with the floppy disk drive has a following slot:

(disk-drive-number (value 2)
(default 1))

The z-inheritance algorithm searches the value of each variable as follows: it searches the user defined value first, if the value cannot be found, it then searches the default value. If the search is not successful, the search is moved to the next entity of its inheritance list. After all the entities in the inheritance list are tried and the search has failed, the system designer is asked to input a variable value. This search is stopped whenever the value of the variable is found, or the variable value is given by the user.

2) The conclusion part of the rule tells you the specific entity to be selected.
3) Figure 3.3 (c) shows examples of the selection rule set. To evaluate the condition of rule R5, b = b1 and e = e1, the z-inheritance algorithm searches the user defined values of b and e on the E22, if not found, searches the



(a) The system entity structure

Inheritance list (I-list)

I-list (E) = (E)
I-list (E11) = (E11)
I-list (E12) = (E12)
I-list (E111) = (E111 E11)
I-list (E112) = (E112 E11)
I-list (E21) = (E21 E)
I-list (E22) = (E22 E)
I-list (E221) = (E221 E22 E)
I-list (E222) = (E222 E22 E)

(b) The inheritance lists

SELECTION RULES:

Rules attached on S11 to select E111 and E112:
R1 if (c = c1 and d = d1) then (select E111)
R2 if (c != c1 or d != d1) then (select E112)

Rules attached on S2 to select E21 and E22:
R3 if (a = a1) then (select E21)
R4 if (a != a1) then (select E22)

Rules attached on S22 to select E221 and E222:
R5 if (b = b1 and e = e1) then (select E221)
R6 if (b != b1 or e != e1) then (select E222)

(c) Selection rule examples

SYNTHESIS RULES:

Rules attached on E for synthesis of the system:
R7 if (S11 = E111) then (S2 = E22)
R8 if (S11 = E111 and S22 = E221) then (ok)
R9 if (S11 = E112) then (S2 != E22)

(d) Synthesis rule examples

Figure 3.3 Inheritance property and the production rules

default values on the entity E22, if still not found, the algorithm moves the next inheritance node E to find the values. The algorithm stops whenever both b and e values are found.

Synthesis Rule:

After a system is decomposed and the configuration of each sub-system is obtained, these sub-systems must be integrated. The synthesis rules describe the constraints of the integration. Each synthesis rule gives the structural constraint, which restricts the way components identified in an entity's decomposition can be joined together. For example in a personal computer system, if with virtual memory is selected for the RAM, the CPU must use 68020 or 80386 as its microprocessor.

Rule syntax:

(Rulename IF (condition) THEN (constraints))

1) The condition part of the rule is dependent on the selected specialization, and is expressed in the following form:

   (spec-1 = entity-1 and/or

    spec-2 = entity-2 ...)

   This expression means that the entity "entity-1" is selected for specialization "spec-1" and/or the entity "entity-2" is selected for specialization "spec-2".

2) The conclusion part of the rule gives the constraints of a certain kind of configuration environment. It has three different kinds of formats:

   *. (ok);

      means that if the rule is triggered, then this configuration is acceptable.

*. (spec-i = entity-i);

means that if the rule is triggered, then you must select entity-i for spec-i.

*. (spec-i != entity-i);

means that if the rule is triggered, then you may not select entity-i for spec-i.

3) Figure 3.3 (d) shows examples of the synthesis rules.

§3.3.2 The rule interpreter

The rule interpreter of the shell includes the rule syntax checking, rule conversion, and the rule attachment. When the user wants to attach rules to a specified node, the rule interpreter checks the type and syntax of the rules, then converts them into the system understood format, and finally attaches both the original rules and their translated rules to the specified node.

As being required, selection rules must be associated with specialization nodes, and synthesis rules must be attached to either aspect nodes or the root of the system entity structure. If the requirement is not satisfied, the user is asked to check the mode of the focus node to make sure the above requirement satisfied. Besides the mode checking, the user's rule must follow the syntax requirement, that is, the rule must be in the list which has five elements: the first element is the rule name; the second is "If"; the third is the condition; the fourth is "then", and the last element is constraints. The formats of the condition and constraints must follow the syntax described in the above section.

The rule interpreter converts user readable rules to an appropriate format that is convenient for LISP to manipulate and is transparent to the user. The format is

(RULE (rulename (rule-type rule-condition rule-action)))

where: "RULE" is the name of a slot in a frame;

"rulename" is the name of the rule;

"rule-type" is the type of the rule, which is either selection rule or synthesis rule;

"rule-condition" is a translated condition of the rule, which is a prefix expression;

"rule-action" is a translated action of the rule.

The rule attachment function stores the transformed rule and the user readable rule into the appropriate attributes of the current frame.

Procedures in LISP use a prefix notation, that is, the procedure name is always specified first, followed by the arguments that the procedure is to manipulate. For example, in LISP a formula is expressed as:

(= 5 (+ 3 2))

Whereas, people usually think about a formula in an infix notation. The above example is expressed as:

3 + 2 = 5

Therefore, the rule interpreter converts a user's infix expression to the machine readable prefix expression. In order to handle precedence of the operators, we use an operator stack and an operands stack. For full explication of these conventions, the reader is referred to (Horowitz, 1984). In rule interpreter recursive calls are employed

in order to handle parentheses.

Finally, the rule attachment adds the user's rule and its translated rule to the frame of the specified node, as shown in Figure 3.2.

CHAPTER 4

INFERENCE ENGINE DESIGN

As has been discussed, FRASES constructs a family of possible system configurations. We need a software system to extract those configurations which conform to the design objectives and constraints. An inference engine is such a software system, which locates knowledge and infers new knowledge from the knowledge base. When we prune the original FRASES with the inference engine, a number of design alternatives may be disregarded as not applicable or not reliable for the given requirements. In this chapter, we shell give a definition of the problem state space, and present both top-down reasoning and bottom-up reasoning as control strategies.

§4.1 Problem State Space

Before a problem can be solved, it must be formally represented. State space representation is one of formal problem descriptions. The *state space,* which defines a problem environment, is a collection of states in which each state corresponds to a unique specification of the domain element. The *initial state* is a beginning condition of the problem. The *goal state* corresponds to acceptable problem solutions. Both of these states are specific states in the state space. The object of a search procedure is to discover a path through a problem state space, starting at an initial state and terminating at a goal state.

We define our problem state space as follows:

A *state* in the state space is a set of configurations of the system being designed.

$$S_k = \{ fc, req \mid fc \quad FC(X), req \quad RQ(D) \}$$

where:

X: the system being designed

D: the system designer

FC(X): the set of all possible configurations of the system being designed

RQ(D): the set of all designer's objectives and requirements

fc: a subset of the system configurations

req: a subset of requirements satisfied by the system configurations

The *initial state* is all possible configurations of the system being designed without objectives and requirements.

$$S_i = \{ FC(X), \phi \}$$

A *goal state* is a subset of system configurations which are satisfied by the all designer's objectives and requirements.

$$S_g = \{ fc, RQ(D) \}$$

As we described, a FRASES constructs a family of system configuration candidates, all of which constitute the initial state. By pruning, some configurations are extracted on the basis of designer's objectives and requirements. These extracted configurations and the corresponding objectives and requirements compose a goal state. When there is no configuration in the goal state, it means that we cannot find any system configuration that satisfies all the current objectives and requirements

RQ(D). When this case happened, the objectives and requirements must be modified.

Having defined the problem state space, we need to find a route between the initial state and a goal state. It is important for us to design an efficient and reliable inference strategy. The most direct and simple application of search is an exhaustive search, in which every possible problem state is tested to see if it is a goal state. In theory, an exhaustive search can be used to solve any finite search problem. Unfortunately, for most real-world problems, the search space is much too large to allow practical exhaustive searches. Therefore a space transformation is necessary for solving this problem.

The space transformation divides a complex problem into a collection of relatively simple components, each with a search space that is more manageable. Based upon the multifaceted objectives of the FRASES representation of the system being designed, the search space is decomposed into several phases. Each one constructs a subset of the search space, and corresponds to one objective of the system designer. Such a decomposition attempts to break the overall problem into smaller sub-task, solves each sub-task independently, and combines the separate pieces to form a solution to the original problem.

From the state space point of view, as shown in Figure 4.1, we see that there are several phases between the initial state and the goal state. Each phase has a sub-goal state, which is the sub-initial state of the next phase. When we go from one state space to a next state, there are more constraints imposed on the next state. Therefore, the system configurations of every state we arrive at are a subset of the previous state.

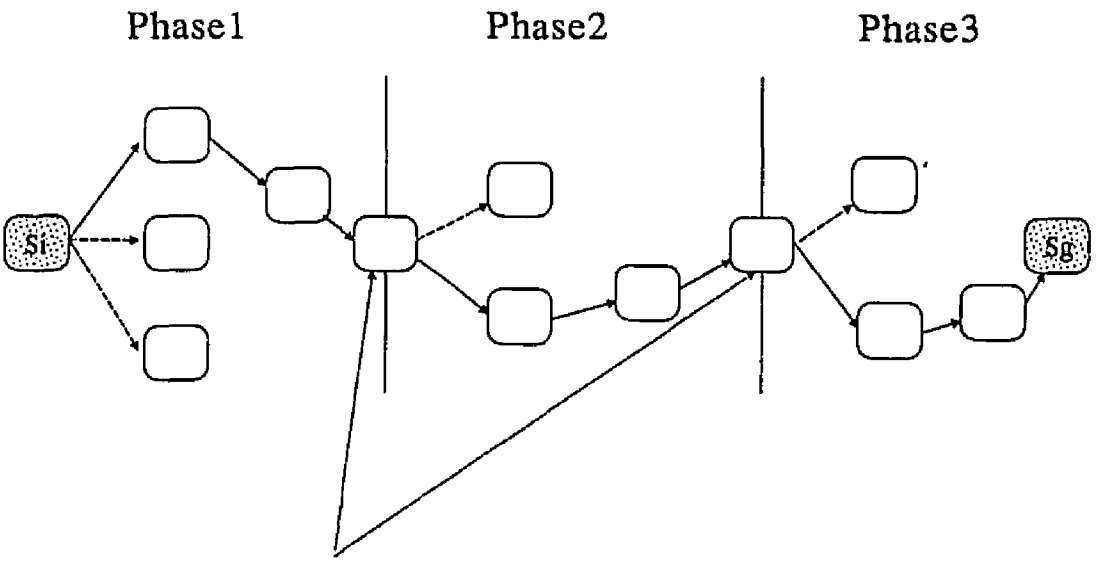
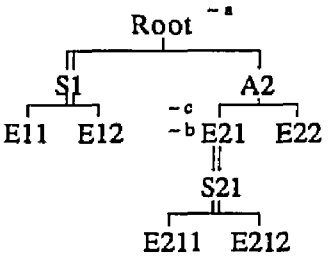


Fig. 4.1 The problem state space

In Figure 4.2 a simple example is given to illustrate the above concepts. Figure 4.2 (a) illustrates a system entity structure and production rules of the system, (b) displays the problem state space, which contains two phases, (c) describes the contents of each state, and (d) shows the initial state and the satisfactory goal state. In this example, the set of all possible configurations of the system FC(X) includes four elements: (1) E11 composed of E211 and E22, (2) E11 composed of E212 and E22, (3) E12 composed of E211 and E22, and (4) E12 composed of E212 and E22. The set of designer's objectives and requirements RQ(D) includes a = a2, b = b1, and c = c1.

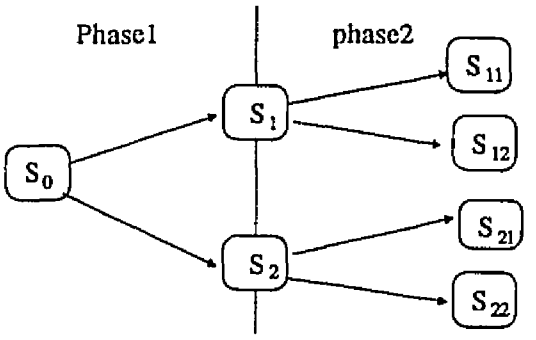
§4.2 Inference Engine Paradigm and Implementation

The engine's inference paradigm is the search strategy that is used to develop required knowledge. Its purpose is to find a path between the initial state and a goal state in the problem state space. So far, many different paradigms are used in an expert system, but most of them are based on one of two fundamental concepts: *forward chaining*, which is a data-driven reasoning process that starts with known conditions and works toward the desired goal, or *backward chaining*, which is a goal-driven reasoning process that starts from the desired goal and works backward toward the requisite condition. In this thesis we have implemented both forward chaining and backward chaining pruning strategies. At the same time, a backtrack search algorithm is employed in the shell. Backtracking on failure is a common control tactic in artificial intelligence and expert system techniques, and can be applied in both forward chaining and backward chaining.



Rules:
R1 if (a = a1) then (select E11)
R2 if (a = a2) then (select E12)
R3 if (b = b1 and c = c1) then (select E211)
R4 if (b = b2 and c = c2) then (select E212)
R 5 if (S1 = E12 and S21 = E212) then (ok)

(a) The system entity structure and production rules



(b) The problem state space

Initial state $S_i = S_0$

Goal state $S_g = S_{21}$

(d) The initial state and goal state

$$S_0 = \left\{ \left\{ \overset{E11}{\underset{E211\ E22,}{\wedge}}\ \overset{E11}{\underset{E212\ E22,}{\wedge}}\ \overset{E12}{\underset{E211\ E22,}{\wedge}}\ \overset{E12}{\underset{E212\ E22}{\wedge}} \right\}, \phi \right\}$$

$$S_1 = \left\{ \left\{ \overset{E11}{\underset{E211\ E22,}{\wedge}}\ \overset{E11}{\underset{E212\ E22,}{\wedge}} \right\}, \{ a = a1 \} \right\}$$

$$S_2 = \left\{ \left\{ \overset{E12}{\underset{E211\ E22,}{\wedge}}\ \overset{E12}{\underset{E212\ E22}{\wedge}} \right\}, \{ a = a2 \} \right\}$$

$$S_{11} = \left\{ \left\{ \overset{E11}{\underset{E211\ E22,}{\wedge}} \right\}, \{ a = a1, b = b1, c = c1 \} \right\}$$

$$S_{12} = \left\{ \left\{ \overset{E11}{\underset{E212\ E22,}{\wedge}} \right\}, \{ a = a1, b = b2, c = c2 \} \right\}$$

$$S_{21} = \left\{ \left\{ \overset{E12}{\underset{E211\ E22,}{\wedge}} \right\}, \{ a = a2, b = b1, c = c1 \} \right\}$$

$$S_{22} = \left\{ \left\{ \overset{E12}{\underset{E212\ E22}{\wedge}} \right\}, \{ a = a2, b = b2, c = c2 \} \right\}$$

(c) The contents of each state

Figure 4.2 The example of a problem state space

With the FRASES knowledge representation, the inference engine prunes the original system entity structure to obtain all desired system configurations by triggering selection rules to get a satisfactory variant or variants, and synthesis rules to check coupling constraints. Based on these concepts, we define *selection pruning* and *synthesis pruning*. By selection pruning we mean that we trigger selection rules on specialization nodes with respect to the designer's objectives and requirements in order to select satisfactory variants. By synthesis pruning we mean that we trigger synthesis rules on aspects nodes and the root of the entity structure to make sure the sub-components can be coupled together. Therefore the resultant pruned model will consist of aspects, specializations, and entities. One or more entities can substitute for the specialization. Through the pruning process, all satisfactory system configurations are produced. The more the constraints that impose on, the less is the number of satisfactory configurations.

§4.2.1 Top-down reasoning

For the FRASES knowledge representation scheme the top-down reasoning process is a froward chaining process. It starts with the initial state, looks at all of the rules to find those rules whose left sides match the designer's requirements and current state conditions, and then uses the right sides of those rules to create the next new state. This process is continued until it arrives at the goal state, i.e. all desired system configurations are generated. All the states between initial state and goal state are called *intermediate* states. In a intermediate state, we evaluate all the rules applied to that state. If there is no matching rule at this state, it means that this is a dead

end state of the path and no sub-system configuration can be generated based on the current objectives and requirements. When this happens, we return to a previous state, cut off this path, and try another match rule. The process of going back to a state that has already been tried is called *backtracking*. When we reach the branch leaf of the system entity structure, a desired sub-system has been created, after which we can continue to prune the other parts of the system entity structure. Based on the matching mechanism, the top-down reasoning is also called a data-driven inference paradigm.

In the implementation of the top-down reasoning inference engine, we divide the reasoning process into two phases: selection pruning phase and synthesis pruning phase. In the selection pruning phase we only check selection rules on specialization nodes of a whole system entity structure, and substitute the specific components for the general component. Finally, when the whole tree has been searched, the configuration candidates of the system being designed come out. Then we go to the next phase. In synthesis pruning we go through those tree branches that were selected in the previous phase. We evaluate every synthesis rule to discard those candidates whose components conflict with each other. In the end we get the desired system configurations which satisfy the designer's objectives and constraints. The reason for separating the selection pruning and synthesis pruning steps in the inference involves the top-down reasoning inference characteristics. Because forward chaining is a top-down inference, it finds the general system components first. The more rules that are checked, the more specific the configurations that are obtained. However, the synthesis rule of a system restricts the coupling constraints of the sub-components

within that system. When we are at the aspect node to check the synthesis rule, we have to know the component configurations of its sub-system. Therefore, separating the two processes is an efficient way to prune the system entity structure.

In the process of pruning a system entity structure, we begin with a specified node of the system entity structure tree, which could be either the root of the tree (i.e. pruning the whole system), or any entity within the tree (i.e. pruning a part of the system). We are faced with two alternative strategies. One strategy is depth-first pruning, with this algorithm all other nodes are ignored until the sub-tree of the first node is pruned. The other strategy is breadth-first pruning, with this algorithm, we get all the component configurations at the current level first, and then go to the next detailed level to find the satisfactory configurations. These two search strategies are implemented in the shell. In the following, we present the depth-first algorithm. The breadth-first algorithm is the same as depth-first except that it adds the children of any node to the back of the queue.

Beginning of the top-down reasoning depth-first algorithm:

```
Top-down-Reasoning-Depth-First-Algorithm (pruning-root)
Begin
; Pruning phase I ---- selection pruning
Form a one-element queue consisting of the pruning-root node
While the queue is not empty, do
   Begin
        Remove the first node of the queue
```

```
        Set working node = the removed node
        If the working node is specialization node {
           Evaluate the selection rules attached to this node with forward chaining algorithm
           If no rule is triggered {
                Delete this specialization node from its inheritance selected attribute list
           }
           Else {
                Put the selected entities on its select attribute list
                Add the selected entities to the front of the queue
           }
        }
        Else {
           Add the children of the node to the front of the queue
        }
   End

; Pruning phase II ---- synthesis pruning
Form a one-element queue consisting of the pruning-root node
While the queue is not empty, do
   Begin
        Remove the first node of the queue
        Set working node = removed node
        If the working node is an aspect node or
                        it is the root of the system entity structure {
             evaluate the synthesis rules attached to the working node with forward
             chaining algorithm
             add its children to the front of the queue
        }
```

```
        Else if the working node is a specialization node {
                Add its selected attribute list to the front of the queue
        }
        Else {
                Add its children to the front of the queue
        }
    End
End
```

End of the top-down reasoning depth-first algorithm.

Here the system entity structure shown in Figure 4.2 (a) is used to illustrate the top-down reasoning depth first algorithm. First the requirements are defined as:

a = a2, b = b1, and c = c1.

Before the pruning, the selected attribute lists of each specialization frame are null.

Beginning of example trace:

```
Top-down-Reasoning-Depth-First-Algorithm (Root)
; Pruning phase I -- selection pruning
; In this phase we prune the system entity structure tree to get the satisfactory configuration
candidates.
        Queue = { Root }.
        Working node = Root, Queue = { }.
        "Root" is an entity node, Queue = { S1, A2 }.
```

Working node = S1, Queue = { A2 }.

"S1" is a specialization node, the rule R1 and R2 are evaluated.

The rule R2 is triggered.

Put the selected entity E12 of rule R2 to the selected attribute list of S1.

Add E12 to the front of the queue, Queue = { E12 A2 }.

Working node = E12, Queue = { A2 }.

"E12" is an entity node, and no child appended to it, Queue = { A2 }.

Working node = A2, Queue = { }.

"A2" is an aspect node, add its children to the front of the queue.

Queue = { E21, E22 }.

Working node = E21, Queue = { E22 }.

"E21" is an entity node, add its children to the front of the queue,

Queue = { S21, E22 }.

Working node = S21, Queue = { E22 }.

"S21" is a specialization node, the rule R3 and R4 are evaluated.

Rule R3 is triggered.

Put the selected entity E211 of rule R3 to the selected attribute list of S21.

Add E211 to the front of the queue, Queue = { E211, E22 }.

Working node = E211, Queue = { E22 }.

"E211" is an entity, no child appended to it, Queue = { E22 }.

Working node = E22, Queue = { }

"E22" is an entity, node child appended to it, Queue = { }.

; End of selection pruning

; At the end of selection pruning, a satisfactory configuration candidate is achieved, that is E12 is composed of E211 and E22.

; Pruning phase II -- synthesis pruning

; In this phase, every candidate obtained from the selection pruning is checked to make sure it meets the synthesis constraints.

Queue = { Root }.

Working node = Root, Queue = { }.

"Root" is the root of the system entity structure.

Evaluate the synthesis rules R5 and R6, which are associated with the node.

Rule R5 is triggered, the selected configuration is accepted.

Add the children of "Root" to the front of the queue, Queue = { S1, A2 }.

Working node = S1, Queue = { A2 }.

"S2" is a specialization node,

add its selected attribute list to the front of the queue, Queue = { E12, A2 }.

Working node = E12, Queue = { A2 }.

"E12" is an entity node, and no child appended to it, Queue = { A2 }.

Working node = A2, Queue = { }.

"A2" is an aspect node, no synthesis rule attached to it.

Add the children of A2 to the front of the queue, Queue = { E12, E22 }.

Working node = E21, Queue = { E22 }.

"E21" is an entity, add its children to the front of the queue,

Queue = { S21, E22 }.

Working node = S21, Queue = { E22 }.

"S21" is a specialization node,

add its selected attribute list to the front of the queue, Queue = { E211, E22 }

Working node = E211, Queue = { E22 }

"E211" is an entity node, no child appended to it, Queue = { E22 }.

Working node = E22, Queue = { }.

"E22" is an entity node, no children appended to it, Queue = { }.

; End of synthesis pruning

; At the end of synthesis pruning, the satisfactory configuration is obtained, that is, E12 is composed of E211 and E22.

End of example trace.

§4.2.2 Bottom-up reasoning

Besides the top-down reasoning inference paradigm, we also provide the bottom-up reasoning algorithm in the inference engine. From the problem state space point of view, the bottom-up reasoning process is a sort of backward chaining process. Backward chaining begins with a goal state, generates next level states by finding all the rules whose right side match the current state, then goes to the next level states. This process is continued until the initial state is found. This method of chaining backward from the goal state is often called goal-directed reasoning.

From the system hierarchical decomposition of the system entity structure point of view, a bottom-up reasoning process starts with the most detailed components and ends at the system level. The configuration of any level cannot be constructed unless all of its sub-components are obtained.

From the FRASES knowledge representation point of view, our work begins with the leaf of the entity structure tree and works upward. After we get the desired configurations of all the sub-components, we will look for those rules having consequence parts that match the current sub-components configurations, and will evaluate the premises of those rules according to the designer's requirements and objectives to get the configuration of the current level system.

The bottom-up reasoning algorithm is implemented using a recursive call, which is determined by bottom-up reasoning characteristics. When we meet a specialization node and get the desired configurations of its children, the selection rules attached to the specialization node are triggered in reverse. On the other hand, when we meet an aspect node, the synthesis rules attached to the aspect are triggered.

Thus the selection pruning and synthesis pruning steps have been combined, which is different than in the top-down reasoning algorithm. The following shows the bottom-up reasoning algorithm.

Beginning of the bottom-up reasoning algorithm:

```
Bottom-up-Reasoning-Algorithm (pruning-root)
Begin
    Get aspects and specializations of the pruning-root
    If the pruning-root has children {AS1, AS2, ...., ASk} {
        For each of the aspect or specialization node
                A-or-S {AS1, AS2, ...., ASk}
        Begin
            Get the children entities of the A-or-S, {E1, E2, ...., En}
            For each of these entities Ei {E1, E2, ...., En}
            Begin
                Call Bottom-up-Reasoning-Algorithm (Ei)
            End
            If the A-or-S is an aspect node {
                Check the synthesis rules associated with the aspect with backward chaining
            }
            If the A-or-S is a specialization node {
                Check the selection rules associated with the specialization with backward
                chaining
            }
        End
```

```
    }
    if the pruning-root is the root of the system entity structure
        evaluate the synthesis rule attached on the pruning-root
End
```

End of the bottom-up reasoning algorithm.

Here the system entity structure shown in Figure 4.2 (a) is used to illustrate the bottom-up reasoning algorithm. Before the pruning the selected attribute lists of each specialization frame are null.

Beginning of the example trace:

```
Bottom-up-Reasoning-Algorithm (Root)
        Get aspects and specializations of the "Root", Queue = { S1, A2 }.
        Working node = S1, Queue = { A2 }.
        Get the children entities of S1, { E11, E12 }.
        Recursive call Bottom-up-Reasoning-Algorithm(E11) to get the configuration of E11.
        Recursive call Bottom-up-Reasoning-Algorithm(E12) to get the configuration of E12.
        "S1" is a specialization node,
        evaluate the selection rules R1 and R2 attached to "S1".
        The user is asked to input the values of variable a and b, a = 1, b = 3.
        Rule R2 is triggered.
        Put the selected entity E12 of rule R2 to the selected attribute list of S1.
        Working node = A2, Queue = { }.
```

Get the children entities of A2, { E21, E22 }.

Recursive call Bottom-up-Reasoning-Algorithm(E21) to get the configuration of E21 = E211.

Recursive call Bottom-up-Reasoning-Algorithm(E22) to get the configuration of E22.

"A2" is an aspect node, no synthesis rules attached to "A2".

Check the synthesis rules attached to "Root".

; At the end of bottom-up reasoning, the satisfactory configuration is obtained, that is E12 is composed of E211 and E22. Here the requirements are a = a2, b = b1, and c = c1.

End of the example trace.

---

§4.2.3 Comparisons

As presented in the previous two sections, the top-down reasoning of depth-first pruning and breadth-first pruning, and the bottom-up reasoning have been implemented in the shell. The selection of these inference paradigms has strongly influenced the efficiency of the reasoning. Along with the complexity of the system being designed, the cost and efficiency are extremely important.

Both of the forward chaining and the backward chaining algorithms have their own advantages. Generally speaking, when we solve a generic problem, if the initial state is unique and easily identified, it is better to reason forward toward the goal state; however if we know much more about the goal state then the initial state, backward chaining may be preferable. The branch factor is another criterion to decide the reasoning direction [Rich, 1988].

For the specific knowledge representation scheme FRASES employed in the shell system, top-down reasoning is superior to bottom-up reasoning for the following

two reasons: first, with the system entity structure, the initial state is very easy to identify. When the system entity structure of a system being designed has been built, all the possible configurations of the system can be obtained from the system entity structure. Those configurations constitute the initial state.

Secondly, and more importantly, when we prune the entity structure top-down reasoning, if some entities cannot be used to construct the system, then those entities and their sub-structures are ignored. For example in Figure 4.2, when selecting an entity for the specialization S1, only the rules associated with S1 are evaluated with respect to the objectives and requirements. If the entity E12 is selected, E11 and its sub-structure are ignored. However, using bottom-up reasoning, we don't know which configurations are chosen until the configurations of its sub-structure are generated, that is, at the time of selection of E11 and E12 for the specialization S1, the configurations of both E11 and E12 has been achieved. This results in a waste of efforts and memory space.

Further the depth-first pruning and the breadth-first pruning have the same efficiency, because of the definition of pruning, by which we have to find all the satisfactory system configurations. If we put heuristic concepts, such as the greedy algorithm, in the pruning process of the shell, the depth-first pruning algorithm has more advantages than breadth-first. This is because the pruning process of the depth-first algorithm will stop, when a desired configuration is obtained, whereas in the breadth-first algorithm, no satisfactory configuration is gained until the whole system entity structure is pruned. Figure 4.3 illustrates the special case. In this example, we define a = a1 and b = b1. Therefore rules R1 and R2 are triggered. The satisfactory

configurations are: E1 composed of E11 and E12, and E2 composed of E21 and E22. If a heuristic concept is used, the pruning is stopped as long as a satisfactory configuration is found. With the top-down reasoning depth-first algorithm, when the pruning gets a satisfactory configuration, E1 composed of E11 and E12, the pruning is stopped, and the configuration of E2 is ignored, whereas with the top-down reasoning breadth-first algorithm, both configurations of E1 and E2 are obtained at the same time.

Now let's think about a special case. For each branch of the system entity structure tree, (i.e. every path from root to leaf), there is only one specialization node, and this specialization node must be at the leaf of the branch, see Figure 4.4. In this case, all three pruning algorithms have the same efficiency, because every entity has to be pruned.

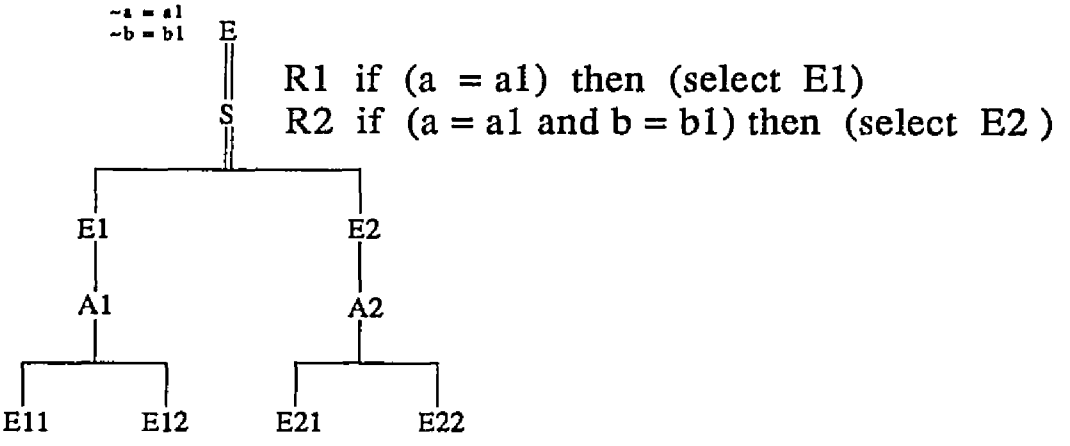


Figure 4.3 An example to present the heuristic algorithm

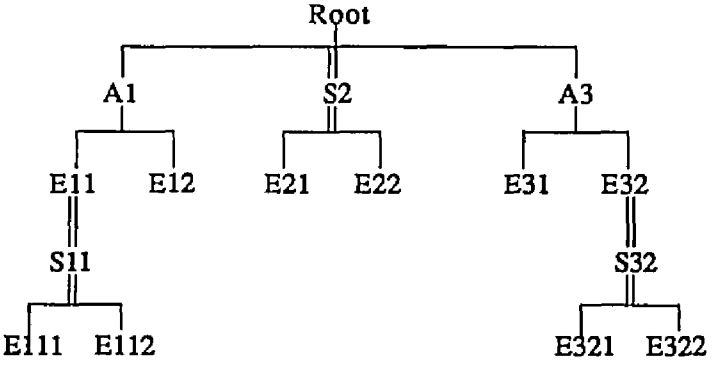


Figure 4.4 A special case of the system entity structure

CHAPTER 5

LISP PROGRAMMING LANGUAGE AND THE SHELL IMPLEMENTATION

In earlier chapters we made a systematic exposition of system modeling, knowledge base design, and inference engine design. These principles may only provide part of the solution to knowledge-based system design. They must be expressed using a computer programming language, which gives rise to a requirement for languages that are as convenient as possible for this purpose.

So far there are three general families of languages used for artificial intelligence programming: (1) functional application languages such as LISP, (2) logic programming languages such as PROLOG, and (3) object-oriented languages such as SMALLTALK and ACTOR. This knowledge-based shell is implemented in Common LISP on VAX11/780.

§5.1 LISP Programming Language

The LISP language was first conceived by John McCarthy in the late 1950's. Since then it has grown to be the most commonly used language for artificial intelligence and expert system development. It is because of the following three most important features that LISP plays its significant role in the AI programming.

(1) LISP (an acronym for *LIS*t *P*rocessing) is a *list processing* language. Historically, list processing was the conceptual core of LISP, and even in modern usage list-

processing activities are very important.

(2) LISP is a *symbolic manipulation* programming language. It is more suitable for carrying out symbolic processing or flexible control than other numerical computation languages, such as FORTRAN, PASCAL, etc. In symbolic processing the basic elements being manipulated are symbols that represent arbitrary objects from the domain of interest.

(3) LISP is a *functional* programming language in which simple functions are defined and then combined to compose more complex functions. Every statement in the language is a description of a function.

§5.2 The Shell System Architecture

The shell system is a decision making system, which is to say that it can select satisfactory system configurations from alternatives with respect to designer's objectives and requirements. An architectural inventory of the shell would include the following four basic elements as shown in Figure 5.1.

1) **User**

The user of the shell can be operating in any of several modes:

*) Designer: User applies his expertise and objectives to build a system model, and uses this model to simulate and verify a real system of the model representing.

*) Tester: User attempts to verify the validity of the system's behavior.

*) Tutor: User provides additional knowledge to the system or modifies knowledge already present in the system.

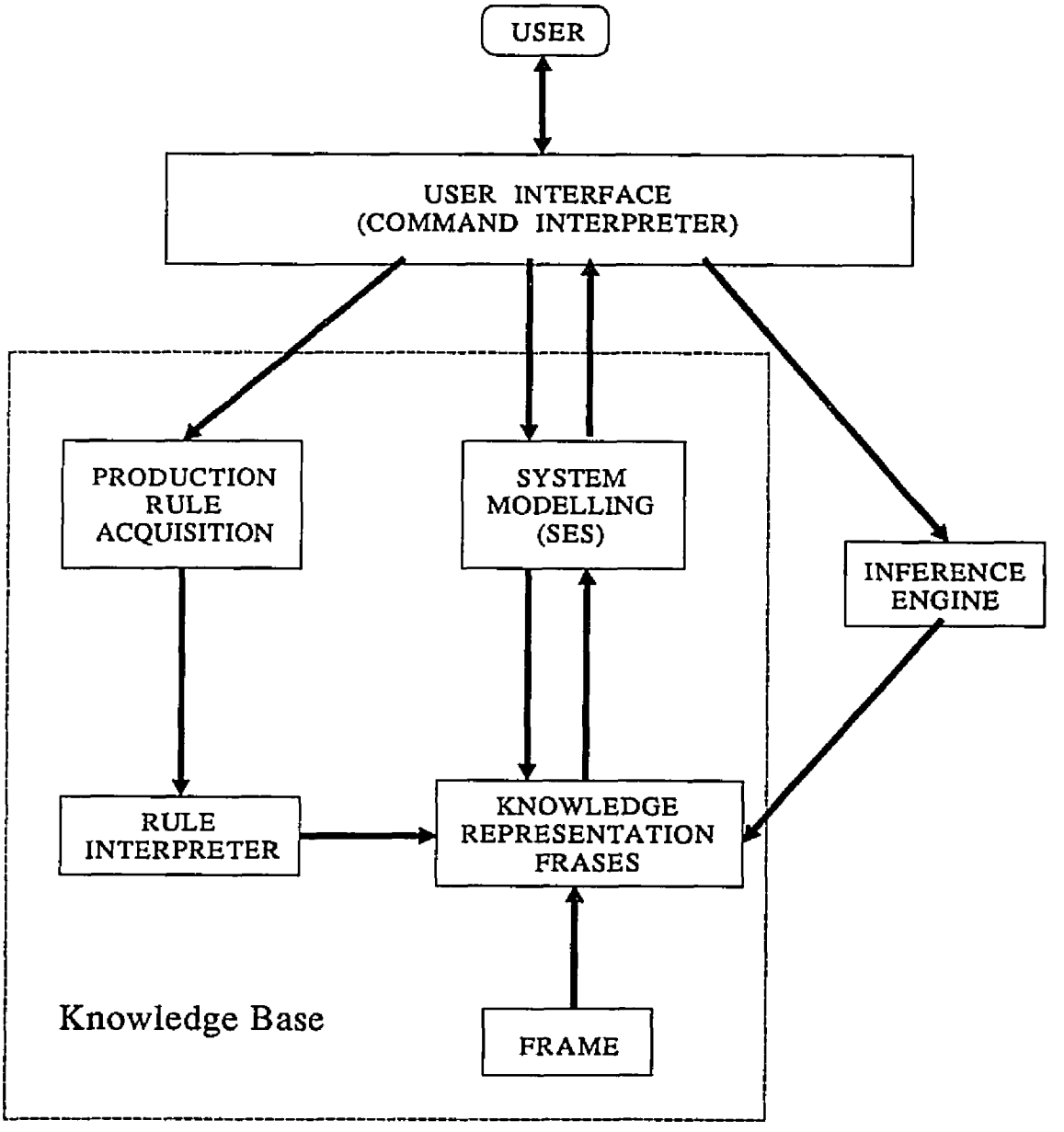


Figure 5.1 The shell system architecture

*) Student: User seeks to rapidly develop personal expertise related to the subject domain by extracting organized, distilled knowledge from the system.

2) **User interface facility**

The user interface facility accepts information from the user and translates it into a form acceptable to the remainder of the shell system. As well, it accepts information from the shell system and converts it to a form that can be understood by the user. The user interface facility for the shell is designed to recognize the mode in which the user is operating, the level of the user's expertise, and the nature of each transaction.

3) **Knowledge base**

The knowledge base is a storehouse for the knowledge primitives available to the shell. The knowledge stored in the base establishes the system's ability to act as an expert. The internal structure of the knowledge base consists of a system modelling module, a frame module, a rule interpreter module, and the FRASES composing module. The system modelling module builds a family of hierarchical structures for a system. The frame module organizes information about the structure, while the rule interpreter module converts the user's rules and constraints to the formats understood by the shell. These three sub-components are integrated by the FRASES module. This knowledge representation scheme using FRASES greatly affects the design of the inference engine.

4) **Inference engine**

Generally speaking, the inference engine is a problem solving algorithm, which is able to infer new knowledge from existing knowledge. It finds the satisfactory

system configuration with respect to the designer's objectives and requirements. The inference engine contains an interpreter that decides how to apply the rules to infer new knowledge and a scheduler that decides the order in which the rules should be applied. In fact, these depend on the type of the inference paradigm.

§5.3 The Shell System Implementation

The shell system is written in Common LISP, an advanced dialect of the LISP language, and run on a VAX11/780. It is implemented using six lisp files: frame.lsp, main.lsp, ses.lsp, rule.lsp, prune.lsp, and mis.lsp.

frame.lsp:

The purpose of this module is to implement data structures to support the shell. When building a system entity structure, each node in the structure is assigned a frame in order to store information about the node. Generally, a frame is a nested association property list. It explicitly establishes the attributes associated with the node. All the functions provided in the module are frame-handling procedures, such as get a specified value, put a value, remove an attribute, get inheritance value, etc.

main.lsp:

This module is an interactive menu driven command handler. The menu is shown in Figure 5.2. It is invoked by a function call

(main-loop)

from Lisp. It initializes the shell system, waits for a command from the user, and then processes the command and passes arguments to a corresponding function. When the system gets the command "exit", it leaves the shell and goes back to Lisp

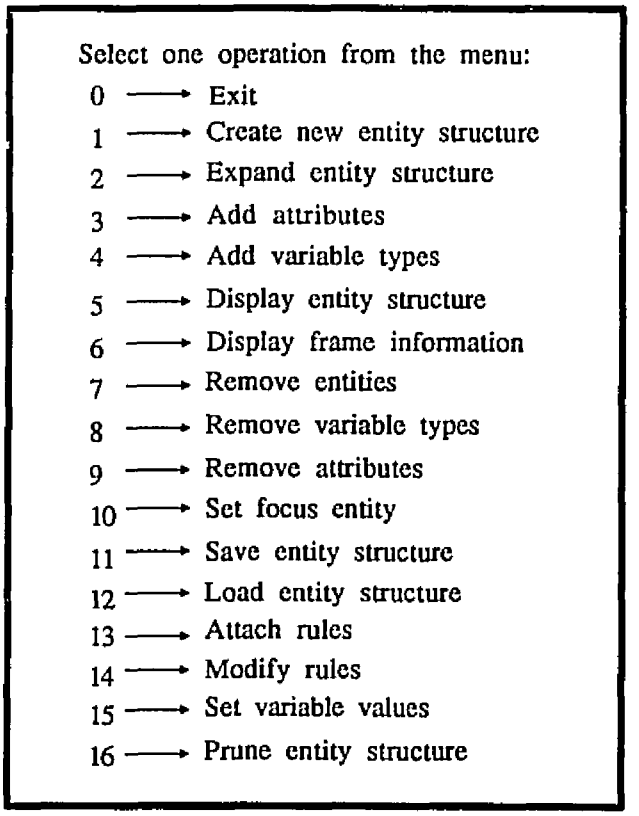


Figure 5.2 The main menu

environment. The commands provided by the shell are the following:

0) Exit

Allows the user to exit the shell system.

1) Create new entity structure

Lets the user start to create a new system entity structure. Before creating a new entity structure, any previously created structure must be saved, otherwise it will be lost.

2) Expand entity structure

Lets the user build an entity structure. It expands the nodes to the current node. In order to meet the alternating mode axiom, this function provides one or more of the specified mode types which are suitable to be attached to the current node. For example, if the current node is in "entity" mode, only node types of specialization, aspect, and multiple-decomposition can be attached to the node. The format is:

((spec s1 s2 ...) (asp a1 a2 ...) (multi-dec m1)).

Otherwise, only an "entity" node can be attached to the current node. At this moment, the format should be:

((ent e1 e2 ...))

When you are expanding an entity structure, some axioms are checked, such as uniformity, strict hierarchy, alternating mode, and valid brothers.

3) Add attributes

Lets the user define the attributes of the current node. Because every node is associated with a frame, the user has to give the attribute name (a slot in the

frame), type (a facet in the slot), and values (values for the facet). You can put one or more attributes to the node at one time. The format is:

```
((slot1 facet1 value-lst1)
 (slot2 facet2 value-lst2)
 ....                      )
```

4) Add variable types

Lets the user define variable types for the current working node. The axiom of valid variables is checked in this command.

5) Display system entity structure

This command displays the current system entity structure. In addition, it indicates the type and level number of each node.

6) Display frame information

This command displays the frame information of the current node.

7) Remove an entity

Lets the user remove the current node and its sub-structure from the system entity structure.

8) Remove variable types

Lets the user remove variable types attached to the current node. The variable types to be removed must be in a list.

9) Remove attributes

Lets the user remove attribute from the attached frame.

10) Save structure

Allows the user to save the current entity structure to an output file. The default file name is the entity structure name with a user specified extension.

11) Load structure

This command reads an entity structure from a user specified file. When the entity structure is read into the shell, the focus node is set to the root of the entity.

12) Set focus

Allows the user to change the current focus node.

13) Attach rules

Lets the user attach rules to the focus node. When the type of the focus node is "specialization", only selection rules can be attached to it. When the type of the focus node is "aspect", or the focus node is the root of the structure, only synthesis rules can be attached. When finished with rule attachments, "End" is entered to return to the shell system.

14) Modify rules

Lets the user modify rules attached to the current focus node. It deletes all the original rules and attaches new rules to the focus node.

15) Set variable values

Before using the top-down reasoning pruning algorithm, the user has to specify the performance measures associated with the behavior aspects of design objectives. This command lets the user assign the objectives and requirements to the corresponding variables. The user has to specify a node name, a variable type, and its value using the following format:

((ent1 var1 val1) (ent2 var2 val2) ...)

16) Prune structure

This command prunes the system entity structure to get a set of satisfactory system configurations. The pruning root is given by the user. There are three pruning strategies for the user to choose from: top-down reasoning using the depth first algorithm, top-down reasoning with the breadth first algorithm, and bottom-up reasoning. After pruning, the shell system will display the pruned system entity structure of the system being designed, which organizes all of the satisfactory system configurations.

rule.lsp

This module is a rule translator. It contains three functions: rule syntax checking, rule interpreter, and rule attachment. After we translate a rule, we attach the rule name, type, condition, and consequence to the associated frame.

prune.lsp

This module uses rule constraints to prune the system entity structure. There are three kinds of pruning algorithms provided: depth first top-down reasoning, breadth first top-down reasoning, and bottom-up reasoning. The implementation of these algorithms was explained in the previous chapter.

ses.lsp

This module allows the user to build a system entity structure. It processes and updates the frame information and entity structure.

mis.lsp

This module provides miscellaneous functions supporting the shell system.

CHAPTER 6

EXAMPLE

In the previous chapters, the theoretical framework and computer implementation are described in detail. Here, we present a simple example, the synthesis of a personal computer system model structure, to illustrating the previous ideas.

The system entity structure of personal computer systems was shown in Figure 2.2. It presented a family of possible structures for a model of personal computer systems. Each object of the system has attached variable types. In this shell, both symbolic and numeric values can be assigned to a variable. For example, some variable types, such as multitasking, data-swapping, less-register-number, virtual-memory, are defined as binary variables. Their values could be either true or false. Some variable types are non-unit variables, such as reliability, graphic-quality, whose values are defined as low, medium, or high. Some other variables are defined as numeric variables, such as disk-drive-number, access-time, *etc.*

To build such a system entity structure in the shell, Command1 through Command9 are used. The structure can also be loaded by Command11 (Load). After building the system entity structure, Command13 (Attach rules) is used to attach rules to the system entity structure. Following are the production rules for the personal computer system:

SELECTION RULE SET:

Rules attached on manufacture specialization for selection of IBM, Apple, and Compaq personal computer:

R1 if (quality = high and cost = high)
then (select IBM)

R2 if (graphic-quality = high and cost = high)
then (select Apple)

R3 if (cost = medium)
then (select Compaq)

Rules attached on memory specialization for selection of with-virtual-memory or without-virtual-memory:

R4 if (data-swapping = true)
then (select with-virtual-memory)

R5 if (data-swapping = false)
then (select without-virtual-memory)

Rules attached on factory specialization for selection of Motorola, or Intel CPU:

R6 if (less-register-number = false)
then (select Motorola)

R7 if (less-register-number = true)
then (select Intel)

Rules attached on M-specialization for selection of M-6800, M-68000, M-68020, or M-68030 Motorola microprocessors:

R8 if (max-speed >= 25 MHz)
then (select M-68030)

R9 if (max-speed >= 20 MHz)
then (select M-68020)

R10 if (max-speed > 12.5 MHz and data-bus-bit <= 16)
then (select M-68000)

R11 if (max-speed <= 4MHz or data-bus-bit <= 8)
then (select M-6800)

Rules attached on I-specialization for selection of I-8088, I-80286, I-80386, or I-80486 Intel microprocessors:

R12 if (max-speed >= 25MHz)
then (select I-80486)

R13 if (max-speed >= 20 MHz or data-bus-bit = 32 or
multitasking = true)
then (select I-80386)

R14 if (max-speed >= 16 MHz and data-bus-bit = 16)
then (select I-80286)

R15 if (M-speed >= 8 MHz and data-bus-bit = 8)
then (select I-8088)

Rules attached on color specialization for selection of color monitor or monochrome monitor:

R16 if (graphic-quality = medium-high and cost = medium-high)
then (select color)

R17 if (graphic-quality = medium or cost = medium)
then (select monochrome)

Rules attached on resolution specialization for selection of Herculus, CGA, EGA, or VGA monitor:

R18 if (resolution = high and cost = high)
then (select VGA)

R19 if (resolution = medium and
(cost = medium or cost = medium-high))
then (select EGA)

R20 if (resolution = low or cost = low)
then (select CGA)

R21 if (graphics-quality = low and cost = low)
then (select Herculus)

Rules attached on mounting specialization for selection of 5-1/4", 3-1/2", or hybrid floppy disk drive:

R22 if (disk-drive-number = 2)
then (select hybrid)

R23 if (disk-drive-number = 1 and size = 360 Kbytes)

then (select 5-1/4")

R24 if (disk-drive-number = 1 and size = 720 Kbytes)

then (select 3-1/2")

Rules attached on O.S. specialization for selection of UNIX, OS/2 or DOS:

R25 if (multitasking = true)

then (select UNIX)

R26 if (multitasking = true)

then (select OS/2)

R27 if (multitasking = false)

then (select DOS)

Rules attached on device specialization for selection of monitor, printer, or floppy disk drive, and hard disk:

R28 if (cost = low or cost = medium or cost = high)

then (select monitor)

R29 if (cost = low-medium)

then (select monitor and hard-disk)

R30 if (cost = medium or cost = high)

then (select monitor and floppy-disk-drive)

R31 if (cost = high)

then (select monitor and floppy-disk-drive and printer)

SYNTHESIS RULE SET:

Rules attached on physical decomposition for synthesis the sub-system:

R32 if (memory-spec = with-virtual-memory and
(I-spec = 80286 or I-spec = 80386 or M-spec = 68030))
then (OK)

R33 if (M-spec = 68000)
then (memory-spec != with-virtual-memory)

R34 if (I-spec = 8088)
then (memory-spec = without-virtual-memory)

R35 if (M-spec = 6800 or I-spec = 8088)
then (O.S.-spec != OS/2)

Rules attached on the personal computer system for synthesis of the whole system:

R36 if (manufacture-spec = Apple or manufacture-spec = Compaq)
then (O.S.-spec != OS/2)

R37 if (manufacture-spec = Apple)
then (O.S.-spec = UNIX)

Now we define performance requirements about the personal computer system.

quality-of-computer = high

cost-of-computer = high

data-swapping = true

less-register-number = true

max-speed = 20 MHz

data-bus-bit = 32

graphic-quality-of-monitor = high

cost-of-monitor = high

resolution-of-monitor = high

disk-drive-number = 2

multitasking = true

With the above performance requirements, the personal computer system entity structure is pruned by Command15 (Pruning). The resultant of the system model is depicted in Figure 6.1.

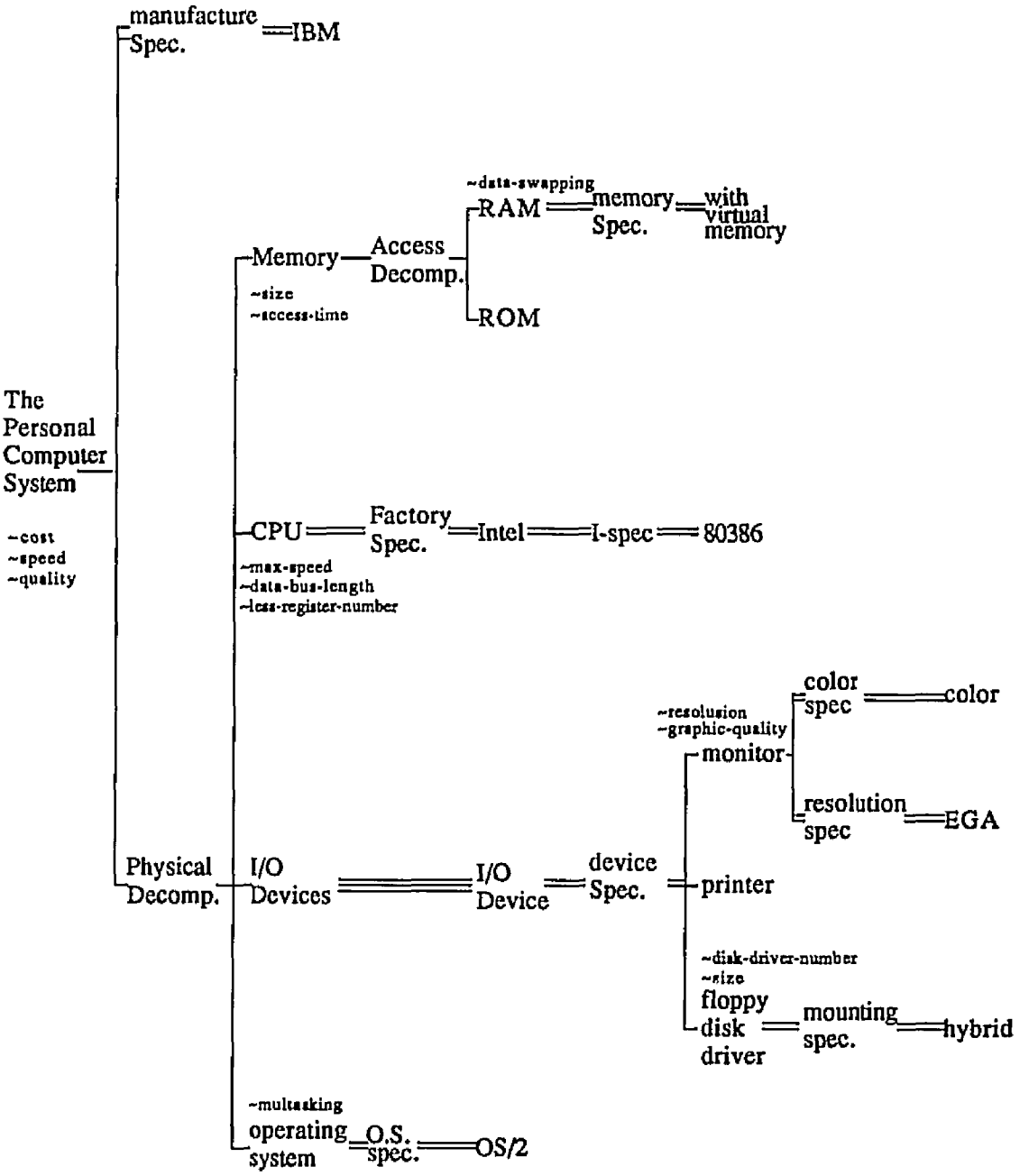


Figure 6.1 The pruned system entity structure of personal computer system

CHAPTER 7

CONCLUSION

In this thesis, an expert system shell for supporting system design has been implemented in Common LISP. This approach gives the user a clear concept about what kind of model is generated by the system as well as the scope of the knowledge base. Improvement has been made to the following aspects:

(1) System design has been placed in the framework of system modelling and artificial intelligence.

(2) A new reliable and efficient knowledge representation scheme - FRASES was introduced in the design of the knowledge base. The scheme combines the multifaceted modelling methodology and artificial intelligence techniques (frame and production rule system), which allows us to easily represent, acquire, and infer knowledge.

(3) A user-friendly rule interpreter was implemented in the shell system. The rules defined the by user were the objectives and the constraints about the system being designed. Therefore we also call this system design process a *constraint-driven* process. The format of a rule can be easily given by the user. It can handle not only the numerical and logical computations but also operation precedence.

(4) Multiple inference algorithms were incorporated into the inference engine design. They infer a set of expected system configurations based upon the

designer's objectives and constraints. After performance comparison among those algorithms, we concluded that for the FRASES knowledge representation scheme the top-down reasoning depth-first algorithm is more efficient than the others, saving both effort and memory space.

The following work is still needed to make the system complete:

(1) Other inference engine topologies will be developed in the system. For example, combining the top-down reasoning and bottom-up reasoning may increase the performance of the inference engine.

(2) Incorporate a graphics system into the entity structure display. As well, the user interface needs to be improved.

(3) Determine the best formula for certainty computation.

LIST OF REFERENCES

1. Fikes, R. and Kehler T., (1985) "The Role of Frame-based Representation in Reasoning", in *Communications of the ACM*, Vol. 28, No. 9, pp. 904-920.

2. Hayes-Roth, F., (1985) "Rule-based Systems", in *Communication of the ACM*, Vol. 28, No. 9 pp. 921-932.

3. Horowitz, E., and Sahni, S., (1984) "Fundamentals of Data Structures in Pascal", McGraw-Hill.

4. Huang, Y. M. (1987) "Building an Expert System Shell for Model Synthesis in Logic Programming", in *Master Thesis, Dept. of Electrical and Computer Engineering, The University of Arizona, Tucson, Arizona.*

5. Rich, E., (1983) "Artificial Intelligence", McGraw-Hill.

6. Rozenblit, J. W., (1986) "A Conceptual Basis for Integrated, Model-Based System Design", in *Technical Report, Dept. of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona,* January, 1986

7. Rozenblit, J. W., Hu, J. and Huang Y. M., (1989) "An Integrated, Entity-Based Knowledge Representation Scheme for System Design," to appear *in 1989 NSF Engineering Design Research Conference*, Amherst, Mass. June 1989.

8. Rozenblit, J. W. and Huang, Y. M. (1987) "Constraint-Driven Generation of Model Structures", in *Proceedings of the 1987 Winter Simulation Conference*, pp. 604-611

9. Rozenblit, J. W., Sevinc, S. and Zeigler, B. P. (1986) "Knowledge-Based Design of LANs Using System Entity Structure Concepts", in *Proceedings of the 1986 Winter Simulation Conference*, pp. 858-865

10. Rozenblit, J. W. and Zeigler, B. P., (1988) "Design and Modelling Concepts", in *International Encyclopedia of Robotics, Application and Automation*, pp. 308-322.

11. Rozenblit, J. W. and Zeigler, B. P., (1985) "Concepts for Knowledge-Based System Design Environments", in *Proceedings of the 1985 Winter Simulation Conference*, pp. 223-231

12. Winston, P. H., (1984) "Artificial Intelligence", Addison-Wesley, Reading, MA.

13. Winston, P. H., and Horn, Berthold, K. P., (1984) "LISP".

14. Zeigler, B. P., (1984) "Multifaceted Modelling and Discrete Event Simulation", Academic Press, London.