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 9003485

Towards a knowledge-based design support environment for design automation and performance evaluation

Hu, Jhyfang, Ph.D.

The University of Arizona, 1989

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

TOWARDS A KNOWLEDGE-BASED DESIGN SUPPORT ENVIRONMENT FOR DESIGN AUTOMATION AND PERFORMANCE EVALUATION

by

Jhyfang Hu

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1989

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Final Examination Committee, we certify that we have read the dissertation prepared by JHYFANG HU

entitled Towards A Knowledge-Based Design Support Environment for Design Automation and Performance Evaluation

and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Jerzy W. Rozenblit	Jerzy W. Rozenblit	6/5/89 Date
Bernard P. Zeigler	B P Zeigler	6/5/89 Date
Ralph Martinez	R Martinez	6/5/89 Date
		Date
		Date

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

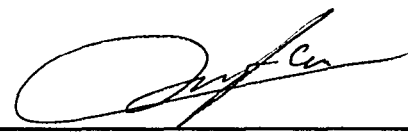
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Jerzy W. Rozenblit Jerzy W. Rozenblit 6/5/89
Dissertation Director Date

STATEMENT BY AUTHOR

This dissertation 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 the rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: 

ACKNOWLEDGEMENTS

I owe thanks to many people who directly or indirectly made this work possible. Professor Jerzy W. Rozenblit, over the last three years has given me invaluable guidance, assistance, and support in moments of doubt. Without his help, this research could not have been completed - to him, my special thanks.

I would also like to express my gratitude to Professor Bernard P. Zeigler, Professor Ralph Martinez, Professor Olivia R. Liu Sheng, and Professor Sudha Ram for their advice and help in completing my degree requirements.

Also, I would like to extend my appreciation to Ms. Xuemei Shao for her concern for my wellbeing during my research.

My final thanks goes to my parents. Their continual support and encouragement has been the most important contribution to the completion of my research.

TABLE OF CONTENTS

TABLE OF CONTENTS — Continued

TABLE OF CONTENTS — Continued

TABLE OF CONTENTS — Continued

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS – Continued

LIST OF TABLES

ABSTRACT

The increasing complexity of systems has made the design task extremely difficult without the help of an expert's knowledge. The major goal of this dissertation is to develop an intelligent software shell, termed the Knowledge-Based Design Support Environment (KBDSE), to facilitate multi-level system design and performance evaluation.

KBDSE employs the technique, termed Knowledge Acquisition based on Representation (KAR), for acquiring design knowledge. With KAR, the acquired knowledge is automatically verified and transformed into a hierarchical, entity-based representation scheme, called the Frame and Rule Associated System Entity Structure (FRASES). To increase the efficiency of design reasoning, a Weight-Oriented FRASES Inference Engine (WOFIE) was developed. WOFIE supports different design methodologies (i.e., top-down, bottom-up, and hybrid) and derives all possible alternative design models parallelly. By appropriately setting up the priority of a specialization node, WOFIE is capable of emulating the design reasoning process conducted by a human expert.

Design verification is accomplished by computer simulation. To facilitate performance analysis, experimental frames reflecting design objectives are automatically constructed. This automation allows the design model to be verified under various simulation circumstances without wasting labor in programming math-intensive models. Finally, the best design model is recommended by applying Multi-Criteria Decision Making (MCDM) methods on simulation results.

Generally speaking, KBDSE offers designers of complex system a mixed-level design and performance evaluation; knowledge-based design synthesis; lower cost and faster simulation; and multi-criteria design analysis. As with most expert systems, the goal of KBDSE is not to replace the human designers but to serve as an intelligent tool to increase design productivity.

CHAPTER ONE

STATEMENT OF THE PROBLEM

Engineering design is a highly complex and creative process. Each design phase requires a tremendous amount of technical knowledge. The increasing complexity of systems has made the design task extremely difficult without the help of an expert's knowledge. Unfortunately, expert's highly technical knowledge is usually valuable and rare. You may not get access to experts when you need them. Think about today's highly competitive industrial market, if experts are not available, what happens? Solving design problems in less time at a lower cost has become the key to survival in today's market war. The ideal solution for solving today's design problems is to have all required experts standing by at your work place. As mentioned before, experts are rare and expensive. Therefore, it is impractical and probably impossible to have all necessary experts available during the design process. Fortunately, the technology of Artificial Intelligence (AI) has made this possible by duplicating the expert's knowledge into a software program, termed the expert system.

1.1 Motivation

In order to accomplish the design process in less time at a lower cost, the need for a high-efficient and error-free design support environment has been increasing rapidly. A knowledge-based system which employs the AI techniques is recognized as one of the best solutions for implementing such high performance systems. At different design phases, appropriate expert systems are employed to help users deal with design problems and to facilitate design process. How to integrate multiple expert systems at different design stages to assist in engineering design poses a very challenging research problem.

In recent years Computer-Aided Design (CAD) systems have been widely used to help engineering design. Most CAD programs are algorithmic in nature. Based on certain assumptions, these CAD programs perform calculations and analysis by applying mathematical formulas. Although most design problems can be solved by using the algorithmic approach, many do not lend themselves to an algorithmic solution. For example, design objectives, requirements, and constraints can be fuzzy, conflicting, and insufficient for making a right decision. In such cases, the designer is forced to make a decision based on incomplete, inconsistent, or uncertain information, and such a decision might be incorrect. More intelligent systems are required to overcome today's engineering design difficulties.

Given the same design problem, multiple solutions may exist under different assumptions and requirements. How to choose the most appropriate solution model for a particular design problem requires some knowledge. Faced with hundreds of programs in a complete CAD system, the novice designer may have extreme difficulty in selecting the right solution model, providing valid design parameters, and verifying or extracting effective results for each design phase. Without an expert's help, the novice designer would probably never solve his design problem efficiently. All the above technical problems found in modern engineering design motivate this research work.

1.2 Objectives

The major objective of this dissertation is to combine the most advanced AI technology with existing modelling and simulation concepts to improve productivity and reliability of modern engineering design. Specific goals of this research are:

1. To propose a powerful knowledge representation scheme for AI knowledge-based engineering design applications. The proposed representation scheme should be able to represent design knowledge properly and efficiently. This scheme should also facilitate knowledge management tasks such as acquisition, representation, control (inference), and refinement.

2. To propose a knowledge acquisition approach that will facilitate the development of a knowledge base. To reduce development time and cost, the proposed acquisition method should be able to a.) generate query patterns automatically; b.) provide customized explanations or instructions to help users complete the acquisition process; c.) validate the information provided by users; and d.) translate the validated information into a form that is ready for machine inference.

3. To propose an inference engine that will perform design reasoning both efficiently and reliably. The proposed inference engine should capture all the dynamics of an engineering design. To achieve the best result, the inferencing engine is capable of incorporating different methodologies (e.g., top-down, bottom-up, or hybrid) into the design reasoning process.

4. To design an automatic program generator in order to facilitate performance modelling. This automation allows designers to test their design models with different simulation requirements without worrying about programming or debugging a math-intensive performance model. This, in turn, reduces the overall simulation cycle.

5. To extend the multifaceted design concepts [Rozenblit and Zeigler 1988] and propose an integrated knowledge-based design support environment to facilitate the engineering design process and performance evaluation by

integrating techniques offered by AI expert systems, databases, Computer-Aided Design (CAD), and program automation.

By solving most design difficulties with expertise and artificial intelligence methods, the overall design cycle and cost can be significantly reduced.

1.3 Approach

The approach taken in this study will utilize methods offered by artificial intelligence, modelling and simulation theory, formal languages, and software design. Knowledge- based design approach has been recognized as one of techniques having the most potential for solving domain specific problems efficiently and intelligently. By providing design constraints and heuristics in the knowledge base, expert systems can be used to solve knowledge-intensive problems in less time at a lower cost. The basic idea behind this research would be to integrate a number of knowledge- based systems to provide expert-level assistance in the engineering design process.

The ideal machines for implementing the proposed design support system are AI machines (e.g., TI-Explorer [TI 1986]) or engineering workstations supporting AI programming languages (e.g., LISP [Winston and Horn 1984]), graphic capability, and window systems (e.g., X Windows [Scheifler and Gettys 1986], Sun NeWS [Sun 1987]). In this dissertation Common LISP is employed for prototype implementation and testing of the system operations.

1.4 Organization

This dissertation has been organized into seven chapters. Chapter 1 provides a survey of engineering design problems and how these problems motivate this research. Chapter 2 defines an integrated, entity-based knowledge

representation scheme, called the Frames and Rules-Associated System Entity Structure (FRASES), for engineering design applications. Formal definitions of FRASES will be given in detail. Chapter 3 proposes a new approach, termed Knowledge Acquisition based on Representation (KAR), for acquiring engineering design knowledge. In Chapter 4, a Weight-Oriented FRASES Inference Engine (WOFIE) is introduced for deriving all possible alternative design models. In Chapter 5, the methodology for automatic generation of experimental frames is developed to facilitate performance evaluation of design models. Chapter 6 develops a prototype architecture for the integrated Knowledge-Based Design Support Environment (KBDSE). Architecture, requirements and implementation of the system are discussed. Finally, in Chapter 7, the conclusion, comments on possible future improvements for the prototype design support environment.

Throughout this dissertation, examples for the design of distributed systems will be used to illustrate applications of the proposed design methodology. Although these examples only deal with design problems at an abstract level, design applications at more detailed levels can be easily expanded by adding required domain knowledge.

CHAPTER TWO

FRASES - A KNOWLEDGE MANAGEMENT SCHEME FOR ENGINEERING DESIGN

The increasing demand for quality knowledge-based systems has resulted in knowledge representation becoming a major topic in AI research. Along with the rising complexity of engineering design problems, knowledge management tasks such as acquisition, control (inference), and refinement become increasingly difficult. A good knowledge representation scheme facilitates knowledge management and assures the reliability and efficiency of a knowledge-based system. In other words, the performance of an expert system is highly affected by the way that its knowledge is represented.

In this chapter a new knowledge representation scheme, termed Frames and Rules-Associated System Entity Structure (FRASES), will be presented. With FRASES, complex engineering design knowledge is organized into a hierarchical, entity-oriented tree structure that simplifies management of complex design knowledge.

2.1 Knowledge Representation in AI

The primary concern for building an expert system is how to translate and represent the expert's knowledge into a form that the computer can read or use. Normally, knowledge used by an expert system is classified into two types:

- Declarative knowledge: This type of knowledge is primarily a description of facts about an object. Declarative knowledge states information, deduces relationships, and classifies objects.

- Procedural knowledge: This type of knowledge provides a way of applying the declarative knowledge. By showing procedures for a course of actions, procedural knowledge recommends what to do and how to do it.

A good knowledge representation should be capable of representing declarative as well as procedural knowledge. A variety of schemes have been exploited for representing knowledge (facts) in AI programs.

2.1.1 Scripts

Scripts [Schank and Abelson 1977] are designed to encode stereotypical event sequences. A script is a structural representation with basic elements such as entry conditions, roles, props, scenes, and results. The entry conditions specify requirements to invoke the system represented. The roles stand for elements of the system. The props are attributes of the elements. The scenes represent events that will occur.

2.1.2 Semantic Networks

The semantic network [Quillian 1968, Shastri 1988] is a graphic knowledge representation scheme that shows the relationship between objects. A semantic network consists of nodes and arcs. Each node represents an object and the labeled arc states the relationship between two connected objects. For illustration, a semantic network representation for the computer module of a distributed system is shown in Figure 2.1.

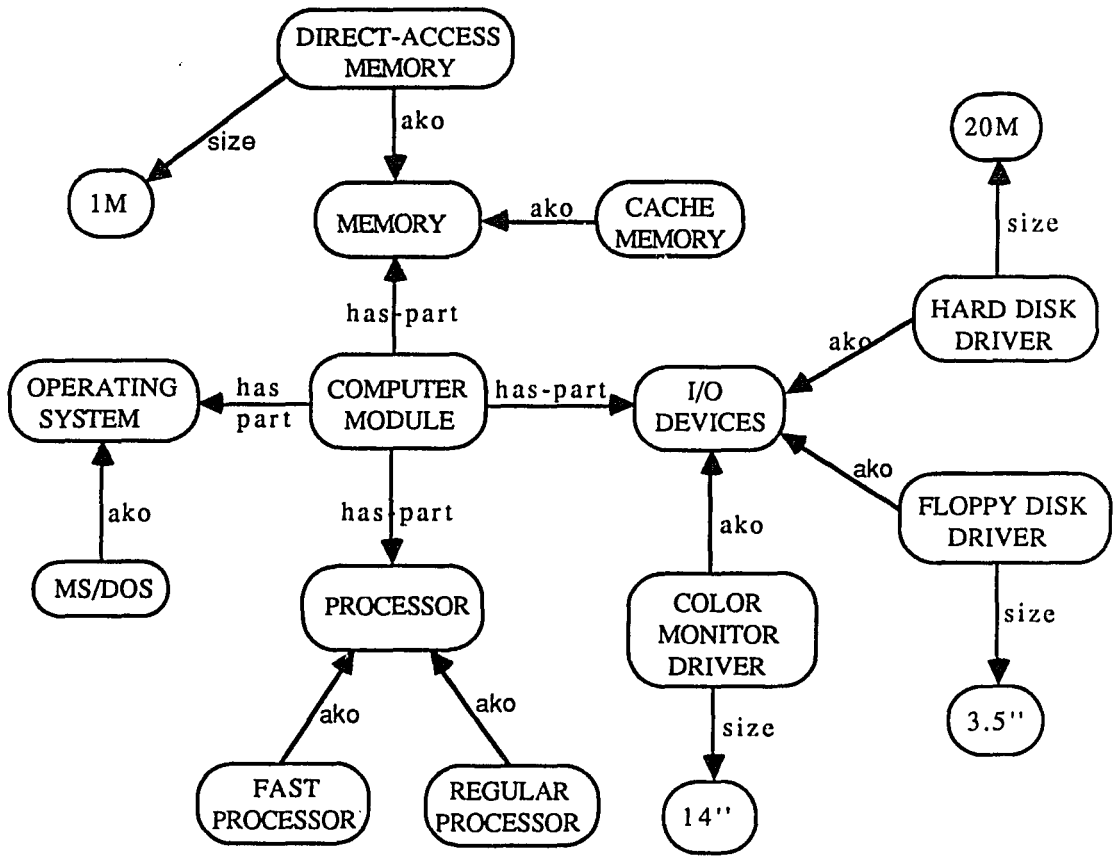


Figure 2.1 A computer module in semantic networks

2.1.3 Production Rules

A production rule [Newell and Simon 1972] states knowledge with two parts: condition and action. Production rules are written in an IF-THEN clause. The IF part states a situation or a premise and the THEN part states an action or a conclusion. A rule is triggered if current facts match the antecedent (or condition) part of the rule. Conflict resolution strategies [Winston 1984] are applied to select the rule to be fired when multiple rules are triggered. Once a rule is fired, its action part is carried out. A typical production rule has the following format: *IF (conditions) THEN (actions).*

2.1.4 Frames

The frame [Minsky 1975] is primarily designed to handle declarative knowledge. For illustration, a typical frame representation for the computer module of a distributed system is shown in Figure 2.2. A frame is a generalized property list which can be divided into discrete elements called "slots". Each slot describes an attribute which may, in turn, contain one or more facets such as "value", "default", "if-needed", "if-added", or "if-accessed". And each facet can have a number of values. Each value can be either a number, a symbol, or a procedure. Procedures that are activated automatically when a value is needed (i.e., if-needed), when a value is placed (i.e., if-added), or when a value is removed (i.e., if-removed), are called "demons". Basically, a frame object has the following structure:

```
(frame-name
    (slot-1 (facet-1 value-1 value-2 ..)
            (facet-2 value-1 ..) ...)
            ...
```

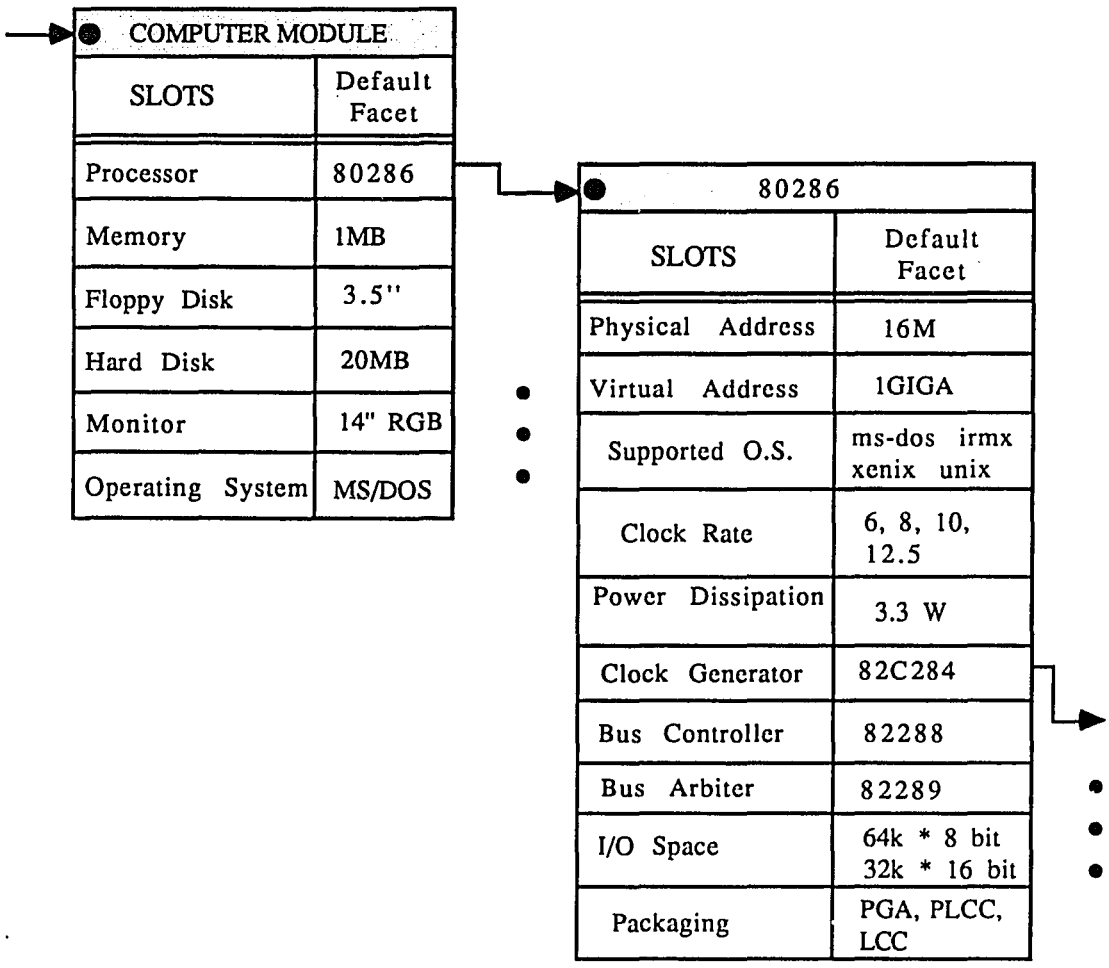


Figure 2.2 A computer module in frames

(slot-n (facet-1 value-1 value-2 ..)
(facet-2 value-1 ..) ...))

To manipulate frame objects, a frame system featuring defaults, demons, and inheritance is implemented and attached in Appendix 2.1 for reference. This implementation of frames uses association lists. An association list is a list of elements, each of which can be identified and extracted from the list using its key. For example, to retrieve information about the processor of a computer module, users may use the following frame message:

(frame-z-get 'computer 'processor)

The frame system will first try the value facet in the processor slot of the computer module. If a value is not found, the default facet is tried. If there is no default values, the frame system will invoke the "if-needed" demon to derive the required value by either asking information from users or computing it from other slot values. Finally, if there is no demon for the processor slot, the Z inheritance procedure [Winston 1984] will be activated to inherit properties from frames related by a-kind-of (or ako) relationship.

2.1.5 Predicate Logic

Predicate logic [Chang 1973, Tanimoto 1987] represents knowledge into a **P(a), Q(x, y), or R(a, f(b))** formula. The predicate symbol **P**, **Q**, and **R** are used to denote qualities or attributes of objects or relationships among objects. Constant symbols **a** and **b** are used to denote particular objects in the problem domain. The **x** and **y** indicate variable symbols and the **f** denotes a function symbol. Predicate logic is an attractive knowledge representation mechanism because logical reasoning can be easily applied to such a scheme. Typical predicate logics are:

premises	*Fly(Bird).* *Isa(Eagle, Bird).*
conclusions	*Fly(Eagle).*

2.1.6 AND/OR Trees

AND/OR trees [Nilsson 1980] depict knowledge by laying out AND and/or OR relationships between objects. Each node (except leaves) in an AND/OR tree has successor nodes, each of which can be either an AND node or an OR node. Each AND node denotes a decomposition of its predecessor. Each OR node represents an alternative of its predecessor. A typical AND/OR tree for representing the computer module of a distributed system is shown in Figure 2.3.

2.1.7 System Entity Structures

A system entity structure [Zeigler 1984, Zeigler 1987] is a labelled tree with attached variable types which satisfies the following axioms:

- Uniformity: Any two nodes which have the same labels have identical attached variable types and isomorphic substructures.
- Alternating Mode: Each node has a mode which is either "entity" or "aspect". The modes of a node and its successors are always different.
- Valid Brothers: No two nodes under the same parent have identical labels.
- Attached Variables: No two variable types attached to the same entity have the same name.
- Strict Hierarchy: No labels appear more than once down any path of the system entity structure tree.

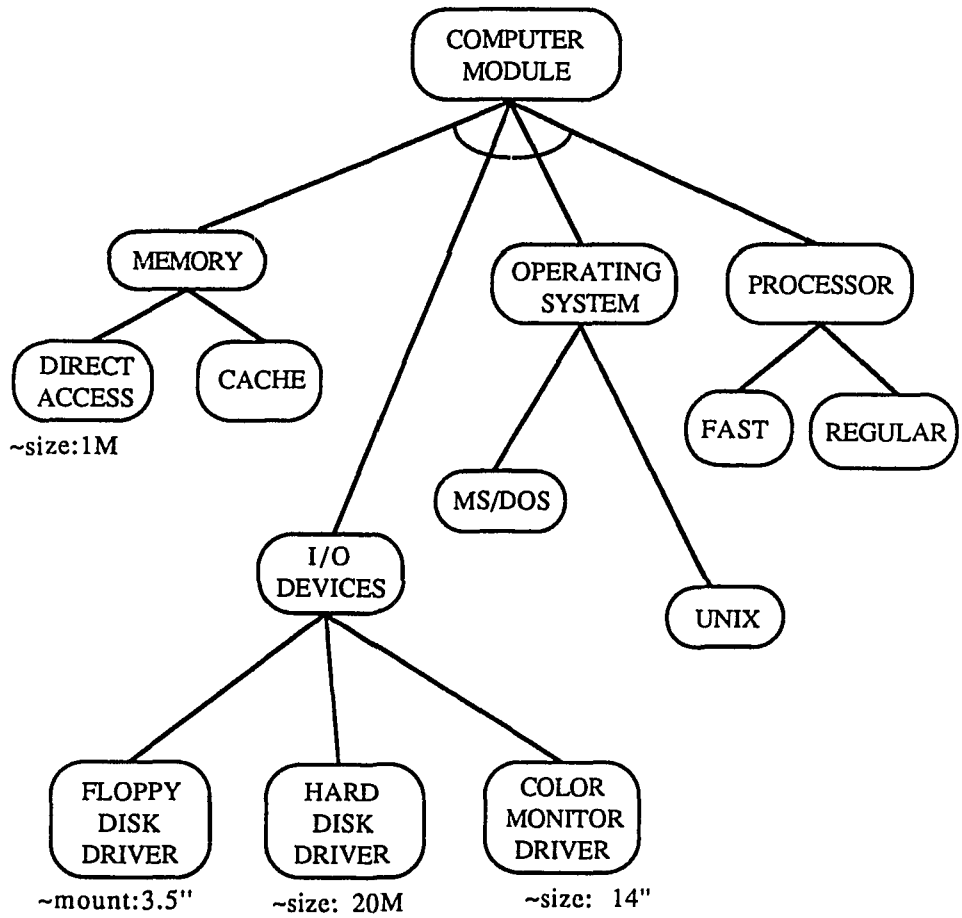


Figure 2.3 A computer module in AND/OR trees

Knowledge is represented in the system entity structure by depicting decomposition, taxonomy, and coupling of objects. A typical system entity structure for representing the computer module of a distributed system is illustrated in Figure 2.4. As shown in the figure, a multiple decomposition (triple vertical bars) is employed to facilitate the representation of multiple entities whose number may vary in the system. To eliminate the need to repeat information at each node, each specialization variant inherits properties and substructures from the parent entity to which it is related by specialization relationship. For knowledge manipulation, the system entity structure allows the following operations [Zeigler 1984, Hu 1987]:

- Naming Scheme: Each entity has a unique path to the root from which a unique name can be generated for the recursive node.
- Distribution and Aggregation: The system entity structure is able to generate distribution or calculate aggregation for a multiple entity.
- Transformation: This operation is used to convert the system entity structure into taxonomy free form, or to lay out choices have been made. Typical transformations of system entity structures are shown in Figure 2.5.
- Pruning: The pruning (Figure 2.6) is a process for deriving desired models by comparing system requirements with the specification of component models. Pruning a system entity structure will result in a set of alternative design models, each of which is termed a composition tree. A composition tree contains only entities and decompositions. Each entity of a composition tree has at most one aspect (decomposition).

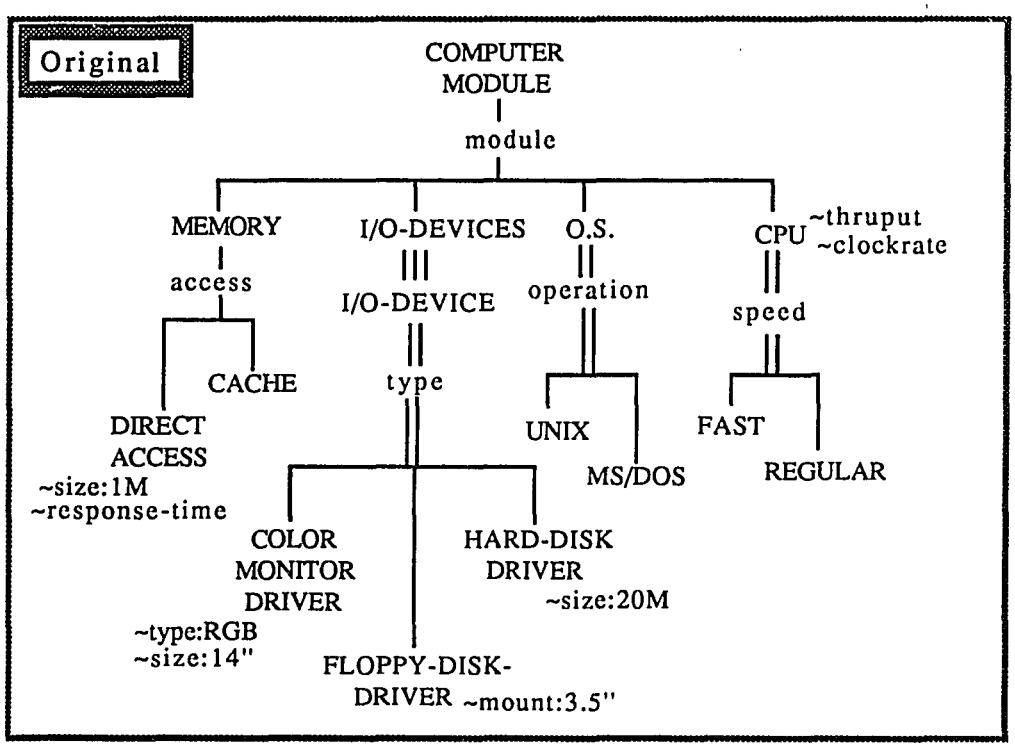


Figure 2.4 A computer module with system entity structure

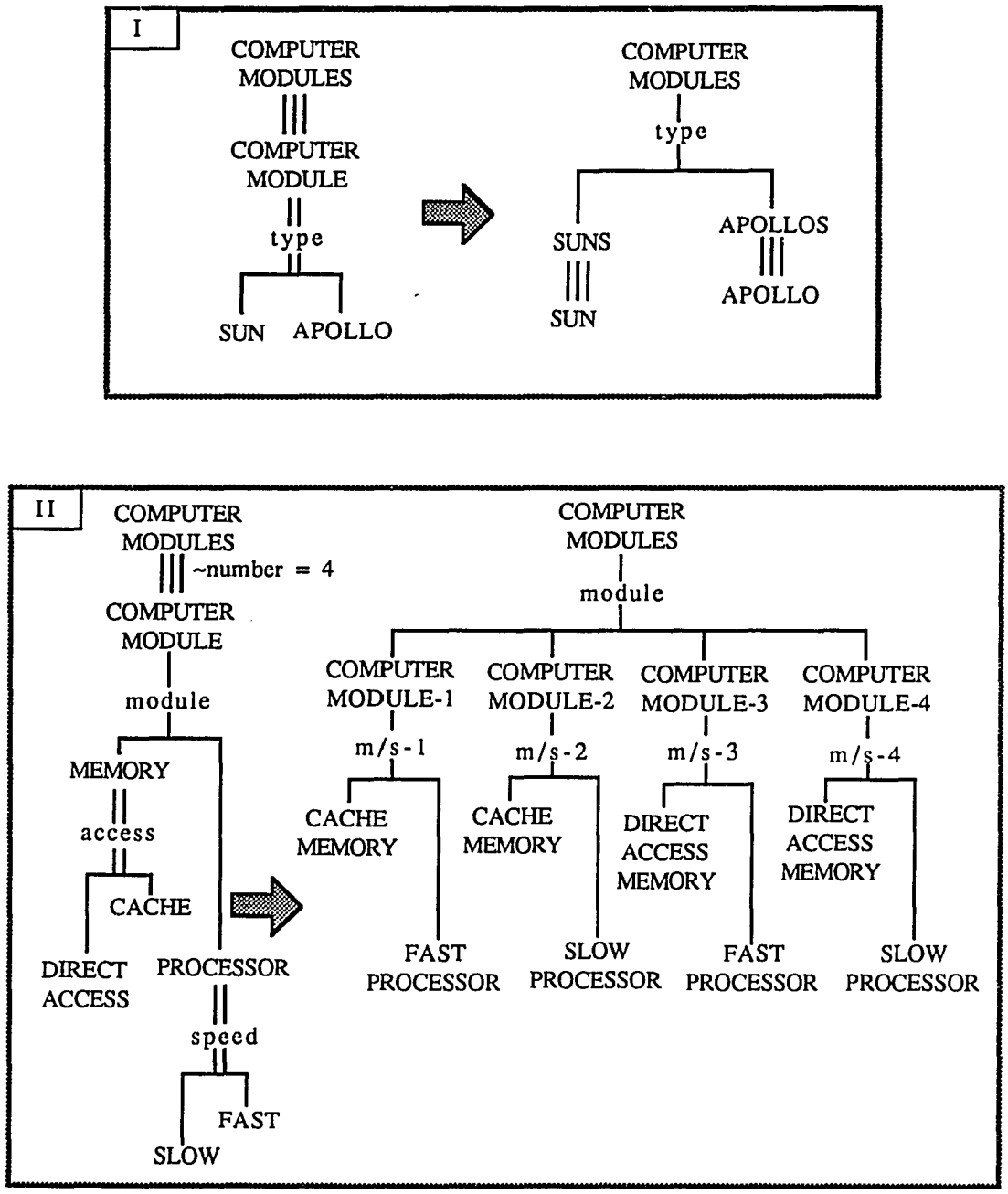
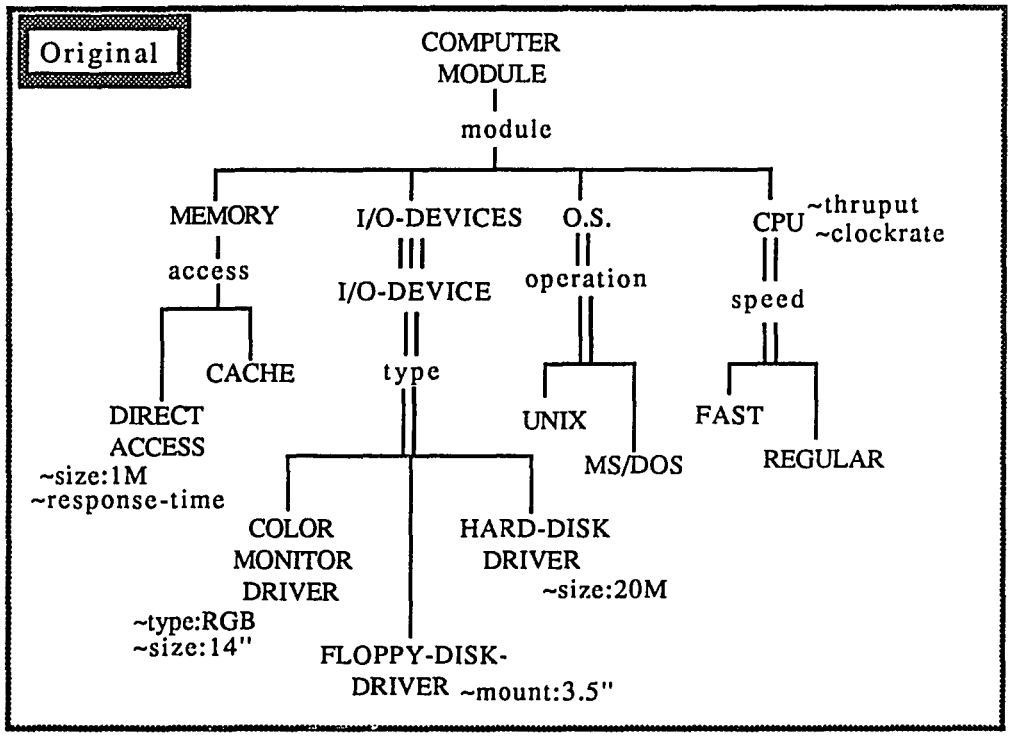


Figure 2.5 Transformation of system entity structures



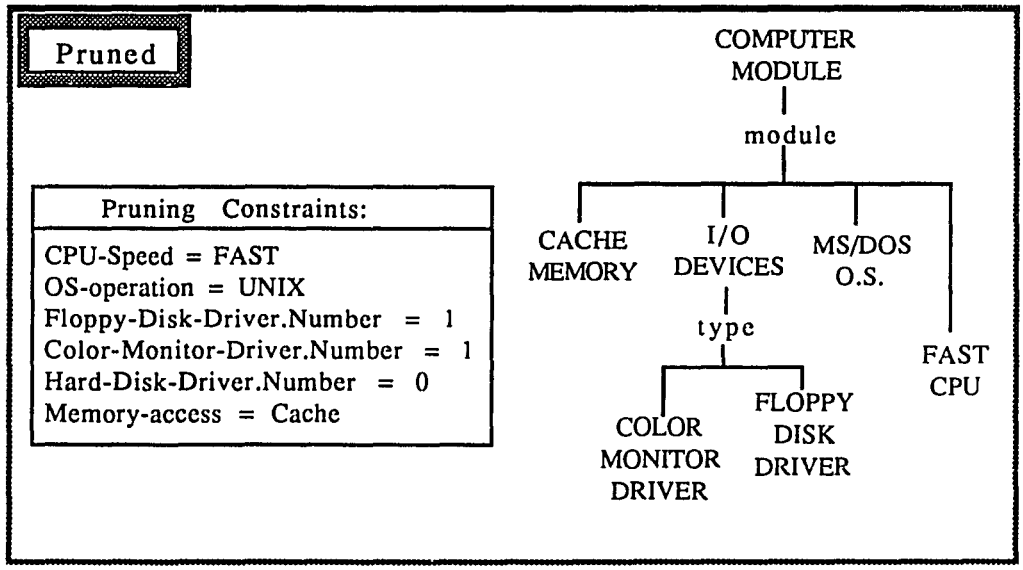


Figure 2.6 Pruning of system entity structures

2.2 Requirements for System Design Representation

When reviewing the common traits in system design, one will find that structured techniques are used to reduce the complexity of the design process. Common traits found in modern design approaches are [Weste and Eshraghian 1985]:

- Hierarchy: The use of hierarchy involves dividing a system into subsystems and then repeating this operation on the subsystems until the complexity of the subsystems is at an appropriate or desired abstraction level.

- Modularity: Modularity helps designers reduce the complexity of system models and clarify an approach to a problem. A modular design facilitates flexibility and future modifications. The ability to divide a task into well-defined modules also aids in a team design where a number of designers are assigned a portion of a complete system to design.

- Regularity: Employing regular structure to simplify the design process is gaining its popularity. Regularity can exist at all levels of the design hierarchy. As seen in computer system design, uniform transistors are used at the circuit level; identical gate structures are employed at the logic level. At a higher level, a multi- processor system is designed with identical processors.

In general, a good knowledge representation scheme for system design applications must be able to denote the above properties within its structure. Furthermore, the knowledge representation scheme should be capable of capturing both static and dynamic knowledge of the system such as:

it Static knowledge:

1. Topological/coupling information of objects.
2. Taxonomy/decomposition of objects.
3. Properties/attributes of objects.
4. Constraints for design synthesis.

Dynamic knowledge:

1. Behavioral characteristics of objects.
2. Adjustments of design parameters.
3. Configuration of design models.
4. Verification of design models.

A good knowledge representation scheme should also facilitate knowledge acquisition, inference, and refinement. Finally, knowledge reflected by the representation scheme must be clear to domain experts, knowledge engineers, and system users.

In order to encompass all required knowledge for system design, we have combined the system entity structures, frames, and production rules into a hierarchical and entity-based knowledge representation scheme, termed the Frames and Rules-Associated System Entity Structure (FRASES). By exploiting a.) the reasoning flexibility provided by production rules; b.) the efficiency in representing declarative knowledge offered by frames; and c.) the visibility and hierarchy supported by system entity structures, FRASES becomes a powerful and efficient scheme for managing domain knowledge that supports design model development.

2.3 Structure of FRASES

FRASES is a superclass of system entity structures that encompasses the boundaries, decompositions, and taxonomic relationships of the system com-

ponents being modelled. All axioms (e.g., Uniformity, Strict hierarchy, Alternating mode, Valid brothers, Attached variables) and operations (e.g., Naming scheme, Distribution/aggregation, Transformation, Pruning) defined originally for managing system entity structures are also present in the FRASES representation.

Each entity of FRASES signifies a conceptual part of the system which has been identified as a component in one or more decompositions. Each such decomposition is called an aspect. In addition to decompositions, there is a relation called specialization. It facilitates representation of variants for an entity. Each specialization variant inherits properties and substructures from the parent entity to which it is related. Each FRASES node is associated with an Entity Information Frame (EIF) for describing design knowledge. Every occurrence of an entity has the same Entity Information Frame (EIF) and isomorphic substructure. During design application, knowledge contained in the EIF is extracted and interpreted by the inference engine. An Entity Information Frame (EIF) is a structure:

$$< M,\ ATTs,\ DSF,\ ESF,\ CRS,\ CH >$$

where

M: is the name of the associated model.

ATTs: are attributes of the entity.

DSF: is the design specification form.

ESF: is the experiment specification form.

CRS: are constraint rules for design synthesis.

CH: are FRASES children of the focused node.

With FRASES representation, behavioral knowledge about objects is described by simulation models stored in the model base. M represents the key

to access its behavioral model.

ATTs are attributes used to characterize the associated object. FRASES partitions the attributes of an object into three categories: static attributes, design parameters, and performance indices. Static attributes describe general properties of an object that does not change over time. Static attributes are usually assigned values (i.e., quantitative or qualitative) when they are generated. During application, values of static attributes are directly retrieved without further calculation or simulation work. Typical static attributes are:

- General description of an entity (e.g., design version, date, vendors, etc.).
- Variables used for FRASES manipulation (e.g., node type, processing weight, etc.).

Design parameters are attributes characterizing the design details of an object. To assure a valid design model that satisfies design objectives, quantitative functions, heuristic rules, and CAD tools are required to adjust the design parameters appropriately. Procedural knowledge for adjustment of design parameters is integrated to the FRASES by frame demons. For example, a design parameter may contain an "if-needed" demon (e.g., a heuristic or quantitative function) for estimating the most appropriate value based on design constraints and objectives. This first-time estimated value is then kept in "value" slot for future application. To assure an error-free design, knowledge about constraints (e.g., related to technology) must be provided to validate the estimated value of the design parameter. In other words, each design parameter in FRASES representation has the following structure:

(Design-Parameter (Value ..) (If-needed ..) (Constraint ..))

Performance indices stand for attributes used to evaluate the performance of the design model. In order to enable the automatic generation of

performance models, performance indices are specified in an algebraic expression consisting of design parameters and atomic frames (see Chapter 5). This algebraic expression is termed a Performance Index Tree (PIT). Each PIT indicates how an experimental frame can be aggregated from atomic frames for measuring the associated performance index.

In order to reduce effectively the total number of alternative design models, knowledge must be provided for rough estimation on the performance of design models. In the worst case, the design model may not satisfy the performance requirements. This requires a design refinement by adjusting related design parameters. In other words, knowledge about the dependency between the performance index and related design parameters should be included for performance tuning. Basically, a performance index in FRASES representation may have the structure as follows:

(Index (PIT ..) (Estimate ..) (Tuning (Up ..) (Down ..)))

Design Specification Form (DSF) accepts the specification of design objectives, constraints, and criteria preference. The contents of DSF define the system requirements that must be satisfied by the system being designed. Each entity of FRASES may have its own DSF so that design specification can be described in a hierarchical manner. Experiment Specification Form (ESF) is used to accept the specification of simulation requirements such as an arrival process, workload process, and simulation controls. Again, ESF is placed with entity nodes of a composition tree. Multiple ESF specifications result in a distributed experimental frame organization.

For illustration, a typical DSF specification for the processor of a computer module is shown in Figure 2.7. With the specified DSF, the processor should be capable of executing 10 Million Instructions Per Second (MIPS), the

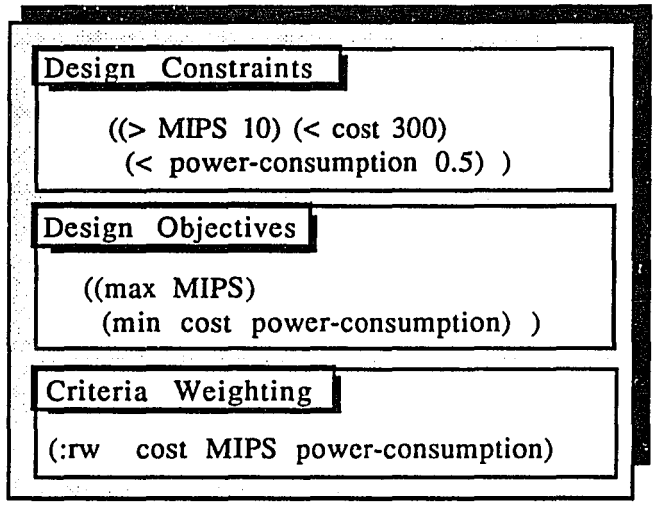


Figure 2.7 Design specification form (DSF)

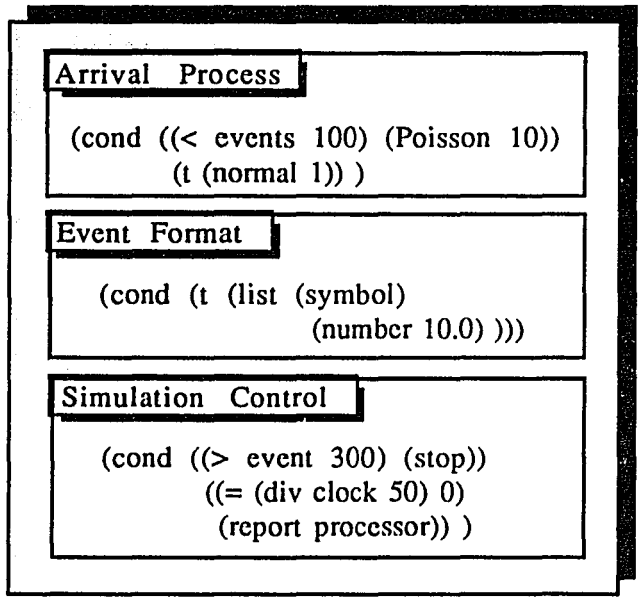


Figure 2.8 Experiment specification form (ESF)

cost of the processor is less than 300 dollars, and the power consumption of the processor must be less than 0.5μ watts. Assume that the discrete event simulation is employed for design verification. Figure 2.8 describes a simplified ESF specification for the processor. The simulation specification indicates that the input arrival rate for the first 100 events will follow the Poisson distribution with the mean value of 10. For subsequent events, the normal distribution with mean of 1 will be employed. Each event is composed of a symbolic identification and numerical workload. Simulation will be executed for 300 events. For each 50 system time units, the measurement of performance indices must be reported.

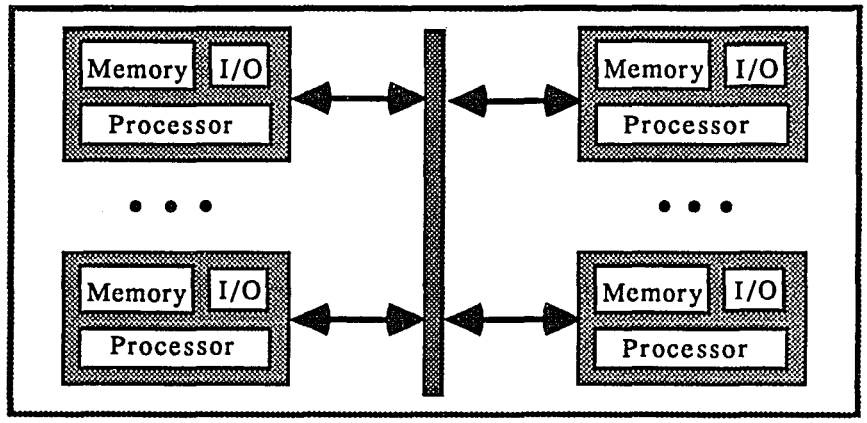
Constraint Rules for design Synthesis (CRS) contains heuristic rules for configuring design structures. Design rules derived from architectural, algorithmic, or technical constraints may also be used to assist in design synthesis. Formally, constraint (selection) rules for pruning alternatives are associated with specialization nodes; and constraint (synthesis) rules for integrating components are associated with aspect nodes.

For illustration, a typical FRASES representation for a LAN-based distributed system is shown in Figure 2.9. As shown in the figure, Design Specification Form (Figure 2.7) and Experiment Specification Form (Figure 2.8) are included to describe design specification and simulation circumstances.

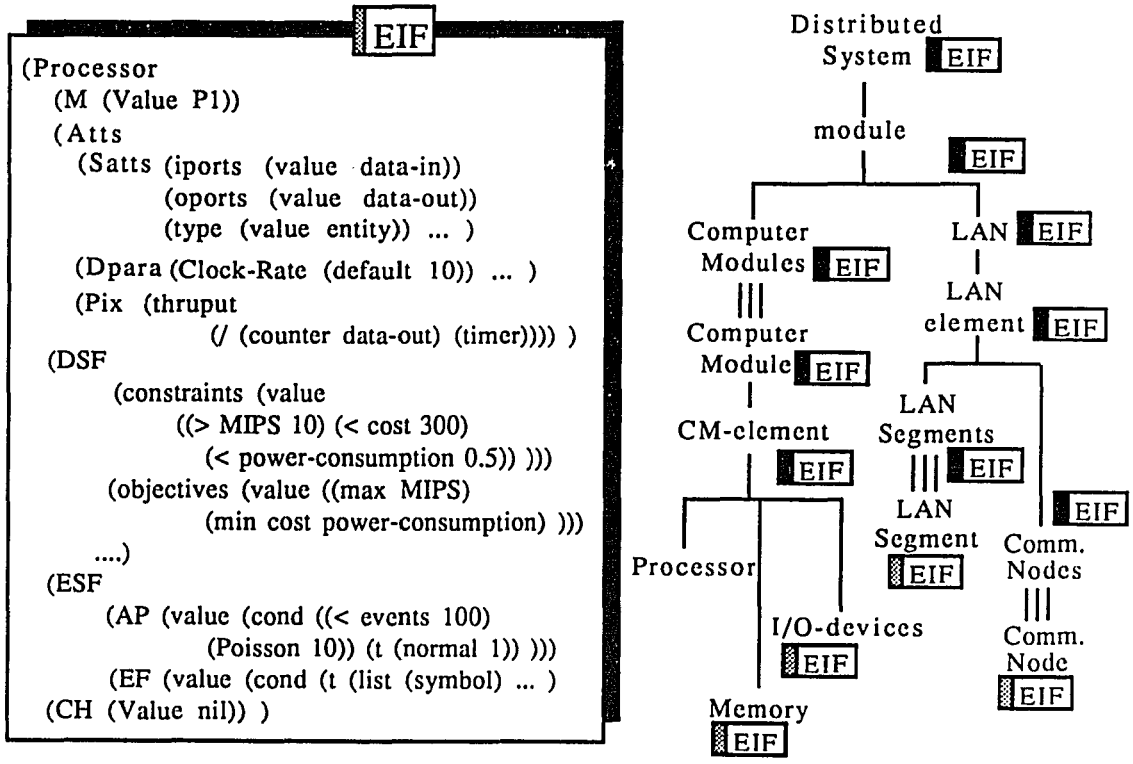
2.4 Advantages of FRASES

Generally speaking, FRASES representation is expected to have the following advantages:

- Generating new knowledge: As in system entity structures, FRASES is a generative representation scheme. Three operations (Figure 2.10) have been defined to generate new knowledge. Considering the design of Local



(a) Schematic representation



(b) FRASES representation

Figure 2.9 A LAN-based distributed system with FRASES

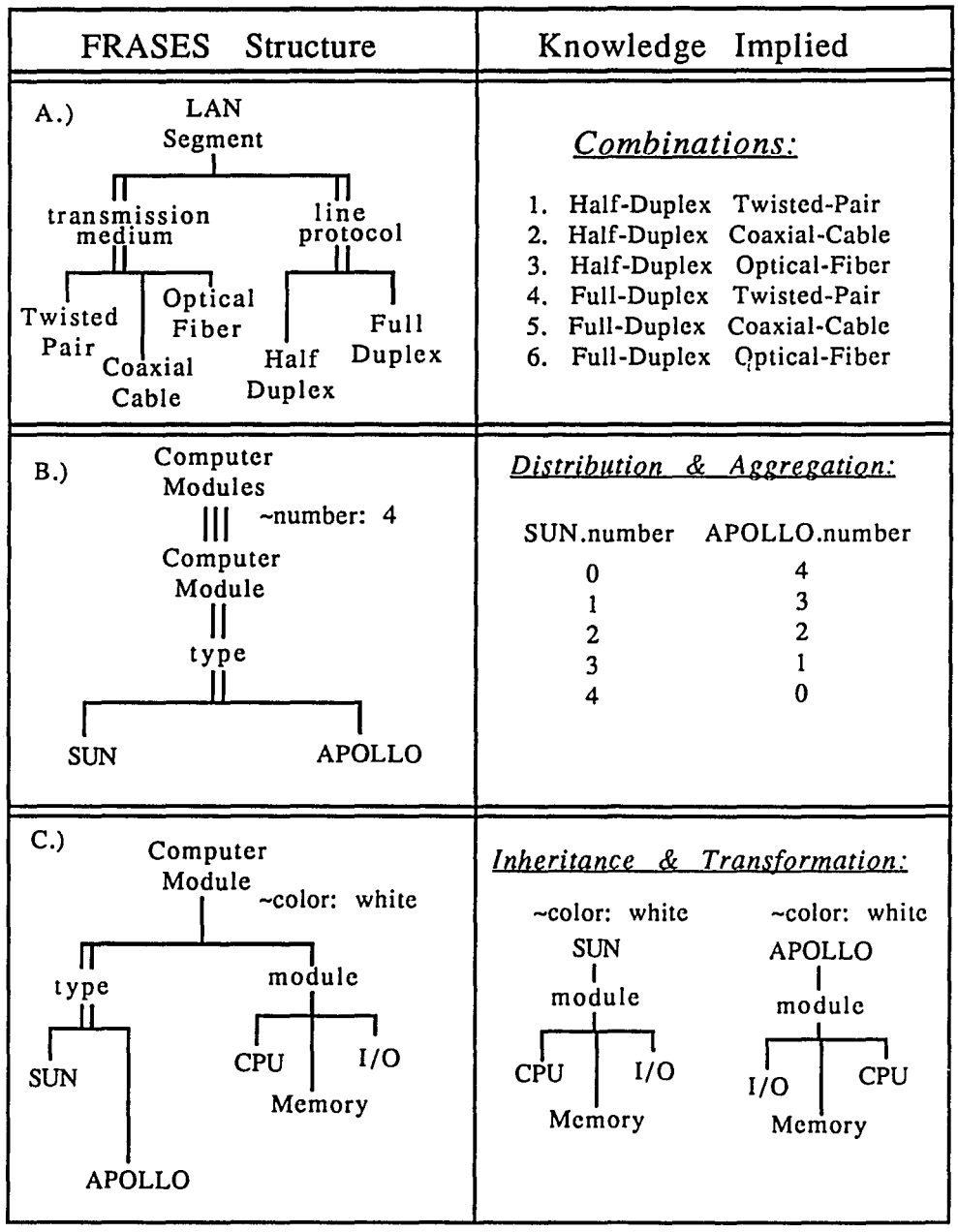


Figure 2.10 Knowledge generation of FRASES

Area Networks (LAN), each LAN segment may be classified into *Twisted-Pair, Coaxial-Cable*, and *Optical-Fiber* based on the *transmission-medium.* On the other hand, a LAN segment can be classified into *Half-Duplex*, and *Full-Duplex* based on the *line-protocol.* If no other constraints are specified, each LAN segment can be interpreted as one of the six possible combinations as shown in Figure 2.10a. In Figure 2.10b, a LAN-oriented distributed system consists of multiple computer modules, each of which can be either a Sun or an Apollo workstation. If four computer modules have been planned for the system and no other constraints are specified, then five possible configurations will be derived. In Figure 2.10c, a computer module is decomposed into *cpu, memory*, and *I/O.* Since Sun and Apollo workstations are specialization variants of the computer module, they will inherit both the property (i.e., "color = white") and the substructure (i.e., cpu, memory, and I/O) from the computer module.

- Organizing knowledge into hierarchy: Exploiting its strict hierarchy, FRASES efficiently describes complex knowledge from an abstract level to more specific level in a hierarchical manner. Local knowledge (i.e., knowledge related only to single node) is represented within its own Entity Information Frame (EIF). Global knowledge (i.e., knowledge related to two or more nodes) is represented within the EIF of their common parent entity. In other words, each Entity Information Frame (EIF) possesses a cluster of knowledge for managing itself and its subtrees. The capability of representing knowledge into hierarchical and modular structures is essential for representing the complex knowledge of system design.

- Facilitating Knowledge Refinement: With FRASES, the knowledge represented can be refined freely both in depth (e.g., levels of abstraction)

and breadth (e.g., details of decomposition) as the technology evolves and changes. Furthermore, the strict knowledge hierarchy of FRASES highly facilitates the task of knowledge refinement. Instead of traversing the whole knowledge base, knowledge segments needed to be updated are localized onto the tree path from the focused entity up to the root node. The inconsistency resulted from neglecting to update all related knowledge chunks can be reduced to a minimum.

- Reducing the Complexity of a Knowledge Base: The characteristics of inheritance and uniformity of FRASES highly reduces the size and complexity of a knowledge base required for the same design application. All the attached knowledge and substructures of an entity will be inherited by its specialization variants. Every time an inheritance is invoked, an XOR operation is employed to inherit substructures and to inherit the knowledge that does not exist in the local frame. Identical nodes dispersed among the FRASES tree are updated automatically and simultaneously according to the axiom of uniformity. This eliminates conflicting description for the same design component.

- Increasing Efficiency of Knowledge Inference: Since FRASES organizes knowledge into a hierarchical and entity-based manner, only rules associated with the focused node has to be examined during each inference cycle. Compared with the global organization in conventional knowledge bases that requires the whole knowledge base to be searched on every inference cycle, FRASES reduces the inference time significantly.

- Adapting to Problem Domains: FRASES is a flexible representation scheme. Unlike other knowledge representation schemes which appear efficient only in specific problem domains, FRASES is flexible for other AI

research applications (Appendix 2.2).

2.5 FRASES vs. System Entity Structure (SES)

The idea behind FRASES is to reinforce the representation power of system entity structures by adding frames and production rules. Major differences between FRASES and SES are:

- Knowledge integration: A system entity structure is a tree with variable types [Zeigler 1984]. Instead of associating variable types with each entity, each FRASES node is associated with a frame object, termed Entity Information Frame (EIF). In SES, information about design specifications and simulation circumstances is not associated with the representation scheme. For better knowledge management, design specification and simulation requirements are included in the FRASES. In SES, procedural knowledge for design synthesis and performance modeling is not contained in the representation scheme. In FRASES, required procedural knowledge is associated with related nodes.

- Attribute structure: In SES, each variable is assigned a single value. In FRASES, each attribute may have multiple values (e.g., value, default, if-needed, etc.), each of which can be a value or a procedure.

- Supported pruning: In SES, a pruning task is accomplished by comparing generic frames [Rozenblit 1985] with variable types associated with each entity. Exhaustive search is required to derive all possible design models. This increases design overhead proportionally with the size of system entity structures. To improve this drawback, FRASES employs a constraint-driven rule-based approach. Undesirable subtrees are cut off at the early pruning stage, there is no need to traverse the whole FRASES tree.

CHAPTER THREE

KNOWLEDGE ACQUISITION BASED ON REPRESENTATION (KAR) FOR DESIGN MODEL DEVELOPMENT

Conventionally, knowledge acquisition and representation are accomplished separately and sequentially. Knowledge engineers are responsible for preparing question patterns and setting up personnel interviews with domain experts for knowledge acquisition. All acquired knowledge is then manually interpreted, verified, and transformed into a predefined representation scheme. In this chapter a new methodology for knowledge acquisition, termed Knowledge Acquisition based on Representation (KAR) is presented. In stead of treating acquisition and representation separately and sequentially, KAR deals with acquisition and representation in an integrated manner. All knowledge acquired with KAR is directly verified and transformed into a representation scheme which is ready for inference. The major objectives of KAR are a.) to automate the knowledge acquisition process; b.) to increase the reliability of a knowledge base; and c.) to reduce the development time and cost of a knowledge base.

3.1 Knowledge Acquisition in AI

Knowledge acquisition is recognized as the main bottleneck in the design of expert systems. Difficulties have been found all over the knowledge acquisition process. Several methodologies have been introduced to help elicit knowledge from experts.

The most common method for knowledge acquisition is an interview [Hart 1985]. In face-to-face interviewing, experts are asked questions and are expected to give informative answers. All details of the interviews must be

recorded for further manual analysis and conversion so that essential knowledge can be extracted and translated into a knowledge representation scheme. However, problems have been discovered in the interview approach: experts are often unaware of specific details of the particular problem; experts are unable to adequately express their knowledge; knowledge engineers are unable to ask all essential knowledge; and experts misunderstand knowledge engineer's questions because of different interpretations of the terminology. All the above situations may result in duplicate, conflicting, or incomplete knowledge.

Some of the interview problems are later resolved by a more structural approach termed the protocol analysis [Waterman and Newell 1971]. With protocol analysis, experts comment on specific examples from a problem domain. For example, experts may look at a specific design example and comment on the question "Why is the design good or bad?". This is different from the interviewing approach which may tackle the same problem by asking question patterns such as "What made the design good or bad?". Often, it is easier to comment on a specific example in the protocol analysis rather than to answer the general questions in the interviewing process. In protocol analysis, knowledge is extracted by detecting general patterns, e.g., experts may always examine one particular characteristic first.

Computer induction [Richie 1984] is another way to deal with knowledge acquisition. With induction, experts select a set of examples of cases (called training sets) together with attributes which are considered in design decision making. Then a program is applied to induce rules from these training sets. The quality of the induced knowledge depends on the selection of training sets, attributes, and the use of induction algorithms.

Much of the difficulty in knowledge elicitation lies in the fact that experts cannot easily describe how they view a problem. This is essentially a

psychological problem. Kelly viewed a human being as a scientist categorizing experiences and classifying them within his own environment. Such a description is very suitable for an expert in his knowledge domain. Based on Kelly's personal construct theory [Kelly 1955], the repertory grid technique [Boose 1988] was developed for knowledge acquisition. Given a problem domain, experts build a model consisting of elements and constructs which are considered relevant and important. The constructs are similar to attributes except that they must be bipolar, (e.g., good/bad, true/false, strong/weak). Elements are analogous to examples in induction. The grid is a cross- reference table between elements and constructs. For example, in acquiring knowledge for programming evaluation, experts may build the grid table with a number of typical programs (elements) and attributes (constructs) such as modularity, testability, portability, meaningful variables, and readable layout. Each square of the grid table is then filled with a quality value or index. Finally, the quality of the programs is then determined by experts based on the summing quality index.

Knowledge acquisition is a time-consuming and labor- intensive task. Although methodologies such as interviewing, protocol analysis, observation, induction, clustering, and prototyping [Kessel 1986, Gaines 1987, Olson and Rueter 1987] have been proposed to help elicit knowledge from experts, none of them is commonly accepted. Different problem domains may require different acquisition approaches for the best result. In recent years, several knowledge acquisition tools such as MOLE [Eshelman and McDermott 1986, Eshelman et. al. 1988], MORE [Kahn et. al. 1985], SALT [Marcus and McDermott 1986, 1989], and ETS [Boose 1985] have been developed to help acquire knowledge from human experts and to automate the construction of knowledge bases with the acquired knowledge.

3.2 Requirements of KAR

To reduce errors (e.g., misinterpretation of the interviewed data) caused by human intervention, a more efficient and reliable approach for acquiring knowledge is to automate the acquisition process based on a certain representation scheme which will completely and efficiently denote all the domain traits and encompass all the essential knowledge. Knowledge acquisition guided by the structure of a knowledge representation is termed Knowledge Acquisition based on Representation (KAR) [Hu and Rozenblit 1989]. To distinguish it from conventional approaches, KAR exploits the structural nature of a representation scheme to motivate acquisition activities.

For KAR, a knowledge representation scheme must satisfy the following requirements:

- The scheme must be able to initiate queries or question patterns automatically for knowledge acquisition based on its structural nature of representation.
- The scheme is associated with tools which will automatically translate the acquired knowledge into an internal form that is ready for inference. The internal representation of knowledge must be clear to domain experts and system users.
- The structural nature of the scheme must provide efficient and systematic mechanisms for knowledge organization, inference, and refinement.
- The knowledge representation scheme must possess axioms and operations to detect conflicting information, to point out that essential knowledge may be missing, and to eliminate repeated or unnecessary knowledge.

In conventional knowledge acquisition, tremendous labor is required to prepare question patterns, to set up personnel interviews, to verify essential knowledge from the interviewed data, to translate knowledge into the representation format, to organize knowledge into a knowledge base, and to validate the inference of knowledge. This labor-intensive manual process increases design overhead as well as development cost of a knowledge base. Instead of treating acquisition tasks separately and sequentially, KAR accomplishes the acquisition process in an integrated manner by using the automatic approach. With KAR, both design cost and cycle for the development of a knowledge base are reduced significantly.

3.3 KAR with FRASES

Although various schemes (see Section 2.1) have been introduced to help represent and manage knowledge, none of these conventional representation schemes satisfies all requirements that qualifies a representation scheme for the KAR approach. FRASES is a complete representation scheme for managing design knowledge, which conveys declarative knowledge (structure and attributes) as well as procedural knowledge (rules for design synthesis and performance tuning). With well-defined axioms and operations, FRASES satisfies all the requirements of the KAR approach as follows:

- The entity-based hierarchical structure of FRASES can be easily employed to represent design structures characterized by modularity, hierarchy, and regularity. By exploiting well-defined axioms and operations of FRASES, query rules for knowledge acquisition generated automatically.

- The acquired knowledge is automatically verified and translated into an internal form (i.e., EIF) that is ready for inference. This greatly reduces

human intervention. Due to the graphic interface of FRASES, knowledge represented with FRASES is clear to domain experts and system users.

- FRASES provides an efficient scheme for knowledge representation, refinement, and inference by partitioning the global knowledge base into hierarchical entity-based frame objects.

- Contradiction, duplication, missing or incompleteness of essential knowledge can be detected by applying verification rules associated with each query rule. In other words, axioms and operations of FRASES are verified on each query cycle to assure the consistency of knowledge.

3.4 Query Rules of FRASES

Conventionally, query patterns are embedded in the software of the knowledge acquisition tool. Users are not allowed to access and modify these predefined queries for other applications. A flexible knowledge acquisition tool should allow users to design their own query rules and customized explanations based on the characteristics of problem domains. In KAR with FRASES, query rules and explanation information are managed separately and stored in a user-accessible environment (e.g., a hard disk). By providing an appropriate directory path, users may define their own query rules and customized explanation in a disk file. During knowledge acquisition, appropriate question patterns are generated based on the interpretation of query rules defined in the KAR rule base.

3.4.1 Problem Domain Query (*-PD)

The first query in KAR with FRASES is to ask what the problem domain is. This acquired problem domain will become the root of a FRASES

tree. The root of a FRASES tree must have a node type "entity". This is automatically managed by the KAR tool.

3.4.2 Entity Query

During knowledge acquisition, if the focused node has a node type "entity", then query rules in this section will be applied to acquire essential information.

1. Querying Multiple Decomposition (E-MD): This query rule inquires if the number of the focused entity varies with design requirements. For example, performance constraints can be a factor that affects the number of computer modules in a distributed system.

2. Querying Design Attributes (E-ATTS): This query rule is used to acquire properties of the focused entity. FRASES classifies attributes of an entity into three categories: static attributes, design parameters, and performance indices (see Chapter 2). Query rules for acquiring attributes are:

- E-SATTS (querying static attributes): This rule queries general information about the entity (e.g., design version, date, etc.).
- E-PORTS (querying I/O ports): This rule acquires input/output ports for the behavioral model of the focused entity.
- E-DPARA (querying design parameters): This rule acquires design parameters required for the focused entity. Design parameters usually vary with design requirements. Various CAD tools, heuristic rules, or analytical models must be associated to calculate or estimate values of design parameters. It is important that the knowledge representation scheme is

capable of associating each design parameter with the desired CAD tools, analytic models, or heuristic functions. In FRASES, this is accomplished by frame demons. For each design parameter specified in E-DPARA query, users will be asked to provide either a value, a set of heuristic rules, an analytical model, or the entry to a CAD tool.

- E-PIX (querying performance index): This rule acquires performance indices that can be measured to evaluate the performance of the focused entity. For each performance index responded to an E-PIX query, users will be asked to give an atomic frame- based algebraic description about the performance index. Each performance index can be associated with estimate functions to provide a rough estimate for design pruning and/or for design refinement purposes.

- E-PP (querying processing priority): This rule is used to acquire processing priority between aspects and specializations under the focused entity. Considering the design of Local Area Networks (LAN) in Figure 3.1, if the "topology" (specialization) has a higher processing priority than the "element" (aspect), this will force the inference engine to select a topology (e.g., star, bus, or ring) before the "access-protocol" (e.g., token-passing or CSMA/CD) of a communication node; and the "medium" and "line-protocol" of each LAN segment are determined. In this case, a top-down design approach is implied. On the other hand, if the "element" (aspect) has a higher processing priority than the "topology" (specialization), then the inference engine will first select "access-protocol" for each communication node and determine the "medium" and "line-protocol" for each LAN segment. In this case, the LAN topology will be the last to be determined. This denotes a bottom-up design approach.

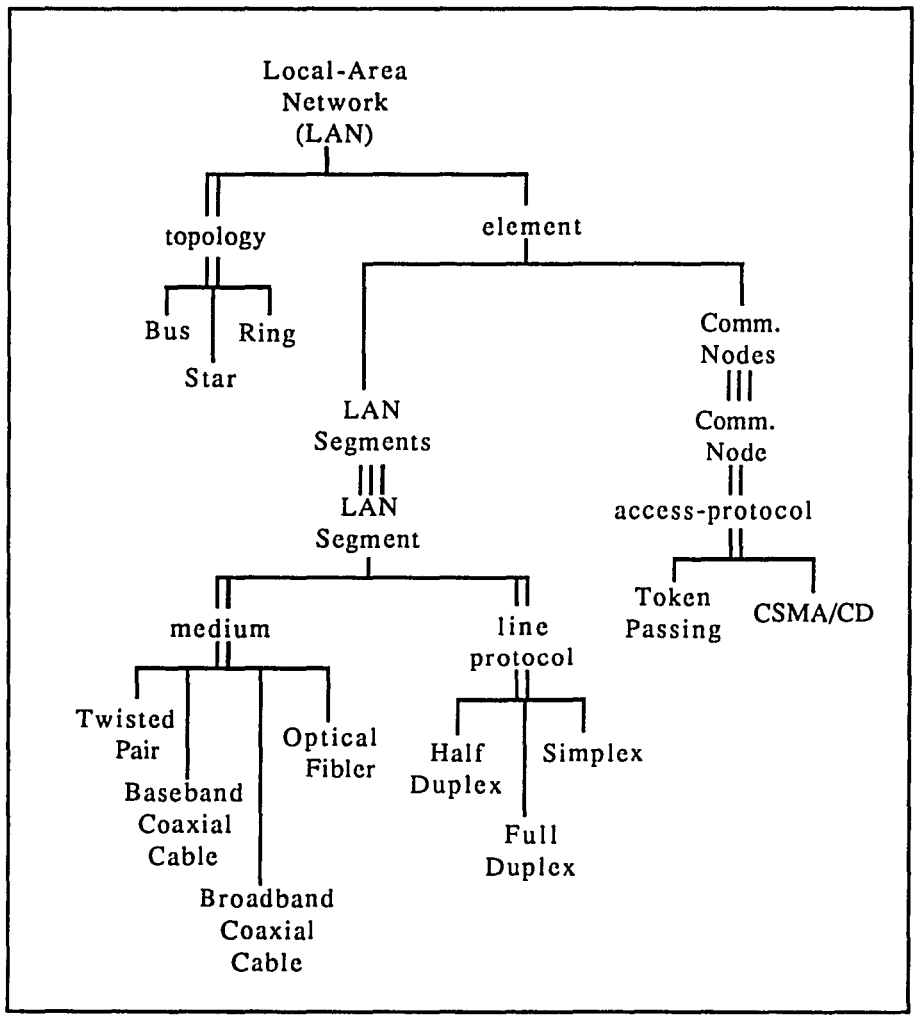


Figure 3.1 Local Area Networks (LAN) in FRASES

3. Querying Specializations (E-S): This rule is applied to acquire specializations from which the focused entity can be classified. For example, in the Local Area Network (LAN) design (Figure 3.1), a LAN can be classified into star, bus, and ring based on "topology".

4. Querying Aspects (E-A): This rule is applied to query aspects from which the focused entity is decomposed. For example, in Figure 3.1, LAN is decomposed into LAN segments and communication nodes based on "element".

3.4.3 Specialization Query

If the focused node has a node type "specialization", then query rules defined in this section must be applied for acquiring knowledge:

- S-E (querying design alternatives): This rule inquires specialization variants of the focused entity.
- S-SEL (querying selection rules): This rule is applied to inquire heuristics (production rules) for selecting a specialization variant.

3.4.4 Aspect Query

If the focused node has a node type "aspect", then query rules defined in this section must be applied for acquiring knowledge:

- A-E (querying subcomponents): This rule is designed to query subcomponents of an entity under a certain aspect.
- A-SYN (querying synthesis rules): This rule is used to acquire constraints (i.e., architecture, algorithm, or technology) for validating the configuration of a system.

- A-COUP (querying coupling information): This rule acquires port coupling among an entity and its children entities.

- A-DP (querying design priority): This rule is used to acquire design priority for each subcomponent of an entity. This information allows an inferencing engine to perform design reasoning in the right sequence and guarantees the crucial components to be designed first. For example, the CPU selection will affect memory configuration. Thus, CPU should have higher design priority than memory.

3.4.5 Model Query

After users are satisfied with the level of design abstraction, behavioral models of entities must be defined and integrated with the FRASES. Because of the dynamics of FRASES transformation, users may have a hard time finding out which leaf entity should have a corresponding model description. The KAR tool should identify each leaf entity that requires a behavioral model description. Querying model behavior is an application-dependent (i.e., depending on the simulator) task. For example, in DEVS- Scheme (Zeigler 1986), query rules for acquiring model behavior would be:

- M-EXT (querying external transition function)

- M-INT (querying internal transition function)

- M-TA (querying time advance function)

- M-OUT (querying output function)

Each query rule is associated with three explanation patterns (i.e., WHY, HOW, and WHAT) as follows:

- WHY is the question asked?
- WHAT does this question mean?
- HOW can this question be answered?

Users may design his own explanation for each query rule by adding an extension (e.g., "*.why*", "*.what*", and "*.how*"). For example, the S-SEL rule of a specialization node can be associated with explanation rules, termed S-SEL.WHY, S-SEL.WHAT, S-SEL.HOW, to explain "Why selection rules are required for a specialization node?", "What a selection rule means?", and "How to specify a selection rule?".

3.5 Verification of FRASES Queries

In order to assure the consistency of knowledge representation, axioms and operations of FRASES must be verified after each query rule is applied. Whenever conflicting or incomplete information is detected, error messages will be signaled to users so that an appropriate correction or modification can be made. For logic errors (e.g., conflicting information), the system explains which axiom or operation of FRASES has been violated and how these errors can be corrected. On the other hand, if a physical error (e.g., typos or missing an "IF" token in the rule specification) is detected, the system should first try to correct the error by referring to the explanation rules associated with the entity (i.e., S-¿SEL.HOW explains how to specify a selection rule). If the user is not satisfied with the system-made modification, he will be requested to enter a correct input.

3.5.1 Verifying Problem Domain Query

- Verifying Domain Existence (*-PD/VDE): If there exists an old definition, then this rule will ask if the old definition should be used, discard or saved.

All objects with identical names are pointed to the same frame record by the frame system. It is important to detect the existence of an old definition in the current environment in order to avoid undesired mixture of knowledge.

3.5.2 Verifying Entity Query

- Verifying Variation Range (E-MD/VVR): What is the maximum and minimum number for a multiple entity.

When the number of an entity varies with design requirements (e.g., answering a "yes" to a E-MD query), the system prompts a question to verify what is the maximum and minimum number allowed for the entity being designed. This information serves as a technical constraint for future design processing.

- Verifying Attached Variables (E-ATTS/VAV): No two variables have the same name.

This verification rule assures no two variables have the same name during the E-ATTS query. Since the frame system allows each slot to have multiple values, if an attribute is detected to appear twice or more times, all values (if different) will be confirmed and assigned to the same slot.

- Verifying Inherited Knowledge (E-ATTS/VIK): All properties inherited through a specialization (i.e., taxonomy) must be validated for the current entity.

This verification rule is applied only when the focused node has a parent with a node type "specialization". Only those properties that not contained in the focus node need be confirmed for inheritance.

- Verify Conflicting Knowledge (E-PORTS/VCK): This verification avoids contradiction in specification of port coupling.

After the E-PORTS query is executed, the definition of input/output ports is compared with the coupling information stored in the aspect parent of the focused node. To detect conflicts in coupling information, the KAR tool detects an undefined port name and the misuse of input ports as output ports or vice versa.

- Verifying Uniformity (E-S/E-A/VUF): Any two nodes with the same labels have identical Entity Information Frame and isomorphic substructures.

All frame objects with the same name are pointed to the same record in the frame system. If KAR tool detects the existence of the named node, the previous definition will be retrieved and displayed. If users agree to use the same definition, then no further queries on this node are necessary (uniformity axiom). This reduces the acquisition process by eliminating repeated queries. If users prefer a different definition for the named node, then the naming scheme of FRASES will be activated in order to give a new name that will break the pointer to its previous definition.

- Verifying Structure Hierarchy (E-S/E-A/VSH): No labels appear more than once down any path of the FRASES tree.

To verify strict hierarchy of a FRASES tree, ancestors of the focused node will be extracted and compared. The name of the focused node should not appear in its ancestors.

- Verifying Valid Siblings (E-S/E-A/VVS): No two entities under the same node have the same labels.

To verify valid siblings, KAR tool checks if any item appears more than once. This can be done by:

```
(find-if #'(lambda (e) (> e 1))
(mapcar #'(lambda (s) (count s siblings)) siblings ))
```

3.5.3 Verifying Specialization Query

- Verifying Uniformity (S-E/VUF): This rule is identical to E-S/E-A/VUF.
- Verifying Structure Hierarchy (S-E/VSH): This rule is identical to E-S/E-A/VSH.
- Verifying Valid Siblings (S-E/VVS): This rule is identical to E-S/E-A/VVS.
- Verifying Knowledge Combinations (S-E/VKC): Combinations of multiple specializations and/or aspects should be confirmed by the users.

To verify the combinations of specialization variants, the KAR tool checks if the focused node has other siblings with a node type "specialization". If there exists a specialization sibling, then all possible combinations of these specialization variants will be listed for confirmation. If the user points out that any one of the combinations is not allowed, constraints should be attached to their parent node.

- Verifying Transformed Knowledge (S-E/VTK): New knowledge transformed from the current structure of FRASES must be validated.

To verify transformed knowledge, the KAR tool checks if a transformation scheme (see Chapter 2) can be applied to the current FRASES. If the current FRASES can be transformed into a number of new structures, then each of these must be confirmed by the users.

- Verifying Pruned Knowledge (S-SEL/VPK): Design models generated from pruning specialization variants must be confirmed by the user.

After a S-SEL query is executed, selection rules should be tested with examples to assure that the inference engine performs the design reasoning properly. If users are not satisfied with the result, then rules must be refined. By reviewing the reasoning process, human experts can refine heuristic rules appropriately.

- Verifying Knowledge Hierarchy (S-SEL/VKH): Knowledge contained in an Entity Information Frame (EIF) is related only to itself and its substructures.

To verify strict hierarchy of a selection rule, the KAR tool checks if all the entities to be selected are children of the focused node (a specialization).

- Verify Conflicting Knowledge (S-SEL/VCK): To avoid contradiction and dead-ends in selection rules.

It is possible that two selection rules might generate conflicting information. For example, users may describe rules for selecting a processor as follows:

S1: If budget = low Then processor $\neq$ *Fast*

S2: If design-purpose = massive-computation Then processor = Fast

What happen if budget = low and design-purpose = massive-computation? To assure the reliability and flexibility of a knowledge base, rule conflicts should be detected and resolved during knowledge acquisition. Otherwise, conflicting resolution strategies [Winston 1984] must be integrated into the inference engine.

To detect dead-ends in selection rules, the KAR tool checks if the rule set causes none of the specialization variants to be selected. Assume that the "speed" in Figure 3.2 is associated with the selection rules as follows:

S1: If design-purpose = massive-computation Then processor = Fast

S2: If budget = low Then processor = Regular

These two rules may cause neither "Fast" nor "Regular" to be selected for "Processor" if "design-purpose $\neq$ massive-computation and budget $\neq$ low". In such a case, the KAR tool asks the user if any siblings of "Fast" (i.e., "Regular" in this case) can be selected when "design-purpose $\neq$ massive-computation"; and if any siblings of "Regular" (i.e., "Fast" in this case) can be selected when "budget $\neq$ low". This verification bridges the gap between rules by applying the concept of learning from hypotheses.

The KAR tool should also verify "If design-purpose $\neq$ massive-computation Then processor $\neq$ Fast" and "If budge $\neq$ low then processor $\neq$ Regular" are valid. For illustration, let us consider the following rule: *"If you have a meeting Then dress up"*. This rule does not mean *"If you do not have a meeting then do not dress up"*. To express this verification in a rule:

Given: *If A Then B*

Verify: *a.) if* $\sim A$ *then* $\sim B$ *b.) if* $\sim A$ *then* $(S - B)^*$

* (S-B): are siblings of B

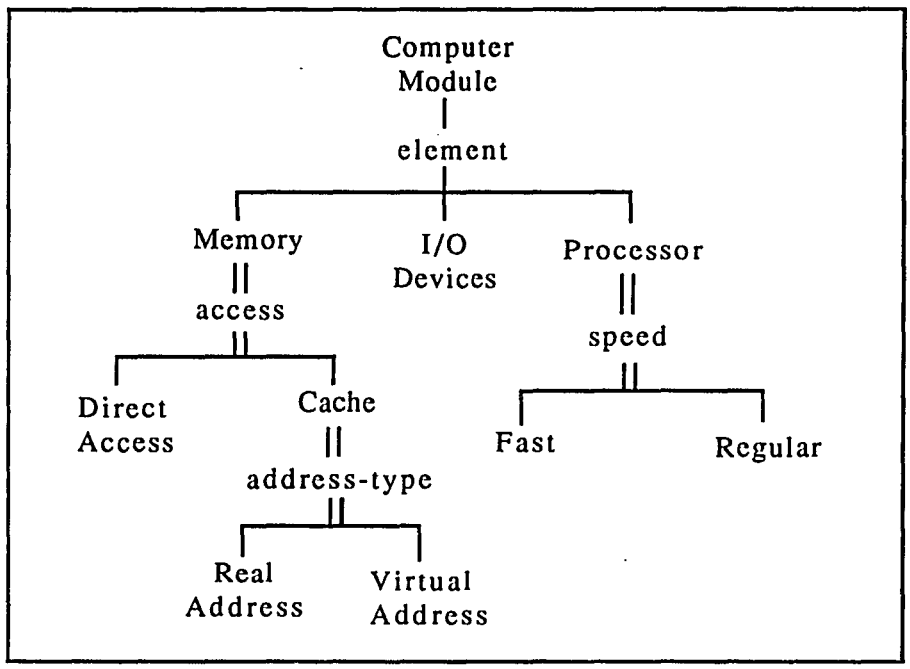


Figure 3.2 Computer Modules in FRASES

- Verify Concise Form (S-SEL/VCF): Rules with an identical action part should be combined into a concise form to reduce the complexity of the knowledge bases.

To verify if there exists concise forms for selection rules, the KAR tool checks the action part of each selection rule. If there exists multiple rules with an identical action (i.e., selecting the same entity), then the condition parts of these rules will be "OR" together to form a new concise rule. This is illustrated as follows:

S1: IF A and B THEN select X.

S2: IF C or D THEN select X.

⇒ *S-12: IF A and B or C or D THEN select X.*

3.5.4 Verifying Aspect Query

- Verifying Uniformity (A-E/VUF): This rule is identical to E-S/E-A/VUF.
- Verifying Structure Hierarchy (A-E/VSH): This rule is identical to E-S/E-A/VSH.
- Verifying Valid Siblings (A-E/VVS): This rule is identical to E-S/E-A/VVS.
- Verifying Concise Form (A-SYN/VCF): This rule is identical to S-SEL/VCF.
- Verifying Knowledge Hierarchy (A-SYN/VKH): Entities specified in a synthesis rule must exist in the substructures of the focused node.

For example, to optimize system performance in Figure 3.2, users may define the synthesis rule as follows:

"IF processor = fast THEN memory = cache."

To assure the strict hierarchy of knowledge organization, this synthesis rule must be kept in an Entity Information Frame (EIF) associated with the common parent of processor and memory (i.e., "element" in this case).

3.6 Query Process of KAR with FRASES

Selecting FRASES to conduct KAR, the query process can be depicted by the flow chart in Figure 3.3. As shown in the figure, KAR adopts top-down strategy of elicitation. The knowledge acquisition process starts from querying knowledge about the problem domain (*-PD). Once the problem domain is specified, the acquisition process falls into a *E–A–S* (Entity–Aspect–Specialization) loop. Component information and knowledge about determining design parameters are acquired at the "Entity" phase. Knowledge about functional decomposition, coupling, and synthesis is elicited in the "Aspect" phase. Knowledge about alternatives and selection rules is acquired in the "Specialization" phase. Experts may choose the most appropriate design approach (e.g., top-down, bottom-up, and mixed) based on the characteristics of design problems by indicating processing priority between "aspect" and "specialization" nodes. At each design level, the most crucial component is denoted by assigning the highest design priority. The *E–A–S* loop will continue until an appropriate level of abstraction is reached. At each acquisition state, appropriate verifications are automatically activated to avoid knowledge inconsistency.

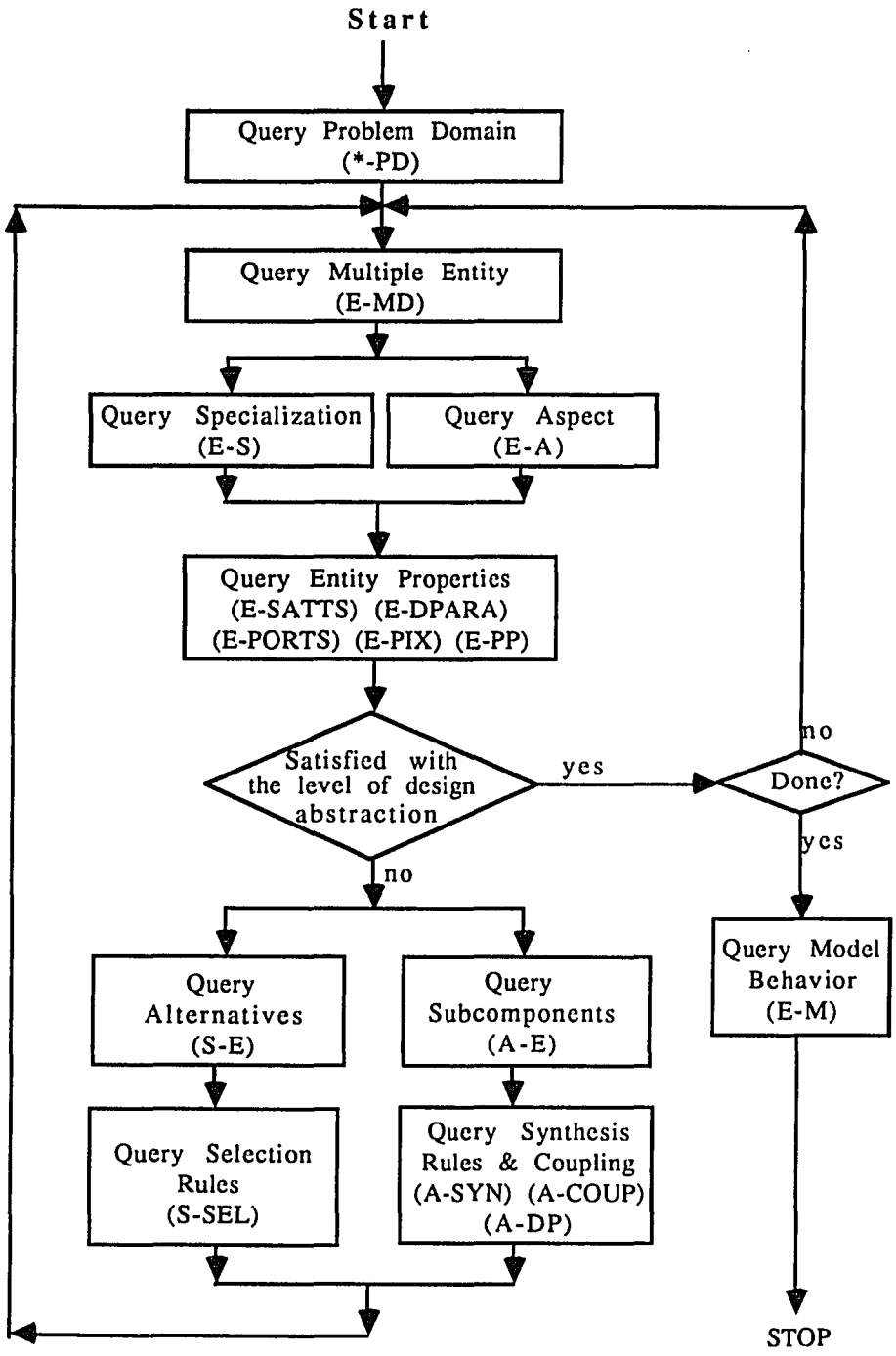


Figure 3.3 Query Transition of KAR with FRASES

3.6.1 Example of KAR with FRASES

To illustrate the operation of KAR with FRASES for knowledge acquisition, let us acquire knowledge for the design of an abstract distributed system.

Although many classifications and taxonomies of distributed systems were published in the last decade [Flynn 1966, Anderson and Jensen 1975, Enslow 1978], different views together with disagreement still exist, on what can be considered a distributed computing system. In the following discussion, distributed systems are categorized into three classes based on the *message transfer scheme* (i.e., via a local area network, an interconnection network, or a direct coupling).

The first class of distributed systems integrates computer modules with a Local Area Network (LAN). Each computer module in the network retains a strong local autonomy and devotes a very limited part of its processing power to common activities. Advantages offered by this type of distributed systems include: to share expensive resources such as laser printers and/or file servers in the network; to provide flexible system extension as the service demand grows; to increase system reliability by replicating data in different server computers; and to offer continued availability when one or more components are failed.

Users may choose different topologies (e.g., bus, ring, star) to organize computer modules of a LAN-based distributed system. To realize a LAN segment, users may choose from twisted pair wires, baseband/broadband coaxial cables, or fiber optics for transmission medium. To define the ability of a LAN segment to transmit and to receive data simultaneously, a LAN segment is classified into three types: simplex, half-duplex, or full-duplex. To synchronize the message passing between computer modules, message flow of each communication node is under the control of medium access protocols (e.g., token passing

or CSMA/CD [Stallings 1985, 1987]).

The second class of distributed systems organizes multiple computer modules with an Interconnection Network (IN). Each computer module is a fully programmable unit capable of executing its own programs. The amount of information exchange among computer module is significantly greater than in the previous case (computer network). The system expansion for this class is limited by the bandwidth of the Interconnection Network (IN). An Interconnection Network (IN) can be realized as a crossbar switches, single or multiple shared buses, or a Multistage Interconnection Network (MIN) [Hwang and Briggs 1984] consisting of digital switches.

The third class of distributed systems is composed of a large number of identical computing cells, each of which is able to perform a fixed and basic operation. These computing cells are interconnected according to a regular topology (e.g., star, ring, or cube), and work in parallel for the fast solution of CPU-bound algorithms (e.g., matrix, vector, or discrete Fourier transformation). A typical distributed system in this class is the Connection Machine [Hillis 1985].

To ease our future discussion, each computer module is decomposed into Memory, Processor, and I/O devices (a simplified case). There are two alternatives for the design of a memory system, Cache and Direct-Access (without cache). For cache implementation, users may choose either virtual address or real address to access the cache memory. With real address implementation, the size of a cache memory is limited to the page size of the system [Loo 1987].

Part of knowledge acquisition process using KAR with FRASES for design of distributed systems is illustrated in Figure 3.4. As shown in the figure, KAR automatically generates query patterns for acquiring knowledge, employs

KAR/EXPERT INTERACTION	FRASES CONVERSION
KAR> What is your problem domain ? => Distributed-Systems	Distributed Systems
KAR> What kind of static attributes do you want to add to the system? => (designer Jeff) (date 4/19/89) (place UA) KAR> What kind of design parameters will be considered at this design level? => cost size KAR> What kind of performance indices will be considered to evaluate a Distributed-System • • •	Distributed Systems EIF
KAR> Can you classify a Distributed-System based on certain specialization? =>why KAR> ->> This question is used to query how experts ->> classify the variants of the entity asked for. For ->> example, a computer network can be clssified ->> into RING, BUS, and TREE based on Topology. KAR> Can you classify a Distributed-System based on certain specializations? => nil KAR> Can you decompose a Distributed-System based on certain aspect? => module	Distributed Systems EIF module
KAR> What are these subcomponents when you decompose a Distributed-System based on Module? => Computer-Modules Message-Transfer-System (MTS)	Distributed Systems EIF module Computer Modules MTS
KAR> Does the number of Computing-Modules vary with design requirements? => yes KAR> Specify the range for the number of Computing-Modules? => 0 64 ;;; *multiple decomposition* • • •	Distributed Systems EIF module EIF Computer Modules MTS Computer Module

- *continued* -

KAR> Can you rank the design priority for Processor, Memory, and I/O-Devices?
=> Processor Memory I/O-Devices

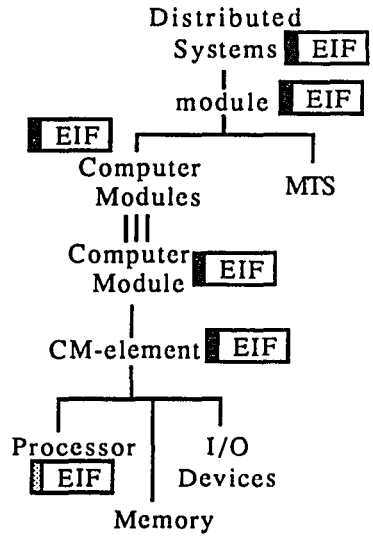
• • •

KAR> What kind of performance indices will be employed to evaluate a Computing-Module?
=> response-time thruput

• • •

KAR> Can you classify a processor based on certain specialization?
=> nil
KAR> Can you decompose a processor based on certain aspect?
=> nil
;;; satisfied with the level of design abstraction
;;; for processor

• • •



KAR> Can you classify a Message-Transfer-System based on certain specializations?
=> technology
KAR> What are the alternatives when you classify a Message-Transfer-System based on technology?
=> IN LAN
KAR> Can you specify selection rules that determine MTS technology?
=>r1: if interaction between computing modules is high or resources sharing capability is required then select LAN.
r2: if interaction between computing modules is low or resources sharing capability is not required then select IN.

• • •

KAR> Will the Distributed-Systems use a LAN or IN as the Message-Transfer-System (MTS) and consist of a variable number of computer modules?
=> yes *;;; verifying inheritance*

• • •

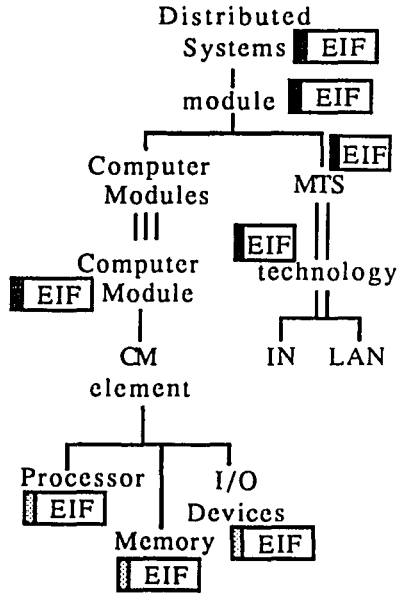


Figure 3.4 Illustration of KAR with FRASES

the built-in explanatory facility to help complete acquisition process, and transforms the verified information into EIF representation for future application. The resulted FRASES is shown in Figure 3.5. Entity Information Frames (EIF) corresponding to the FRASES (Figure 3.5) is attached in Appendix 3.1.

While laying out the FRASES structure, a user is allowed to expand his FRASES in both directions: vertical (levels of design abstraction) and horizontal (details of functional decomposition).

The most appropriate levels of design abstraction is determined by how the final system will be realized. Let us take the design of distributed systems as an example, if the system is going to be implemented with off-the-shelf products, then leaf entities of the FRASES would be commercial VLSI parts available on the IC market (e.g., realizing computer modules with Intel single board computers). On the other hand, if a user decides to optimize system performance by customizing the design of computer modules, the computer module will be decomposed into a number of functional modules. In this case, the original FRASES is extended by one level in vertical direction.

The design decisions involved to deal with the question "how to partition the system?" heavily influence the structure of the final product. FRASES allows various partitions to considered simultaneously by decomposing an entity under different aspects. Although a system may be partitioned in different way varied from expert to expert, criteria for common truth should be considered during decomposing a system (or subsystem) into functional modules. For example, the system should be decomposed in a way to minimize the amount of communication needed to perform a specific task and consequently obtain a maximum of local autonomy.

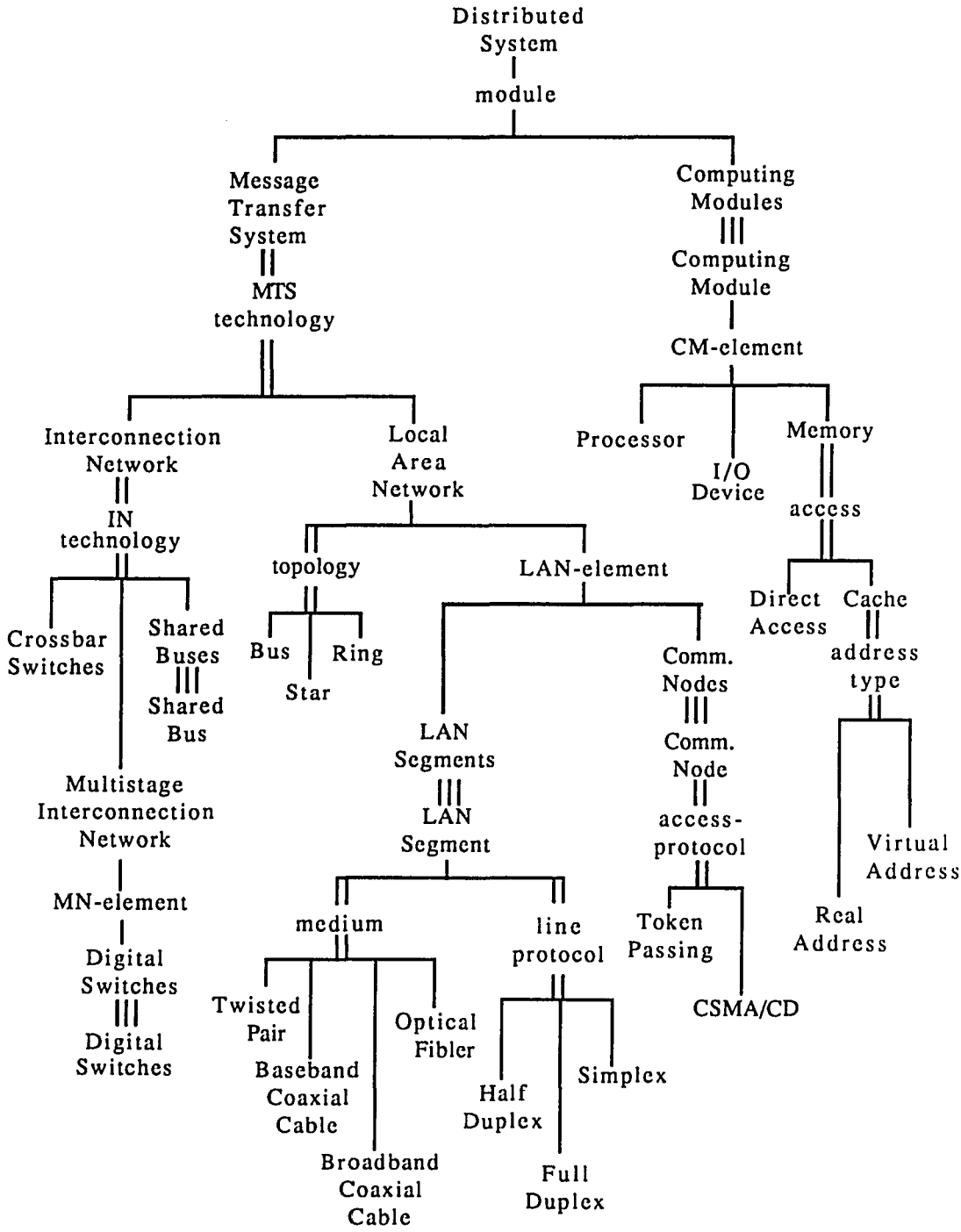


Figure 3.5 Distributed systems in FRASES

3.7 Advantages of KAR with FRASES

In Figure 3.6, a simple comparison between the conventional knowledge acquisition and KAR with FRASES is made. The diagram in the middle column delineates the acquisition process for developing a knowledge base. Several advantages are expected from using KAR with FRASES for knowledge acquisition:

- Efficiency: Question patterns necessary to acquire design knowledge for decomposition, taxonomy, pruning, and synthesis of systems are generated automatically. Knowledge provided by users is verified automatically by applying appropriate verification rules to assure consistency of knowledge. With FRASES, verified knowledge is translated directly into ready-for-inference Entity Information Frame (EIF) format. During knowledge acquisition, heuristic rules are validated with testing examples (see S-SEL/VPK). By examining results, experts can determine if knowledge refinement is required (unsatisfied case).

- Flexibility: One of the problems found in conventional knowledge acquisition methods is that they only appear efficient in a specific problem domain. With the flexibility of FRASES, KAR will fit other AI research applications.

- Manageability: The entity-based, hierarchical structure allows the knowledge represented to be examined and modified easily. During knowledge acquisition, users are allowed to traverse the FRASES tree and change the contents of Entity Information Frames (EIF).

Conventional Interview	Acquisition Phase	KAR with FRASES
Preparation of question patterns	START → Preparation	Automatic generation of question patterns based on associated query rules.
Interviewing and recording.	Elicitation	Expert/System interaction via user-friendly interface.
Detect duplicated and conflicting information manually.	Verification	Verification rules are automatically applied to detect conflicting and duplicate information.
Manually translate essential knowledge into required representation.	Translation	Essential knowledge is translated into EIF formats automatically.
Manually organize transformed knowledge into a knowledge base.	Organization	Knowledge is organized into a hierarchical, entity-based FRASES structure.
Validate knowledge via expert's verbal confirmation.	Validation → DONE	Knowledge inferencing is validated by practicing on-line design application.

Figure 3.6 Interviewing vs. KAR with FRASES

- Cost-Effectiveness: Unlike conventional acquisition approaches which require human intervention for labor-intensive preparation, interviewing, verification, translation, and organization, the complex process of knowledge base development is automated and efficiently handled. The fast turnaround of knowledge base development highly reduces the cost of knowledge-based systems.

CHAPTER FOUR

WOFIE - A WEIGHT-ORIENTED FRASES INFERENCE ENGINE

The inference engine is the procedure which generates consequences, conclusions or decisions from the existing knowledge. For instance, in production systems the inference scheme can be either a forward chaining or a backward chaining based on its control strategy. The forward chaining proceeds from the initial state toward the goal. On the other hand, backward chaining searches from the goal back to the initial state. Two major factors influence the question of whether it is better to reason forward or backward [Rich 1983]:

- The number of start and goal states: We would like to move from the smaller set of states to the larger set of states.
- Branching factor: Normally, the forward chaining is preferred in the problem domain where many facts lead to few conclusions. On the other hand, if each fact in the problem domain can lead to many conclusions, then the backward chaining scheme is more appropriate.

Decisions and/or conclusions are made from evaluating available knowledge. Given declarative knowledge, logical reasoning is performed simply with induction or deduction. When knowledge is incomplete or inexact, probabilistic reasoning is employed to predict the potential solution. In such a case, an appropriate probability model such as Bayes' Rule or fuzzy logic is used to determine the certainty factor or degree of belief. The design of an inference engine is influenced by its knowledge representation schemes and application domains. Before an inference engine is designed, designers, by examining the characteristics of problems, must determine the best knowledge representation scheme.

4.1 Complexity of an Inferencing Engine

In conventional knowledge-based systems, all the rules are placed in one linked list and all the true assertions are placed in another linked list. In the forward chaining method, the inference engine cycles through the rule list, checking at each rule whether all the premises are in the assertion list. If all the premises are found, the rule's conclusion is added to the assertion list. However, it is possible that a rule's conclusion may be a premise in a rule that precedes it. So, if the rules are only traversed once, some conclusions may not be deduced. Therefore, it is necessary for the forward chaining algorithm to return to the first rule each time a new conclusion is added to the assertion list. If N is the number of rules, the computation cost for the linked list forward chaining is $O(N^2)$ in the worst case.

On the other hand, in the backward chaining method, the inference engine chooses one of the final conclusions as a hypothesis and then attempts to prove it by proving all its premises that conclude the hypothesis. In order to prove each premise, the inference engine is forced to search through the rule linked list and finds rules that conclude the rule's premises. This process is repeated until the hypothesis is proven or failed. If N is the number of rules, the computation cost for the linked list backward chaining is $O(N^2)$ in the worst case.

According to the above analysis, the linked list data structure become inefficient as the number of rules increases. To overcome the computation cost problem, it is possible to store the rules in a graph (e.g., AND/OR tree) and reduce the worst case computation cost to $O(N)$. In most cases, computation cost for the graph-oriented inferencing should not increase proportional to N. This is due to the fact that conclusions may be made before all graph nodes are traversed. For example, rules are organized into a binary tree with "OR"

relation for sibling nodes. The computation cost for this binary tree organization becomes $O(log_2 N)$.

4.2 Problems of Knowledge-Based Design

The engineering design process can be regarded as a process of making decisions about a number of system components at different levels of design abstraction. With an AI knowledge-based approach, all the design heuristics (rules) will be organized into a knowledge base. The inference engine is then applied to derive design models based on interpreting the rules of the knowledge base. How the design knowledge should be organized, how the reasoning process should be performed, and how the design alternatives should be selected are all questions that pose problems for developing an inference engine for design application.

4.2.1 Organization of Design Knowledge

Engineering design is a knowledge–intensive task. To accomplish a small design may require a large number of rules. How to organize these rules appropriately to increase the efficiency of design reasoning is not a trivial problem. In order to derive a good rule base organization, let us first examine the relationship between rules. Normally, rules (heuristics) used to determine the selection of different components are independent of each other. For example, in the design of distributed systems, rules for selecting a LAN transmission medium are not related to rules for selecting a memory configuration of a computer module. It is unwise to put all the rules together because this will increase the search time for inferencing these rules. To optimize the performance of design reasoning, heuristic rules should be organized into distributed modules. Furthermore, the dependency between rule modules should be reduced to the minimum to

facilitate knowledge management. Since the linked list data structure is too inefficient to be considered, the graph-oriented data structures are preferred to organize rules in each module. On each rule module, an inference engine with either forward or backward chaining can be applied.

4.2.2 Sequence of Design Reasoning

Since the overall rules are partitioned into modules, a search procedure is required to determine the order in which rule modules are activated. Unlike other applications, engineering designs require components of a system to be designed in a particular sequence. Essential components are always determined before other components can be designed. The design sequence may be altered by environmental factors, problem domains, personal preferences, and technical constraints. This requires a flexible search scheme to conduct the design reasoning process in the right sequence. Application of wrong search algorithms may increase the design cost, or, in the worst case, produce an undesired design model.

Referring to the FRASES in Figure 3.5, let us assume that the user's goal is to design a LAN-based distributed system. According to the depth-first search, the first decision being made is to select an *Interconnection Network (IN)* from crossbar switches, multistage networks, or shared buses, which contradicts design objectives. Obviously, all the decisions made below this entity (*IN*) are unnecessary unless the *IN* is selected as the message transer system. Bad assumptions (i.e., *IN* is the desired message transfer system) in depth-first inferencing may cause the overall design reasoning to be inefficient.

Some of the problems in the depth-first inference are resolved by the breadth-first inference. According to the breadth-first search, the inference engine starts with selecting a *MTS technology* before any other decisions are made.

Unfortunately, a breadth-first inference always causes the *topology* of LAN to be determined before the *medium-access-protocol* is selected. Although there is no severe problems with this design sequence, it violates the design rule to optimize the performance of a LAN-oriented distributed system.

In other words, the design reasoning process with conventional searches such as depth-first and breadth-first [Winston 1984] requires the knowledge base (tree) to be organized into a particular order which may be impractical in real world application. In order to increase the efficiency of design reasoning, a flexible design inference engine should not restrict itself to pure top-down or bottom-up.

4.2.3 Generation of Design Models

Conventional inference engines derive at most one design model since only one path is expanded at each branching point. Theoretically, the best design model will be generated via expanding the most promising node at each branching point. However, this is not always guaranteed. For example, choosing faster memory may not increase the overall system performance at all. In some cases the supposed-to-be-the-best design model may not satisfy the performance requirements, thus force the system to derive another design alternative by repeating time consuming search procedures. This drastically increases overall design turnaround time. A more intelligent search allowing parallel generation of design alternatives is preferred.

4.3 Reasoning Process of WOFIE

To overcome the problems in developing a design inferencing engine, WOFIE partitions the overall rules into modules and organizes these rule modules into a

FRASES structure. Instead of a depth-first or a breadth-first search algorithm, a weight-oriented search is used to determine the order in which rule modules are activated. With the weight- oriented search, the most crucial component is assigned the highest processing priority. During design application, rules associated with the most weighted component will be traversed first. By appropriately setting up the priorities of specialization nodes, the weight-oriented search is capable of emulating the dynamic design process conducted by a human expert.

How the weight information is incorporated into FRASES entities is a task accomplished by assigning appropriate weights to entities during knowledge acquisition (E-PP and A-DP queries). Priority between different design levels is denoted by assigning different priorities between aspect and specialization nodes (E-PP query). Priority between components at the same design level is denoted by ranking priorities of components (A-DP query). If components of an entity should be designed first, then the aspect node is assigned a higher weight. This implies a bottom-up design methodology. On the other hand, if specialization variants of an entity must be selected before subcomponents are determined, then the specialization node will be assigned a higher weight. This implies a top-down design methodology. In a multilevel system design, processing priority between aspects and specializations can be assigned appropriately at each level of design abstraction to denote a hybrid design methodology.

On each inferencing cycle, instead of searching the whole rule base, only rules associated with the focused node are examined to select qualified alternatives. Unqualified entities and their subtrees are removed from the FRASES tree.

To enable parallel generation of design models, multiple selection is allowed at each branching point. All selected entities are marked by adding selection information to its static attribute slot. New assertions of a triggered

rule are stored in the appropriate frame objects. Instead of searching a global assertion list, all inquiries to inferred facts are accomplished by passing messages between frame objects. This reduces computation cost required to search for facts that match the "IF" parts of a rule.

After all rule modules are traversed, the pruned FRASES are converted into a number of composition trees by synthesizing components in a bottom-up manner. To validate the configuration at each design abstraction, synthesis rules (constraints) associated with aspect nodes are extracted and examined to remove invalid combinations of design models.

To illustrate the order in which rule modules are traversed, the weight-oriented inference algorithm is described as follows:

ALGORITHM: Weight-Oriented Inference
Form a queue (Q) with the root node;
repeat
N = (car Q);
if N.type = specialization
then
interpreting selection rules associated with N;
mark selected entities and cut off undesired subtrees;
endif;
NEXT = children of N;
NEXT = sort NEXT in order of decreasing weight;
Q = (append NEXT (cdr Q));
until Q is empty.

4.4 WOFIE vs. Other Inference Engines

To distinguish the advantages of WOFIE, design reasoning process performed on the same example by four different inferencing engines will be examined. They are:

- Linked list forward chaining.
- Linked list backward chaining.
- Depth-first FRASES inferencing engine.
- Weight-oriented FRASES inferencing engine.

Let us use the design of a computer module in Figure 3.2 as our example. Assume that a processor can be classified into a fast or a regular processor based on the speed (clock- rate) of the processor. If the computer module is designed for massive computation, then a "Fast" processor should be selected for implementation. On the other hand, if the computer module is designed for machine control, then a "Regular" processor is preferred to save design cost.

A memory system can be designed as a direct-access or a cache architecture based on the requirements of the system performance. To access a cache memory, users may choose from real-address and virtual-address. The size of a real-address cache is limited to its page size [Loo 1987]. Since the selection of the processor strongly affects memory design choices, the processor must be designed before a memory system is configured.

Figure 4.1 depicts the knowledge base for a linked list forward and backward chaining. As shown in the rule base, processing priorities of components are carefully incorporated into the rule structure. For example, in rule-7, "Processor is selected" is a premise of "Memory = Cache". This causes a processor to be designed before a memory system is selected.

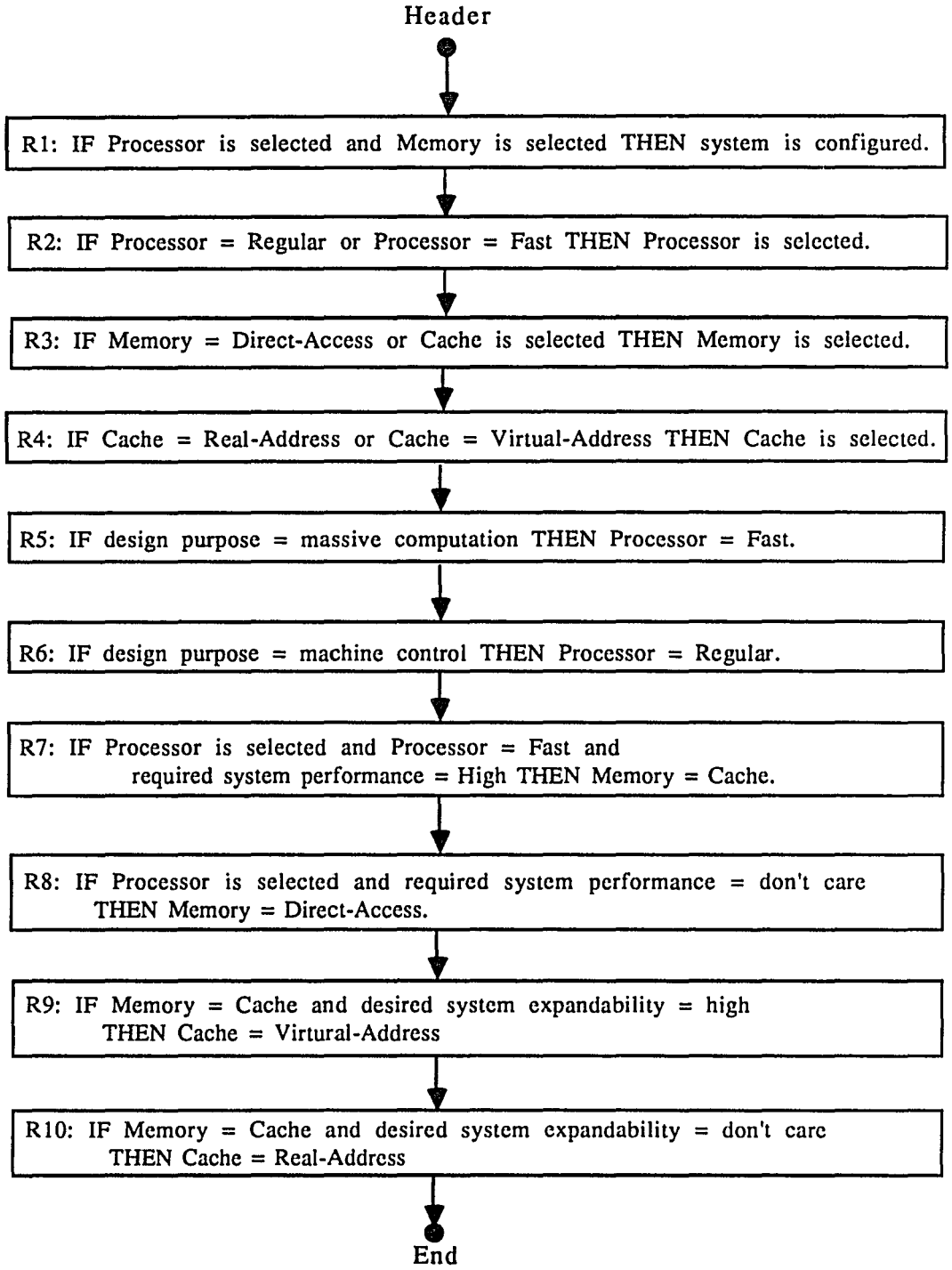


Figure 4.1 Rule list for global forward/backward chaining

With the linked list forward chaining, all the ten rules are loaded to the working memory and managed as a list. Given initial facts: a.) design purpose = massive computation; b.) desired system performance = high; c.) desired system expendability = high, the inference engine performs the design reasoning process as shown in Figure 4.2. As shown in the figure, the inference engine searches through the rule list and fires the rule-5. New fact (e.g., processor = fast) is then added to the assertion list. Since a new fact is generated, the inference engine goes back to the beginning of the rule list and tries to find out if any rules are fired because of the new fact. As shown in the figure, rule-2 is then fired. The search procedure will continue until the goal is reached (i.e., system is configured).

For linked list backward chaining, the inference engine explores the same rule base as shown in Figure 4.1, but derives the solution in a different direction. Again, all the rules are loaded into the working memory and managed as a list. On each inference cycle, the user is asked a question, if necessary, to provide information about unknown attributes in the "IF" part of a rule. The overall reasoning process is shown in Figure 4.3. As shown in the figure, the firing of rule-2 forces the inference engine to assume a regular processor is selected. This assumption is then proved incorrect by the firing of rule-6. This failed assumption forces the inference engine to go back to rule-2 and make another assumption about processor. In other words, a bad assumption in backward chaining increases computation overhead significantly.

Although graph approach reduces computation cost significantly, it poses some difficulties when it is applied to the system design reasoning. This is due to the complexity and dynamics of design problems. To assure an error-free design, essential components of a system must be designed in a particular sequence. This requires the design reasoning graph to be organized in a particular

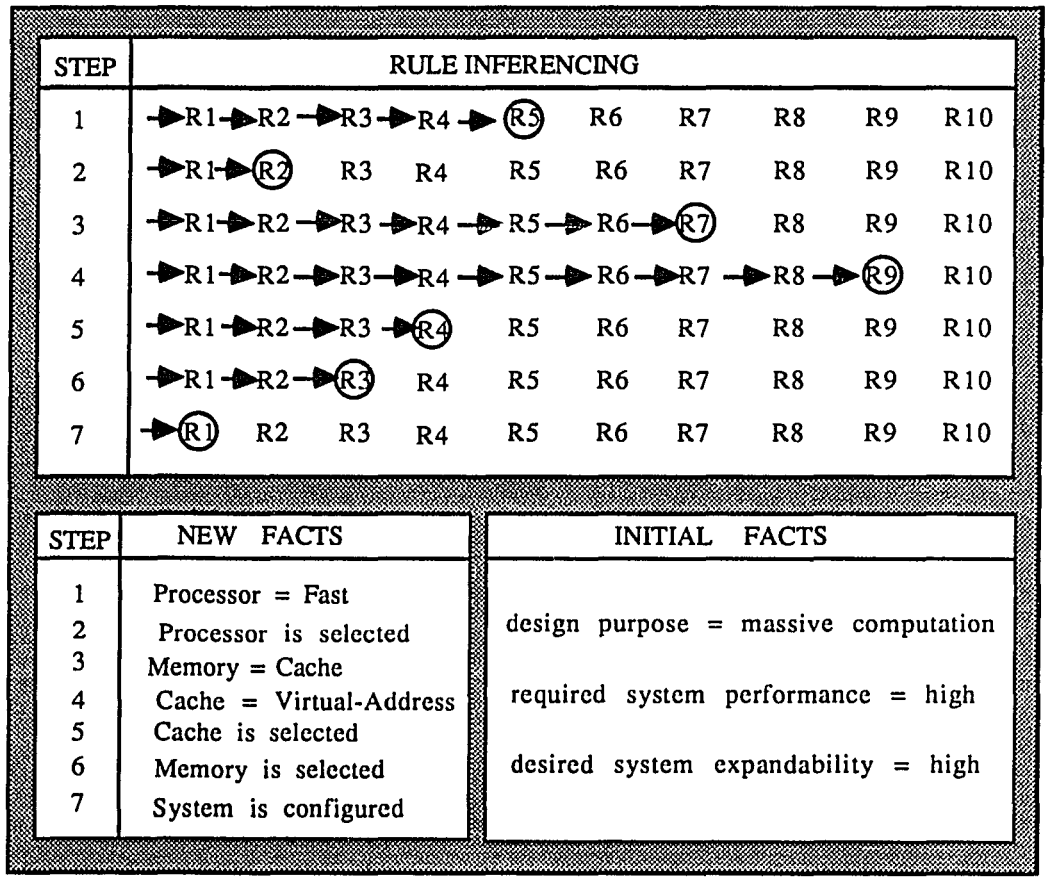


Figure 4.2 Design reasoning with forward chaining

STEP	RULE INFERENCING
1	(R1) R2 R3 R4 R5 R6 R7 R8 R9 R10
2	R1 → (R2) R3 R4 R5 R6 R7 R8 R9 R10
3	R1 → R2 → R3 → R4 → R5 → (R6) R7 R8 R9 R10
4	R1 → (R2) R3 R4 R5 R6 R7 R8 R9 R10
5	R1 → R2 → R3 → R4 → (R5) R6 R7 R8 R9 R10
6	(R1) R2 R3 R4 R5 R6 R7 R8 R9 R10
7	R1 → R2 → (R3) R4 R5 R6 R7 R8 R9 R10
8	R1 → R2 → R3 → R4 → R5 → R6 → R7 → (R8) R9 R10
9	R1 → R2 → (R3) R4 R5 R6 R7 R8 R9 R10
10	R1 → R2 → R3 → (R4) R5 R6 R7 R8 R9 R10
11	R1 → R2 → R3 → R4 → R5 → R6 → R7 → R8 → R9 → (R10)
12	R1 → R2 → R3 → (R4) R5 R6 R7 R8 R9 R10
13	R1 → R2 → R3 → R4 → R5 → R6 → R7 → R8 → (R9) R10

STEP	REASONING PROCESS
1	Assume "system is configured" and prove both processor is selected and memory is selected
2	Assume "processor is selected" and prove processor = regular
3	Assume "processor = regular" and prove design purpose = machine control; Assumption "processor = regular" is failed.
4	Assume "processor is selected" and prove processor = fast
5	Assume "processor = fast" and prove design purpose = massive computation. Assumption "processor is selected" in step-2 is proved.
6	Assume "system is configured" and prove memory is selected
7	Assume "memory is selected" and prove memory = direct-access
8	Assume "memory = direct-access" and prove required system performance is don't care. Assumption is failed.
9	Assume "memory is selected" and prove cache is selected
10	Assume "cache is selected" and prove cache = real-address
11	Assume "cache = real-address" and prove desired system expandability is don't care. Assumption is failed.
12	Assume "cache is selected" and prove cache = virtual-address
13	Assume "cache = virtual-address" and prove desired system expandability is high. Assumption is proved.

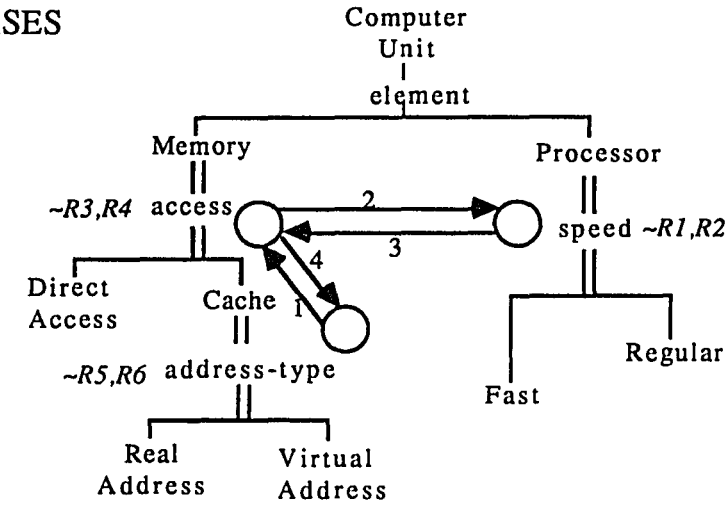
Figure 4.3 Design reasoning with backward chaining

pattern for correct inference. It becomes extremely difficult when the number of system components increases. To resolve the difficulty in building a global reasoning graph, the better approach is to partition the global reasoning graph into a number of subgraphs that can be managed by a graphic representation scheme (e.g., FRASES).

FRASES trees are different from inference trees. With FRASES, heuristic rules are distributed among specialization nodes (Figure 4.4a). Each specialization node has a small set of rules. Data structure for each rule set can be either a linked list or a graph. Both forward and backward chaining can be applied to interpret these rules. Design is completed when all rule modules are interpreted. Now, the major problem in the FRASES approach is to control the inference of rule modules in the right sequence.

With depth-first FRASES inference engine, the total number of rules (Figure 4.4b) is reduced from ten to six. In order to capture processing priorities appropriately, chaining between rule sets is made by specifying an extra premise in the "IF" part of a rule. For example, the premise *If processor is selected* in rule-3 and rule-4 shows the processor has higher priority than memory. In Figure 4.4a, the order in which rule sets are fired is indicated. The inference engine starts traversing the rule module of cache "address-type". Since the selection of memory "access" is a premise of of rule-5 and rule-6, the inference engine shifts the design reasoning task to memory "access". During interpreting the rules associated with "access", the premise (i.e., *If processor is selected*) of rule-3 and rule-4 indicates that the processor has higher design priority than memory. Hence, the rule set for processor "speed" is activated for selecting an appropriate processor. After Processor "speed" is determined, inference engine comes back to memory "access". Finally, the cache "address-type" is selected. The overall design reasoning is shown in Figure 4.4c.

(a) FRASES



(b) Rule base

R1: IF design purpose = massive computation THEN select Fast.
R2: IF design purpose = machine control THEN select Regular.
R3: IF Processor is sleected and required system performance = high THEN select Cache.
R4: IF Processor is selected and required system performance = don't care THEN select Direct-Access.
R5: IF Memory is selected and Cache is selected and desired system expandability = high THEN select Virtural-Address.
R6: IF Memory is selected and Cache is selected and desired system expandability = don't care THEN select Real-Address.

(C) Reasoning process

STEP	FIRING SEQUENCE	REASONING PROCESS
1	(R5) R6	Rule transition to prune memory
2	R5 R6 (R3) R4	Rule transition to prune processor
3	R5 R6 R3 R4 (R1) (R2)	if fast processor meets requirements if regular processor meets requirements
4	R5 R6 (R3) (R4)	if Cache meets requirement if Direct-Access meets requirements
5	(R5) (R6)	if Virtural-Address meets requirement if Real-Address meets requirements

Figure 4.4 Design reasoning with depth-first search on FRASES

Several drawbacks have been discovered in the depth–first FRASES approach:

- Processing priority must be incorporated into a rule structure. This, in turn, increases the complexity and development difficulty of a knowledge base. Besides, incorporating processing priority into rule structures may increase the overhead and complexity of design reasoning if the forward chaining is employed. In order to perform inference properly, the inference engine will have to duplicate the rule in multiple locations. For example, rule-3 and rule-4 associated with "access" in Figure 4.4a must be duplicated and linked to "speed". This increases the complexity of a rule base. Another approach to inference properly in forward chaining is to load the relevant rule sets into the working memory together with the focused rule set. For instance, instead of individual inference on "access" and "speed", rule sets of "access" and "speed" are integrated together and interpreted at the same time. This increases the inferencing overhead when the number of relevant modules increases.

- Inference engine may perform trivial works toward the desired system (e.g., reasoning on nodes that are not desired for the target system). Since the design sequence is driven by a depth–first search, the inference engine may start traversing a specialization node that is not related to the desired system. In this situation, all inference works are wasted. For example, even though the user's objective is to choose the "direct–access" for memory implementation, the inference engine still has to go over rule–5 and rule–6 which are not really necessary.

- For optimization, rule structures must be updated whenever the FRASES structure is changed. For example, if the processor subtree and the mem-

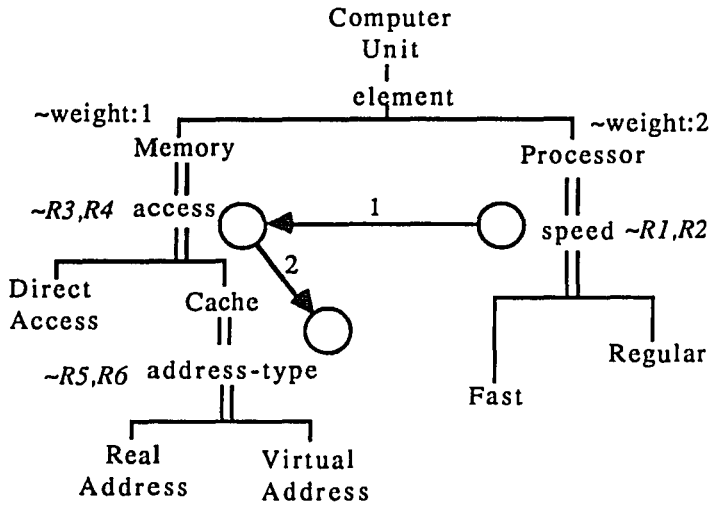
ory subtree (see Figure 4.4a) are switched, the processing priority between processor and memory is automatically implied (i.e., if the depth–first scheme is used). Thus, both rule–3 and rule–4 should be optimized by removing the premise *If processor is selected.*

- Knowledge management becomes difficult. Users are responsible to insert processing priority properly in order to assure an error-free design. Since the rule structure is affected by FRASES organization, any modification on the FRASES (e.g., add/remove/change/move an entity) may cause an extreme difficulty in updating all the related rules.

To improve above drawbacks, a Weight-Oriented FRASES Inference Engine (WOFIE) was developed. Based on the weight-oriented search, the order in which rule modules are fired is indicated in Figure 4.5a. Notice the difference between Figure 4.5a and Figure 4.4a. Since design priority for each node is indicated by experts during knowledge acquisition, there is no need to incorporate processing priority into rule structure as the approach shown in Figure 4.4b. The complexity of a rule base is highly reduced (Figure 4.5b). No matter how the FRASES is organized, design reasoning is always conducted in the right sequence. The rule structure is not affected by FRASES organization. Users need not worry about a broken chaining between rule modules. Knowledge refinement become much easier than other approaches (e.g., FRASES with depth-first search).

The overall design reasoning process is illustrated in Figure 4.5c. Firing sequence is properly directed by weights (processing priorities) associated with entities. In this example, only single specialization variant is selected. In fact, multiple specialization variants may be selected, which in turn, result in multiple design models. Synthesis constraints associated with decomposition

(a) FRASES



(b) Rule base

R1: IF design purpose = massive computation THEN select Fast.
R2: IF design purpose = machine control THEN select Regular.
R3: IF required system performance = high THEN select Cache.
R4: IF required system performance = don't care THEN select Direct-Access.
R5: IF desired system expandability = high THEN select Virtural-Address.
R6: IF desired system expandability = don't care THEN select Real-Address.

(C) Reasoning process

STEP	FIRING SEQUENCE	REASONING PROCESS
1	R1 R2	R1 is triggered
2	R3 R4	R3 is triggered
3	R5 R6	R5 is triggered

(d) Composition tree

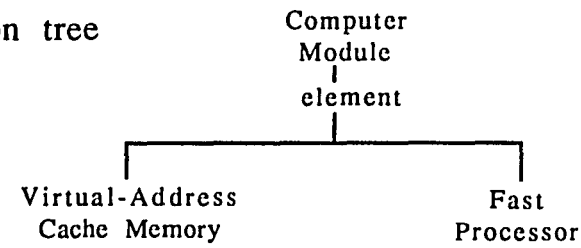


Figure 4.5 Design reasoning with WOFIE

nodes will be applied to validate each design combination. From above analysis, WOFIE is expected to have the following advantages:

- Allowing dynamic processing priority.
- Having simpler rule structure.
- Performing efficient reasoning.
- Facilitating knowledge management.

CHAPTER FIVE

AUTOMATIC GENERATION OF EXPERIMENTAL FRAMES

Performance evaluation is one of the most important tasks in system design modelling and simulation. A good design support system should assist in both design process and performance evaluation. In KBDSE, performance evaluation is accomplished by coupling experimental frames to the design model structure. Basically, an experimental frame specifies a limited set of circumstances under which a system is to be observed or subjected to experimentation. To facilitate performance analysis, experimental frames that correspond to the simulation requirements are programmed automatically. Whenever simulation requirements are changed, experimental frames are modified without human intervention. Automatic generation of experimental frames reduces the overall design cycle, relieves users from programming math-intensive models, and enables users to test the design model with various simulation circumstances (e.g., arrival processes, service processes, performance indices, and simulation controls). Generally speaking, automatic generation of experimental frames improves both the efficiency and the reliability for the simulation-based system design.

In this chapter a methodology for automatic generation of experimental frames using the concept of atomic frames will be presented. Although the proposed methodology is generic in nature, the DEVS-Scheme [Zeigler 1986, 1987] will be employed as the target simulator for explanation.

5.1 DEVS-Scheme Simulation Environment

The DEVS-Scheme is a knowledge-based discrete event simulation environment developed based on the AI object- oriented programming and the

multi-facetted modelling methodology [Zeigler 1984]. A knowledge-based simulation system encoding knowledge about simulation techniques may lessen the need for modelers to be experts in simulation programming, advises them on the selection of models for specific purposes, and interprets simulation results with expert statistical judgement. Several advantages distinguish the DEVS-Scheme from other simulation environments:

- Modularity: The DEVS-Scheme is implemented with AI object-oriented programming so that high maintainability and expendability are assured.
- Hierarchy: Corresponding to model structure, the DEVS simulation structure is implemented hierarchically with simulators and co-ordinators. The simulation structure can be partitioned into one or more substructures and mapped to a high speed distributed computing environment [Zhang and Zeigler 1989].
- Simplicity: Various function macros have been provided in DEVS-Scheme to facilitate the model construction.
- Flexibility and Visibility: Since model structures are accessible to run time modification, a flexible and high efficient run time improvement of the model structure is possible. Moreover, the DEVS-Scheme is associated with a visible window system so that the simulation process can be easily monitored from the screen.

5.1.1 Basic Structure of DEVS Models

DEVS-Scheme implements the set-theoretic formalism for discrete event models [Zeigler 1976, 1984]. To briefly review, a DEVS atomic model is defined by the structure:

$$M = < X,\ S,\ Y,\ \delta_i,\ \delta_{ex}, f, t >$$

where

X: is the set of external input events

S: is the set of sequential states

Y: is the set of output events

δ_i: is the internal transition function dictating state transitions due to internal events.

δ_{ex}: is the external transition function dictating state transition due to external input events.

f: is the output function

t: is the time-advance function

According to DEVS formalism, a coupled model is defined by specifying its component models and the coupling relations which establish the desired communication links.

The first step in developing a discrete event simulation is to build an abstract model of how the system being simulated works. An engineer must abstract those features and properties essential to the system's function and ignore the rest to keep the model to a manageable size.

To carry out the simulation of DEVS models, three types of processors are defined: simulators, co-ordinators, and root-co-ordinators. Simulators and co-ordinators are assigned to handle atomic-models and coupled-models in a one-to-one manner. A root-co-ordinator manages the overall simulation and is linked to the co-ordinator of the outermost coupled model. Simulation proceeds by means of messages passed among the processors which carry information concerning internal and external events, as well as data needed for synchronization.

5.2 Components of an Experimental Frame

The basic structure of an experimental frame is defined as a coupling of generators, transducers, and acceptors (Figure 5.1). The generator provides input segments to the design model. The transducer observes model outputs and calculates statistics for performance evaluation. The acceptor monitors simulation run and performs simulation control as requested.

5.2.1 Structure of a DEVS Generator

In discrete event simulation, a generator is capable of scheduling itself for a state transition, producing an output as a function of this state, and rescheduling itself for the next such iteration. Generators may be used to implement arrival processes among other classes of input segments to models. In other words, a generator can be used to imitate the dynamics of external events in the working environment. An output from a generator represents an event with the desired attributes such as the identity of the event and the workload of the event. Formally, a DEVS generator has the structure of:

$$GEN = < S,\ \delta_i,\ ta,\ Y,\ \phi >$$

where

S: set of sequential states

δ_i: Internal transition function

ta: time advance function

Y: set of outputs

ϕ: output function

Without external inputs, the internal transition function implies the state transition from the present state to the next state after certain amount

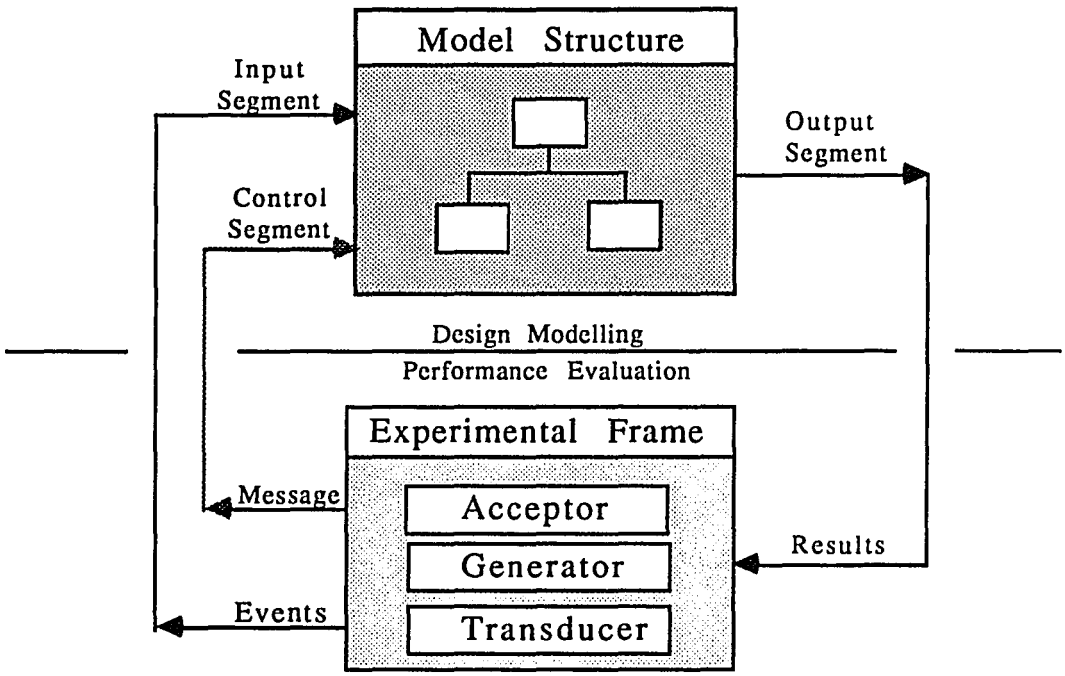


Figure 5.1 Structure of an experimental frame

of time. The time advance function indicates the time the system is allowed to stay for each state.

5.2.2 Structure of a DEVS Transducer

A transducer is a passive model with a designed initial state. When begun in such a state, the transducer maps its input segments into output segments. A DEVS transducer has a structure of:

$$TRAN = < X,\ S,\ \delta_{ex},\ Y,\ \phi,\ q_o >$$

where

X: is the set of inputs

S: is the set of sequential states

δ_{ex}: is the external transition function

Y: is the set of outputs

ϕ: is the output function

q_o: is the initial state

The time advance function need not be included in the specification since it is infinite. Transducers may be employed to gather statistics about model trajectories.

5.2.3 Structure of a DEVS Acceptor

An acceptor is a passive model with states partitioned into two sets, the accepting states and the non-accepting states [Zeigler 1984]. Formally, an acceptor is a structure of:

$$ACC = < X,\ S,\ \delta_{ex},\ q_o,\ F >$$

where

X: is the set of inputs

S: is the set of sequential states

δ_{ex}: is the external transition function

q_o: is the initial state, $q_o \in Q$ (*total state set*)

F: is the set of accepting states

The time advance function need not be included in the specification since it is infinite. Acceptors may be used to assure that model trajectories satisfy specified constraints.

5.3 Evolution of Experimental Frames

Rozenblit and Zeigler [Rozenblit and Zeigler 1986] proposed a scheme for formulating experimental frames based on the concept of system entity structures and generic observation frames. In that scheme, generic frames are initiated with model component names in a manner consistent with the model's I/O specification. This specification is derived from coupling constraints represented in the system entity structure.

Rozenblit has further extended the specification of simulation experiments by defining the distributed experimental frame architecture [Rozenblit 1985]. In his approach, a frame composition tree isomorphic with the model composition tree is defined. Thus, a hierarchical, multicomponent model is evaluated in a hierarchical frame where each atomic model has an atomic frame and each coupled model has a coupled experimental frame, respectively. The distributed experimental frame has been realized in the DEVS-Scheme environment [Duh 1988]. This realization is limited in that the modeler has to define

each experimental frame based on the modelling specification and check the frame/model coupling consistency, manually. Another limitation in Duh's implementation is that the modeler has to update experimental frames manually whenever simulation circumstances (e.g., arrival processes, service processes, performance indices and simulation controls) are changed. The system is not intelligent enough to alleviate the modeler's work. To evaluate a complicated design model, experimental frames usually increase the complexity in structure proportionally and require tremendous labor in programming such a complicated performance model. This motivates us to extend Rozenblit's distributed framework by providing methods for automatic generation of experimental frames [Rozenblit and Hu 1988, Hu and Rozenblit 1988].

5.4 Building Atomic Experimental Frames

To automate the generation of experimental frames, each experimental frame component (e.g., generator, transducer, or acceptor) is partitioned into a number of functional modules. Each functional module stands for a basic software segment which can be easily extracted from the data base or generated by a program automation algorithm. For identification, these functional modules are termed the atomic frames. In the concept of atomic frames, each experimental frame component is functionally partitioned into I/O modules and a number of atomic frames.

5.4.1 Atomic Generator Frames (AGF)

The main function of a generator is to provide input segments to the design model. In discrete event simulation, the trajectory of input segments is controlled by the arrival process and event formats. To enable the automatic

generation of a generator, users are requested to specify arrival process and event formats in terms of Atomic Generator Frames (AGF). Typical AGF are:

NUMBER	numerical random generator
SYMBOL	symbolic random generator
DISTRIBUTION	Exponential, Normal, etc.

NUMBER stands for a random number generator. To adapt application needs, users are allowed to set the upperbound and lowerbound of a NUMBER atomic frame. Besides, users may select the output of a NUMBER atomic frame to be either an integer or a real number. To request a NUMBER atomic frame, users may choose either one of the following procedure calls:

(NUMBER 100)	integer outputs with range [0,100]
(NUMBER 10.0)	real outputs with range [0.0,10.0]
(NUMBER -2 5)	integer outputs with range [-2,5]
(NUMBER -1.0 1)	real outputs with range [-1.0,1.0]

SYMBOL represents a random symbol generator. Users may control the output of a SYMBOL atomic frame by giving one or more indices as the header of symbols to be generated. To request a SYMBOL atomic frame, the following procedure calls are used:

(SYMBOL)	random symbols (e.g., xxx)
(SYMBOL 'CPU)	e.g., CPU-xxx, CPU-yyy, ..
(SYMBOL 'CPU 'QUEUE)	e.g., CPU-QUEUE-xxx, ..

DISTRIBUTION stands for different type of probability distribution functions. Each distribution function can be selected to emulate the arrival process and/or service process. To formulate desired distribution, users may adjust parameters such as mean, variance, sampling interval, and amplifying

factor. For example, to request a NORMAL distribution with mean 1, variance 20, and sampling interval 0.1, the following procedure is used:

(NORMAL :mean 1 :variance 20 :sample 0.1)

During application, the Automatic Frame Generator (AGF) first examines the Experiment Specification Form (ESF) and identifies all the atomic frames required to construct the generator. Frame messages are then sent to the frame system for requesting atomic frames. Each time an atomic frame is requested, the frame system creates an instance of the requested frame and assigns a unique name to the generated frame for identification.

To sample the next value of an Atomic Generator Frame (AGF), the following frame message is employed: *(FRAME-REPORT Atomic-Frame)*

Whenever a frame message is received, demons are activated to return the next value of the atomic frame.

5.4.2 Atomic Transducer Frames (ATF)

A transducer measures performance indices for a design model. In general, each Atomic Transducer Frame (ATF) is designed to perform a basic mathematic function. Typical ATF are:

Counter	counts the number of events
Repeater	counts the times of repeats
Sum	accumulates a series of numbers
Timer	accumulates the time elapsed
Time-Reader	reads the system clock
Maximum	computes the maximum value
Minimum	computes the minimum value
Mean	calculates the mean value
Median	calculates the median value

To enable the automatic generation of transducers for the measurements of performance indices, users are requested to express a performance index in terms of Atomic Transducer Frames (ATF).

An ATF can be regarded as an abstract gauge attached to a coupling port from which statistics related to a performance index is collected and computed. All the atomic frames are generic and independent of problem domains. In real application, generic atomic frames are extracted and aggregated into domain-specific experimental frames. The aggregated frames can be stored in the frame base for possible reuse in the future. In order to facilitate model retrieval and storage, experimental frames are organized into two levels as shown in Figure 5.2. At a generic level, the data base contains atomic frames from which domain specific frames are assembled.

During application, the Automatic Frame Generator (AFG) first examines the Design Specification Form (DSF) and identifies all performance indices interested for evaluation of the design model. Frame messages are then sent to the frame system to request a copy of each atomic frame. Each instance of the requested ATF has a unique name for identification. The newly created frame instance will inherit all properties including demons defined for the original frame class.

To access an Atomic Transducer Frame (ATF), the following frame message is used: *(FRAME-ACCESS Atomic-Frame Data)*

Each time the FRAME-ACCESS message is received, the "if-accessed" demon is activated to perform appropriate actions. For instance, the if-accessed demon of a counter responds to a FRAME-ACCESS message by incrementing its internal register.

To acquire the current value of an ATF, the following frame message is used: *(FRAME-REPORT Atomic-Frame)*

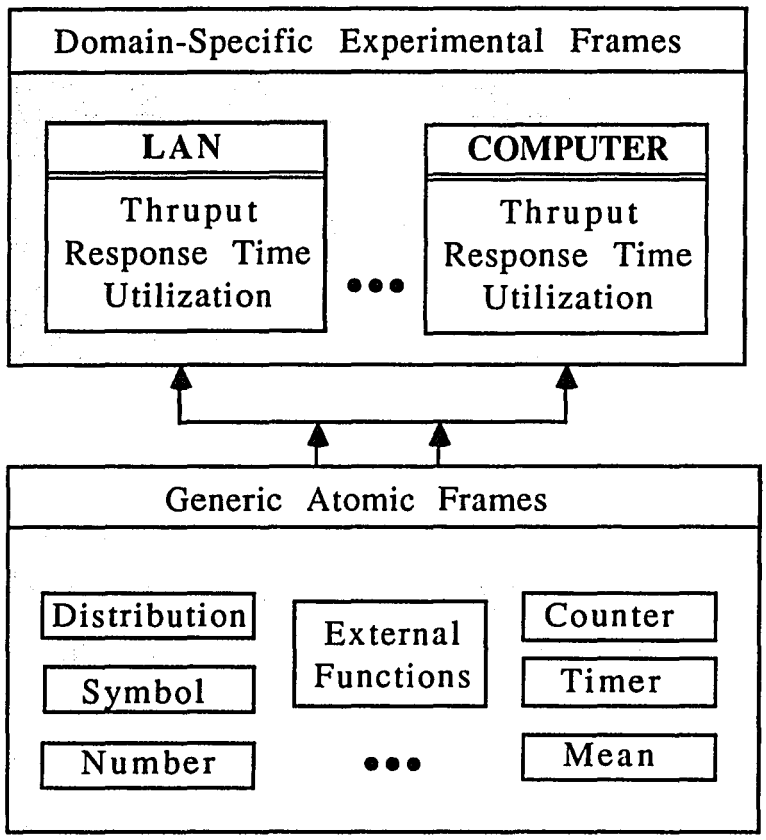


Figure 5.2 Organization of experimental frames

Whenever a FRAME-REPORT message is received, the "if-requested" demon reads the appropriate slot value of the called frame and return it to the calling procedure.

5.4.3 Atomic Acceptor Frames (AAF)

An acceptor acts like a supervisor that monitors the simulation run and controls simulation activities. Acceptors may use Atomic Transducer Frames (ATF) mentioned above to formulate the condition of an accepting state. For each accepting state, simulation controls will be issued to the corresponding models. The design objective is to develop control operations that are clear to users for specifying simulation controls. Available simulation controls for specifying an acceptor include:

STOP	permanently terminates simulation model format: (stop 'model-1 'model-2 ...) effects: sleep and no response to external events
DISABLE	temporarily terminates simulation models format: (disable 'model x-clocks) effects: disable model for x clocks
ENABLE	enables a disabled model format: (enable 'model-1 'model-2 ..) effects: enable disabled models
REPORT	reports one or more attributes format: (report 'model attribute) effects: no side effects
PAUSE	pauses simulation and stay in LISP shell format: (pause) effects: waits (return) to continue
INIT	initialize simulation models format: (init 'model-1 'model-2 ..) effects: state variables are changed

For a better understanding, the schematic structure of a frame-oriented generator, transducer, and acceptor is illustrated in Figure 5.3.

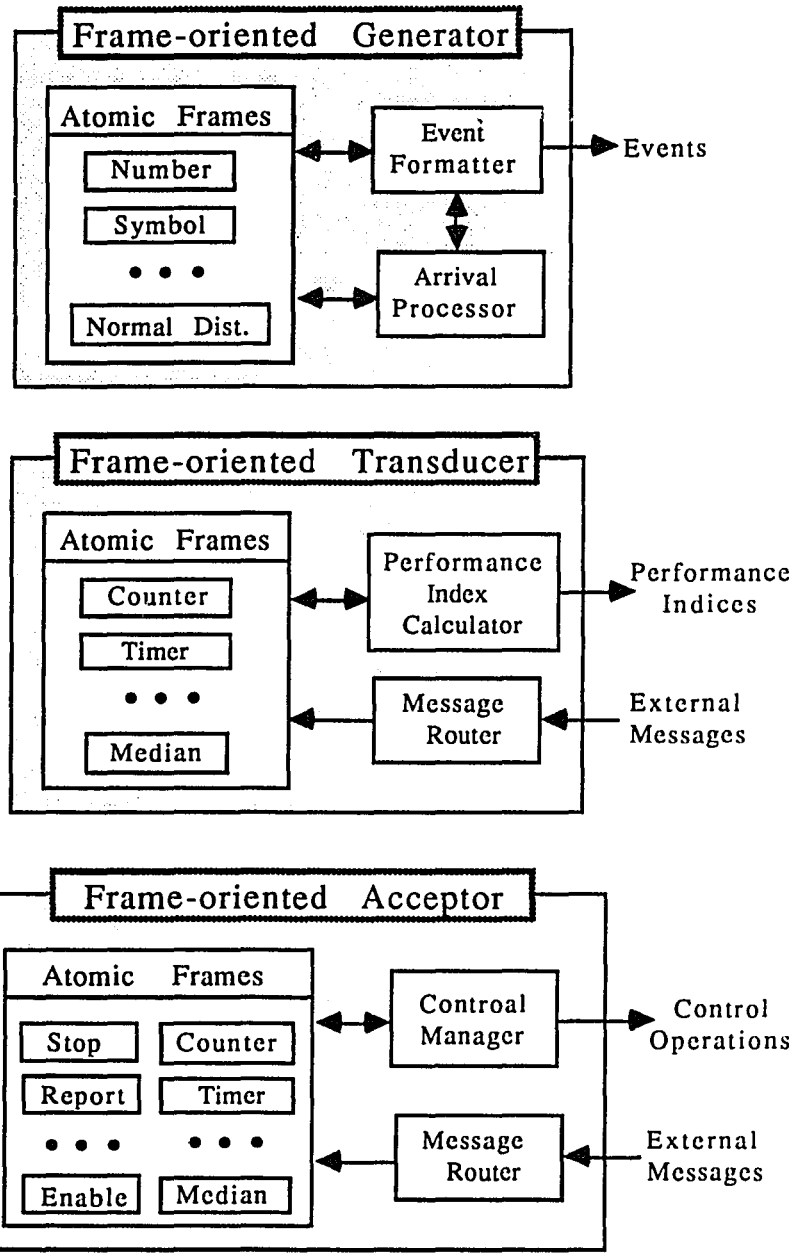


Figure 5.3 Schematic representation of experimental frames

5.5 Specification of Simulation Experiments

Before the embedded simulator is activated to evaluate the design model, simulation requirements must be specified. A typical discrete event simulation experiment is characterized by the following parameters:

1. *the arrival process.*
2. *the event format.*
3. *the performance indices.*
4. *the simulation control.*

With our design methodology, the design process is driven by system constraints and objectives. Performance indices interested are directly extracted from the Design Specification Form (DSF). For simulation purpose, only the simulation control (SC), arrival process (AP), and event format (EF) need to be defined.

Simulation Control (S.C.) enables the system to construct acceptors that issue appropriate controls such as signaling errors, reporting statistics, stopping simulation, restarting simulation, initializing models, and disabling/enabling models.

Arrival Process (A.P.) defines the inter-arrival time between events. Different probability distributions are supported to emulate various patterns of arrival processes. The specification of Arrival Process (A.P.) will enable the system to realize the time advance function of a generator.

Event Format (EF) defines the structure of an event. A typical event consists of multiple elements such as job identification, workload, and processing priority. The information of event formats will be exploited to construct the output function of a generator.

There are two ways to specify the simulation experiments: 1.) through the natural language or 2.) through the specification language. Here only the Experiment Specification Language (ESL) is discussed. To illustrate the grammar of ESL, the extended Backus-Naur Form (BNF) is used for macro expression. For reference, basic elements of an extended BNF representation are shown in Table 5.1 and the syntax grammar of ESL is listed in Table 5.2.

BNF	Meaning
::=	is defined by
|	or
{}	appears zero or more times
[]	appears one or more times
<>	refer to another syntax
"token"	symbol that must be matched exactly

Table 5.1. The extended BNF notation

<S.C.>	(cond [(<syntax> (<sim-act> {<value>}))])
<A.P.>	(cond [(<syntax> <p-dist>)])
<E.S.>	(cond [(<syntax> <arg-lst>)])
<SYNTAX>	{(<log-op> [(<rel-op> <index> <value>)])} | t
<REL-OP>	">" | "<" | "=" | ">=" | "<=" | "<>"
<ARG-LST>	("number" {real | integer}) | ("symbol" {symbol}) | <p-dist>
<SIM-ACT>	"stop" | "pause" | "init" | "report" | "enable" | "disable"
<INDEX>	"clock" | "event" | symbol | (<atf> symbol)
<ATF>	"counter" | "timer" | "time-reader" | "mean" | "maximum" | "minimum" | "median"
<P-DIST>	(<d-fun> {":mean" number} {":amp" number} {":variance" number} {":sample" number}
<D-FUN>	"normal" | "exponential" | "poisson"
<VALUE>	symbol | number

Table 5.2 Grammar of ESL

5.6 Constructing a DEVS Experimental Frame

The first step toward the automatic generation of experimental frames is to collect essential information from the Design Specification Form (DSF) and the Experiment Specification Form (ESF). For example, from DSF all performance indices needed are identified. The performance indices together with the arrival processes, event format, and simulation controls are used as generic information for constructing experimental frames. Atomic frames related to simulation parameters are then extracted and aggregated together to form the desired experimental frame.

To adapt to different simulation environments, language-dependent program generators must be attached to construct simulation models. In other word, the program automation algorithm in the Automatic Frame Generator (AFG) is a simulation language-dependent module. Multiple program automation algorithms are required if the system supports multiple simulation languages. For illustration, the design flow for automatic construction of experimental frames is shown in Figure 5.4.

According to our design methodology, an experimental frame is implemented as a coupling of generators, transducers, and acceptors. The automatic generation of an experimental frame can be divided into four subtasks as follows:

1. *Generation of generators*
2. *Generation of transducers*
3. *Generation of acceptors*
4. *Coupling of frame components*

To illustrate the automatic generation of experimental frames, the DEVS-Scheme simulation environment [Zeigler 1986] is employed as our simulation media. With DEVS formalism, a simulation model is composed of external

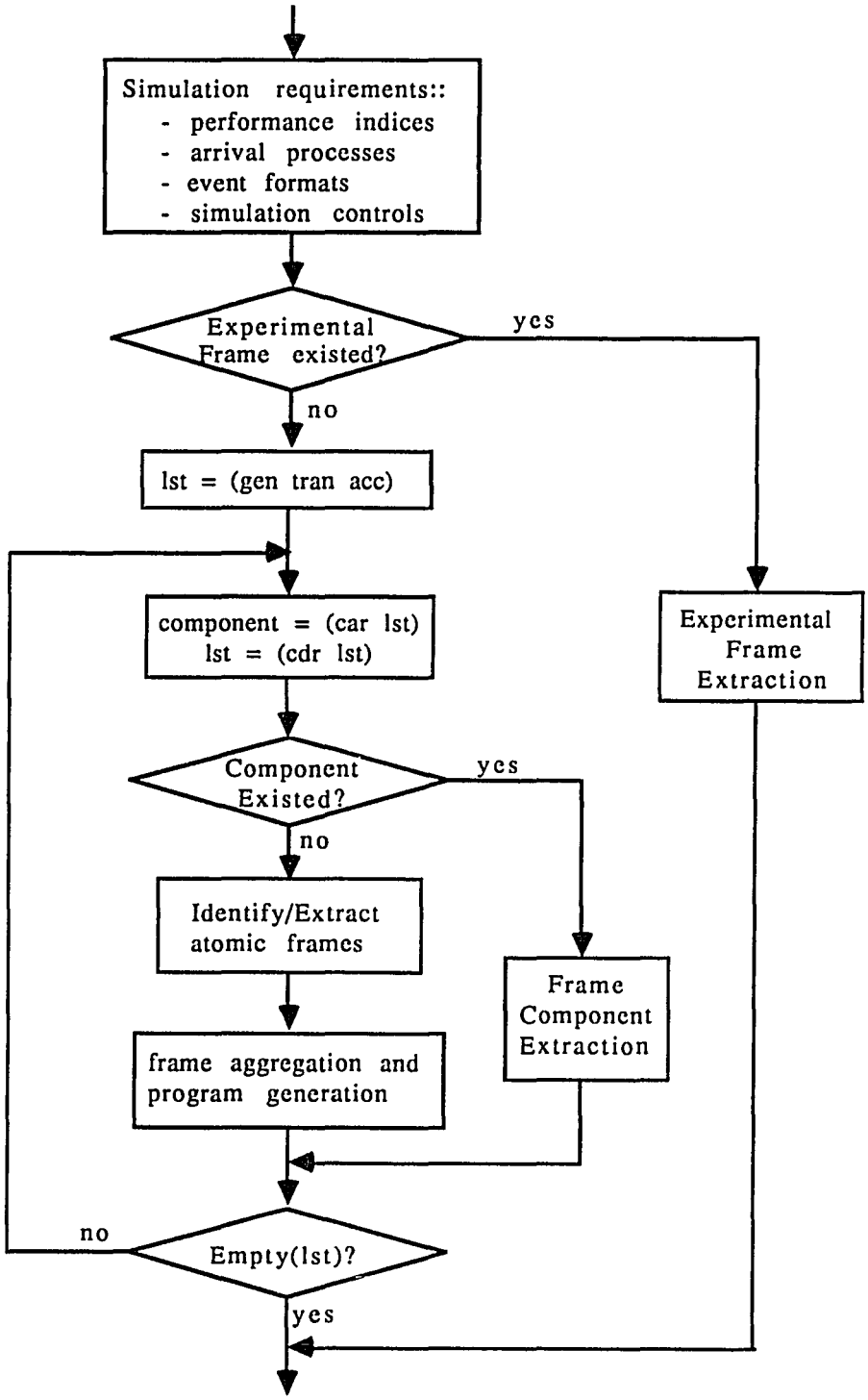


Figure 5.4 Flow of experimental frame generation

transition functions, internal transition functions, time advanced functions, and output functions. To automatically generate an experimental frame for the DEVS-Scheme simulator, the system will have to refer to DEVS-Scheme syntax grammars, analyze generic simulation parameters, and extract appropriate atomic frames for realizing DEVS transition functions.

5.6.1 Constructing a DEVS Generator

A DEVS generator consists of a time-advance function to emulate arrival processes, an internal transition function to control state transition, and an output function to generate events as requested. For implementation, two variables: *EVENT* and *CLOCK*, are reserved to indicate the current system clock and the number of events generated. Users may use *EVENT* and *CLOCK* to schedule events with a different arrival process or event format.

To construct the time-advance function, the arrival process specified in the Experiment Specification Form (ESF) is examined to identify atomic frames for generators. Each requested atomic frame is generated by the frame system with a unique name. Appropriate frame messages for sampling values will be added to control the interarrival time. There are only two states in a frame-oriented generator: active and passive. The frame-oriented generator will stay in an active state until it is notified to stop (e.g., a passive state). For output function, appropriate frame messages are generated to report values of atomic frames for outputs. Every time an output event is generated, the reserved variable, *EVENT*, is incremented by 1.

5.6.2 Constructing a DEVS Transducer

The first step toward the generation of a DEVS transducer is to identify performance indices from the Design Specification Form (DSF). The Perfor-

mance Index Tree (PIT) associated with each performance index is referred to guide the aggregation process of a Frame-Oriented Transducer (FOT). In fact, the desired transducer could be either extracted from the database or generated by aggregating atomic frames, depending on the existence of the transducer in the current experimental frame base.

The Performance Index Tree (PIT) is a semantic structure which indicates the algebraic relation between the performance index and its relevant atomic frames. In other words, the information of PIT provides information about how atomic frames will be aggregated to form the corresponding performance index. For example, a typical performance index tree for measuring the utilization of a Local Area Network (LAN) can be represented as the semantic tree in Figure 5.5. As shown in the figure, each leaf node of a Performance Index Tree (PIT) can be either an atomic frame or a state variable defined in the simulation model.

Automatic generation of transducers provides an efficient way for dealing with performance evaluation in system design modelling and simulation. For example, performance indices can be dynamically changed without worrying about reprogramming and debugging the math-intensive performance evaluation models. The automation algorithm for generating a Frame-Oriented Transducer (FOT) involves the following steps:

1. *Identification of performance indices*
2. *Extraction and Aggregation of Atomic Frames*

The external transition function of a DEVS transducer can be regarded as a message router. The main function performed by this message router (or external transition function) is to propagate messages to the appropriate atomic frames. Whenever a frame is accessed, demons are activated to perform a basic mathematic calculation.

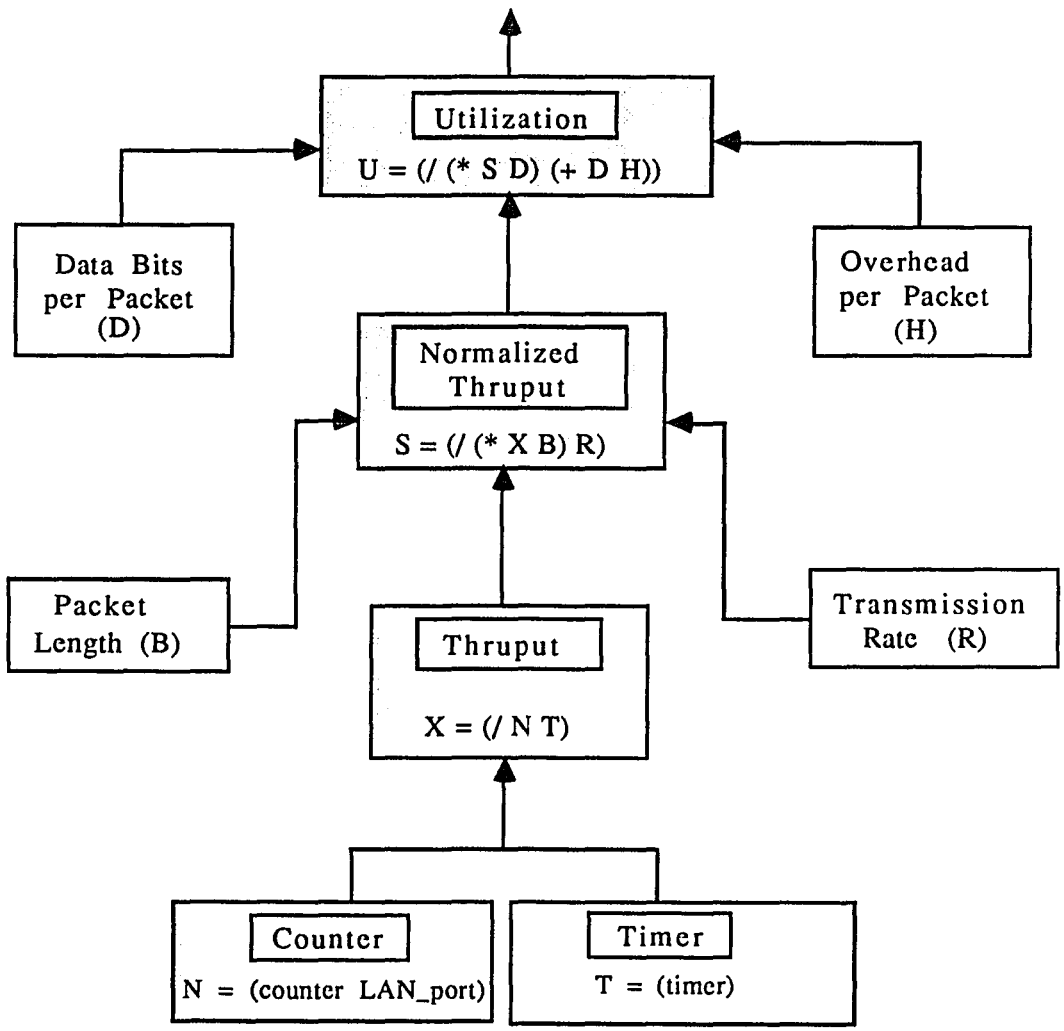


Figure 5.5 Performance index tree of LAN utilization

The internal transition function is implemented as a passivate function. In other words, a frame-oriented transducer stays in the passive state until an external transition occurs. This internal transition function also provides a by-pass connection from external transition to output transition. The output function of a frame-oriented DEVS transducer is designed to calculate and report the performance index measured. All components (e.g., state variables or atomic frames) of a PIT are acquired and computed to derive the value of the performance index.

5.6.3 Constructing a DEVS Acceptor

A DEVS acceptor performs simulation controls such as initializing model states, initiating/terminating simulation, and monitoring/reporting simulation status. On receiving each external message, the acceptor checks the current status to see if it falls into one of the accepting states. If one of the accepting states is found, its associated control actions will be executed. For implementation, the Automatic Frame Generator (AFG) examines the Simulation Control (SC) specified in the Experiment Specification Form (ESF) and identifies what kind of atomic frames have been employed to formulate accepting states. Instances of these atomic frames are then generated by the frame system. In addition, control operations associated with each accepting state are reformed into DEVS messages to invoke appropriate methods of DEVS atomic models.

A frame-oriented DEVS acceptor is designed to have only external transition function. All simulation controls are issued by message passing instead of physical port coupling. This reduces the communication overhead and complexity of experimental frames for DEVS simulation.

Basically, the external transition function first routes external messages to appropriate atomic frames and resets state variables if necessary. The accepting/rejecting status is then checked to see if control operations would be executed.

5.7 Advantages of Frame Automation

To guarantee an error-free design, the design model is usually tested under various conditions and evaluated with different performance indices. With the conventional (manual) approach, designers are always annoyed by reprogramming and debugging math-intensive performance models whenever simulation parameters are changed. For instance, selecting a different performance index requires a new transducer; changing arrival processes or event formats requires remodelling of generators; updating simulation controls requires modification in acceptors. Generally speaking, manual approach for performance modeling always increases design cycle because of the following factors:

1. *human think time*
2. *modelling capability*
3. *typing and I/O speed*
4. *acknowledgement of system*
5. *friendliness of user interface*

The objective in automating the generation of experimental frames is to provide a flexible and efficient way for dealing with performance analysis. Users are allowed to experiment with design models by using different combination of simulation parameters without wasting time and labor in developing math-intensive performance models. With automatic program generation, software problems caused by human intervention are reduced to the minimum.

CHAPTER SIX

INTEGRATED KNOWLEDGE-BASED DESIGN METHODOLOGY

Modern engineering design is a highly complex creative process. Given certain design specifications involving conflicting objectives and constraints, engineering designers must be able to consider a wide range of possible configurations. To assure an efficient and reliable design, it is necessary to incorporate the knowledge-directed framework into the design process. Because of its knowledge-intensive characteristics, engineering design is a particularly good domain for the application of AI knowledge-based systems. In recent years, expert systems have been introduced to cope with the increasing difficulty of engineering design. By using expert systems, an expert's highly valuable knowledge is duplicated and encapsulated as software that can be spread widely. With this built-in knowledge, expert systems (or perhaps more properly, known as knowledge-based systems) are able to reason like an expert's brain, at least on a limited scale. To construct computer programs that capture a significant amount of an engineer's knowledge is the goal of Intelligent Computer-Aided Engineering (ICAE) [Forbus 1988]. Expert systems, a branch of Artificial Intelligence (AI), has the major concern to make computers more intelligent. With the aid of AI techniques, both quantity and quality of engineering design productivity can be highly improved.

6.1 Basic Architecture of an Expert System

Generally speaking, each knowledge-based system is developed to solve only a specific problem domain. Most expert systems consist of a knowledge base and an inference engine. The knowledge base provides various facts, rules, and heuristics about objects and how these objects are interrelated. The inference

engine is the mechanism designed to mimic a human expert's reasoning process based on the information stored in the knowledge base. Usually, an intelligent, user-friendly interface (i.e., a natural language processor) is used to provide communication between the expert system and users. The explanatory facility explains HOW, WHY, or WHAT decision has been made. At any point, users are allowed to ask how the system reached an intermediate conclusion or why the system needs the particular information requested. Some expert systems may also include a knowledge acquisition tool to facilitate the development and refinement of a knowledge base. In addition, one or more database systems may be incorporated to provide data required for processing. The basic structure of an expert system is depicted in Figure 6.1. The following sections will review the major concerns in developing an expert system. These general concepts will provide background knowledge for understanding future discussion.

6.2 Knowledge-Based System Design

The term system design will denote in our framework the use of modeling and simulation techniques to build and evaluate models of the system being designed.

Our research aims to develop and implement a methodology of design in which design models can be synthesized and tested within a number of objectives, requirements, and constraints. This framework, termed knowledge-based system design and simulation, is presented in detail in [Rozenblit and Zeigler 1985, 1988, Rozenblit 1985]. The design objectives (understood here in a broader context that includes requirements and constraints of the design process) drive two fundamental processes in the methodology: first, they facilitate the construction, retrieval, and manipulation of design entity structure [Rozenblit and Zeigler 1986, 1988]. Secondly, the objectives serve as a basis of the specification

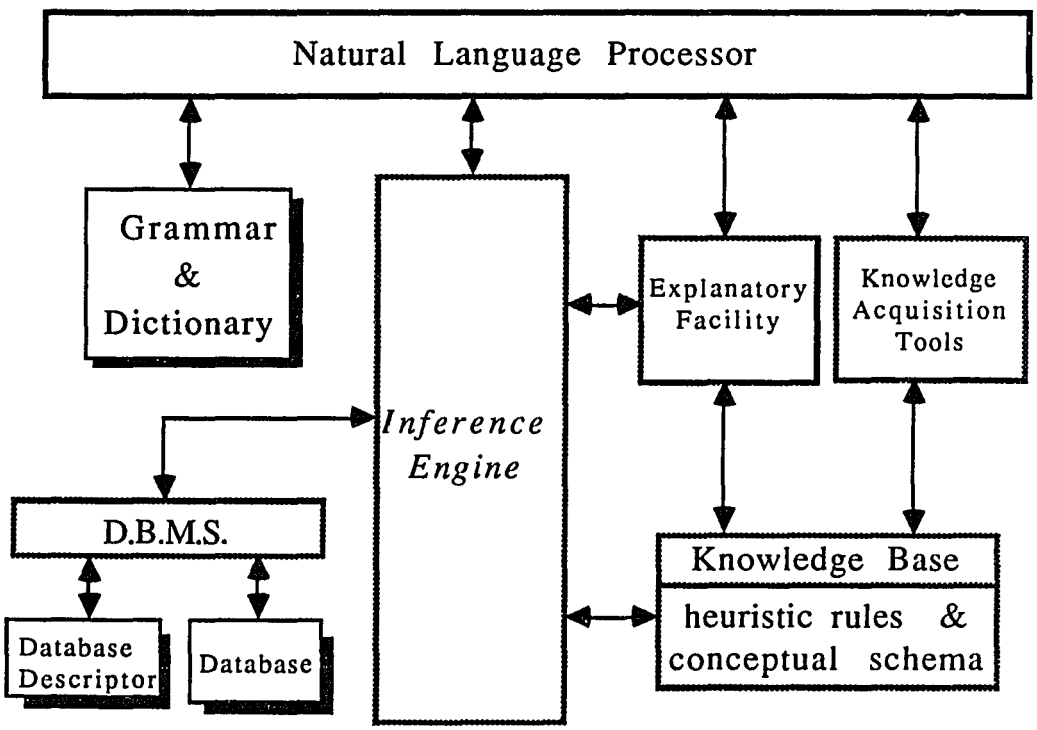


Figure 6.1 Basic structure of an expert system

of experimental frames [Zeigler 1984, Rozenblit and Zeigler 1988, Rozenblit and Hu 1989].

Given the system entity structure the designer has a choice of a number of design model alternatives. This is due to the multiplicity of aspects and specializations. We employ the production rule problem solving approach [Winston 1984] to support automatic selection of entities from specializations and synthesis of structures underlying the design model.

6.3 Design Phases of KBDSE

The Knowledge-Based Design Support Environment (KBDSE) applies modeling and simulation concepts to unify engineering design activities and develops a methodology for systematic design model construction and evaluation. The input of the KBDSE is a set of design specification underlining requirements, objectives, and criteria preference. The output of the KBDSE is a design model structure with the specification of design parameters for each model component.

Design models are derived from identifying multiple conflicting objectives and requirements of design specification. Therefore, design objectives play a fundamental role in guiding the synthesis of design models and the specification of experimental circumstances.

Evaluation of design alternatives is accomplished by simulation. The experimental frame concept [Zeigler 1984, Rozenblit 1986] is used to specify a simulation study. Briefly, an experimental frame defines conditions with which a design model can be observed and experimented. Alternative design models are evaluated with respect to experimental frames that reflect design performance questions. Simulation results are compared and traded off in preference of conflicting criteria. This results in a ranking of models and supports choices of alternatives that best satisfy the design specification.

With KBDSE, the system design process is formalized as the following:

- Selecting the problem domain by retrieving the desired FRASES tree.
- Identifying system requirements (e.g., cost, performance, technology, resources) from the design specification.
- Performing design reasoning based on the embedded knowledge to derive all possible alternative design models.
- Specifying simulation circumstances for service process, workload process, and simulation controls.
- Constructing experimental frames conforming to design objectives and simulation requirements.
- Coupling the design model with the experimental frame generated in step 5 and simulating each design model for performance analysis.
- Analyzing performance statistics and selecting the best design model.
- Report the best design model in step-7. If there exists no design models satisfying all performance requirements, then ask for performance tuning by adjusting related design parameters.

For illustration, a schematic representation of this design process is outlined in Figure 6.2. With KBDSE, the complex engineering design process is handled intelligently and efficiently to reduce the overall design cycle and cost.

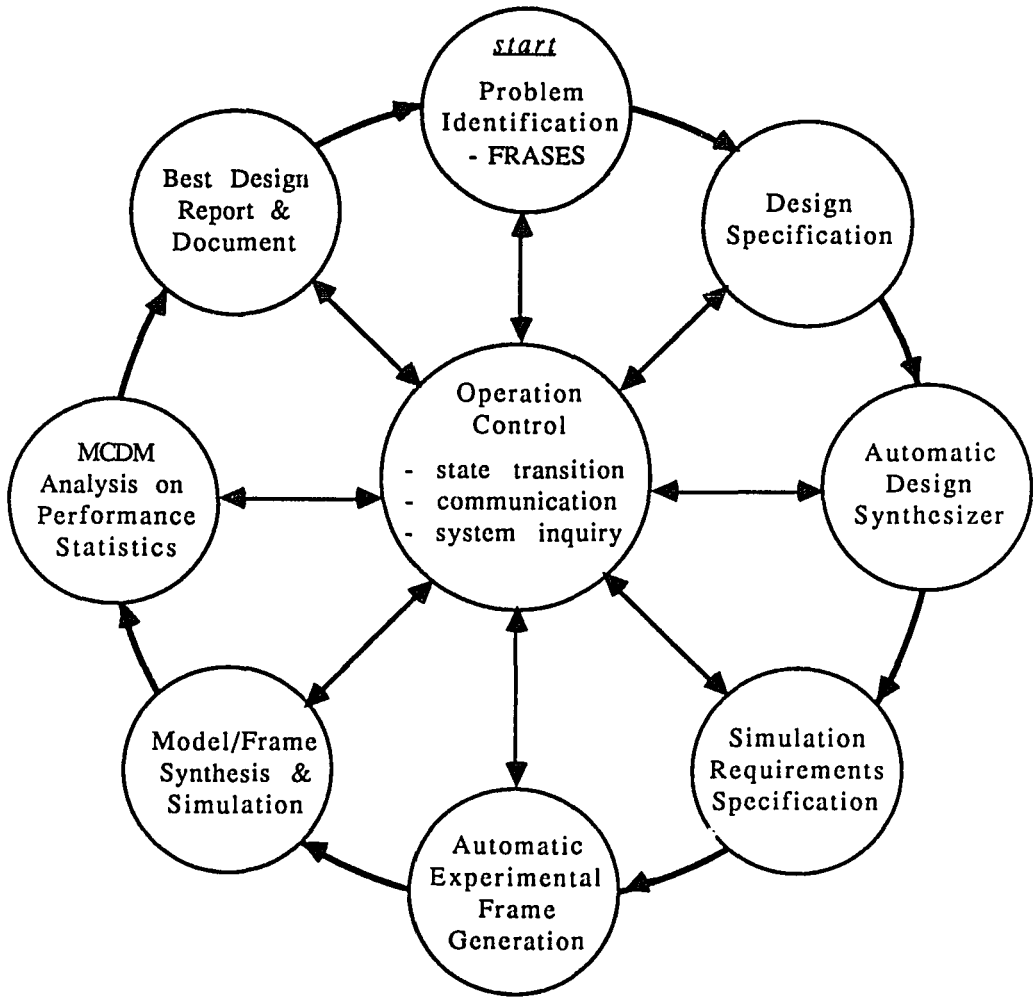


Figure 6.2 Design phases of KBDSE

6.4 System Architecture of KBDSE

The basic concept behind the integrated knowledge- based design methodology is to incorporate a team of expert systems to assist in the design process and achieve expert- level performance (e.g., at least to some extent).

The integrated Knowledge-Based Design Support Environment (KBDSE) is a software shell developed from integration of multiple intelligent systems. Corresponding to each design phase in the engineering design process, one or more special purpose knowledge-based systems are developed to provide technical assistance. For instance, the Design Specification Expert (DSE) aids in converting system requirements into design constraints, objectives, and criteria preferences; the expert system termed Automatic Design Synthesizer (ADS) derives all possible design alternatives; the Experiment Specification Expert (ESE) is employed to assist in specifying simulation circumstances; the Automatic Frame Generator (AFG) constructs experimental frames conforming to the specified simulation requirements; the Simulation Model Manager (SMM) converts each design alternative into a model structure for simulation; the Best Design Selector (BDS) selects the most appropriate design model by analyzing the preference of criteria and performance statistics of each design alternative. Throughout the design process, System Operation Manager (SOM) supervises and controls the transition between design phases. Along with other active modules, a schematic architecture of KBDSE is laid out in Figure 6.3.

6.5 Design Specification Expert (DSE)

A typical design may have many components, each of which is related to many other components. In KBDSE, the relationships between components are represented by the structure of FRASES. Given a design specification, design

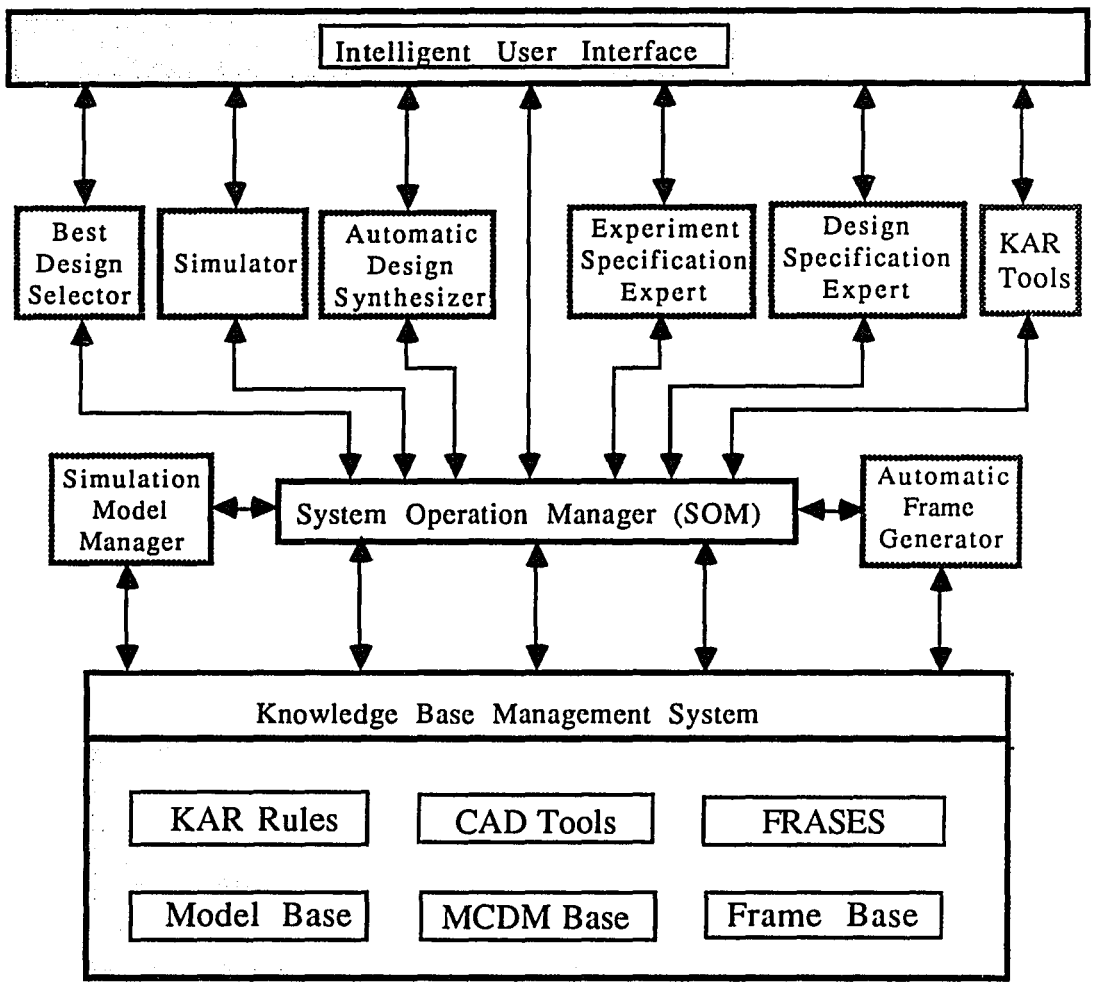


Figure 6.3 Architecture of KBDSE

modules are pruned from the FRASES. It is important to provide an efficient scheme for design specification. To facilitate design reasoning, system requirements are related to cost, performance, technologies, and resources are specified as design constraints, objectives, and criteria preferences. The design specification is acquired and saved in the Design Specification Form (DSF) for future processing.

Basic functions performed by DSE are:

- To generate query patterns for acquiring design objectives, constraints, and criteria preference.
- To explain questions such as "why" the requested information is needed; "what" the requested information means; and "how" to specify the requested information.
- To suggest possible performance indices and to point out that essential information may be missed.
- To check the validity and consistency of a design specification.
- To convert validated information into an internal representation that is ready for inference.

Formally, each FRASES node with the type "entity" may have a Design Specification Form (DSF) slot in its Entity Information Frame (EIF). In other words, users are allowed to specify design objectives, constraints, and criteria preference hierarchically at different levels of design abstraction.

6.5.1 Specifying Design Constraints

Design constraints implies requirements that must be satisfied by the resulting system. Each design constraint is expressed by the "relation", the "index", and the "value". Logic operators are allowed in multiple conditional specification of design constraints. For example, a typical design constraint specification for the processor of a computer module of a distributed system can be described as follows:

(AND (>= MIPS 10) (< cost 900) (= reliability 'high))

As shown above, the "value" used to indicate the quality (good, bad, high, low, etc.) or quantity (100, 0.9, etc.) of its associated attribute can be either a numerical or symbolic value. Specification of design constraints enables the system to compare specification of component models for the closest fit to the desired application, to derive the best architectural configuration for the system being designed, and to select the best-fit technology and vendors for design realization.

In general, design constraints can be classified into two types:

- Static constraints: This type of design constraint is expressed by static attributes. Static attributes are variables used to describe an object's general information. Typical static attributes include the unit cost, maximum capacity, maximum working temperature, technology type, etc. Static attributes are usually initiated with values (i.e., quantitative or qualitative) while constructing the FRASES structure. During design application, static attributes are directly applied without invoking simulation activities.

- Dynamic constraints: Dynamic constraints are specified with dynamic attributes. Dynamic attributes are indices related to performance analysis of design models. Values of dynamic performance indices are affected and

changed by design requirements and the behavioral characteristics of systems. Before a dynamic constraint can be applied to prune design structures, CAD tools and/or simulation activities are required to estimate the values of dynamic attributes. Typical dynamic attributes include thruput, response time, and utilization, etc.

6.5.2 Specifying Design Objectives

Design objectives imply the design goal. Information of design objectives will be used to conduct the design optimization. Typical specifications of design objectives are stated to maximize and/or minimize one or more performance indices. This is shown below:

((maximize index-1 ...) (minimize index-a ...))

For example, typical design objectives for the processor of a parallel/distributed system can be specified as:

((Max MIPS reliability) (Min cost))

6.5.3 Specifying Criteria Weighting

A criteria weighting scheme conveys the designer's preference over a set of design criteria (or performance indices). The specification of criteria weighting schemes will authorize the system to optimize the most crucial factor that limits the product's usefulness. Different design applications present different criteria preferences. For instance, in designing a storage peripheral, four primary criteria are considered: capacity, speed, cost, and reliability. The designers of office systems, for example, are concerned with only capacity, speed, and cost, in that

order of priority. The designers for the industrial environment, on the other hand, have different priorities: reliability being the most important, followed by cost and then speed. The least important criteria for industrial storage is capacity since factory-floor computers typically handle limited functions such as controlling one machine or monitoring one process. Therefore, their storage requirements are small.

In some cases, designers may choose a single criteria to drive optimization. For example, in consumer products it is cost; in aircraft it is weight; in implanted medical devices it is power consumption; in military systems it is reliability.

It is important to provide various criteria weighting schemes with which criteria preference can be indicated properly. With KBDSE, four types of criteria weighting schemes are allowed to express the preference over criteria:

- Unknown weighting: Unknown weighting is employed when the user is unable to give any preference between criteria. Under this situation, all design criteria are regarded as having equal importance. Available MCDM methods under this category are MaxiMin, MaxiMax, MiniMax, Hurwicz, Regret criterion, or Bayes-Laplace [Kmietowicz 1981, Osyczka 1984].

- Complete weighting: Complete weighting is used when the user is able to specify exact preference for each criterion. An example of this category is to assign each criterion a positive value and let the sum of the total weights be equal to 1:

$$(i.e. \sum_{i=1}^{n} P_i = 1).$$

- Ranked weighting: Ranked weighting is used whenever partial preference information is available but is not comprehensive enough to give the exact preference values of the criteria. Ranked weighting can be further classi-

fied into weak ranking (i.e. Pi > Pj) and strict ranking (i.e. Pi - Pj $\geq$ K). A typical example for weak ranking is to list the trade-off order of criteria. Available MCDM methods in this category are Fishburn's theorem, Extreme expected payoff method, and maximum variance method [Kmietowicz 1981, Canon 1974].

- Fuzzy weighting: Fuzzy weighting is used whenever the user is able to define the preference range of criteria with the lower/upper bound.

It is hard to say which MCDM method is the best. Application of MCDM methods in the wrong category may cause an incorrect decision in selecting the best design model. To assure a reliable solution, the most appropriate MCDM method must be selected carefully based on the characteristics of the design problems.

A typical specification of a criteria weighting scheme can be expressed as either one of the following:

Scheme	*Expression*
Rank	(:RW cost reliability speed)
Complete	(:CW (cost 0.5) (speed 0.2) (reliability 0.3))
Unknown	(:UW cost reliability speed)
Fuzzy	(:FW (cost 0.5 0.8) (reliability 0.2 0.4))

During design specification, it is preferred that an intelligent system be capable of recommending the possible degrees of importance for each performance index. For example, in computer bus design, if the bus is used for the transfer of programs between memory and CPU, then the efficiency of the allocation protocol is the most important, since the transfer will occur frequently and the amount of data will be relatively small (4 bytes for a 32-bit CPU). The time between the request for use of the bus and the completion of the data

transfer must be short. On the other hand, communication with a relatively slow peripheral device (e.g., a line printer) will occur less frequently and involve the transmission of long blocks of data. In this case, the data transfer rate is usually more critical than the allocation protocol. The explanatory facility of the Design Specification Expert (DSE) can be used with the expert's knowledge to help specify criteria preference appropriately and correctly.

6.5.4 Verifying Design Specifications

One of the most important functions performed by the DSE is to check the consistency and validity of design specification. For example, users may specify a performance index with an incorrect data type, have a syntax error in the design specification, or specify a non- allowable performance index (i.e., no knowledge associated for measuring the interested performance index). The Design Specification Expert (DSE) is responsible for detecting all errors before translating a design specification into constraint rules for design pruning.

6.6 Automatic Design Synthesizer (ADS)

Manual pruning has several drawbacks. For example, it requires an expert operator, it generates only one design configuration after each pruning process, and it requires enormous processing overhead.

The major goal of the Automatic Design Synthesizer (ADS) is to improve the drawbacks that exist in a manual approach. With ADS, the efficiency of design reasoning is increased and all design alternatives are generated simultaneously. Furthermore, ADS offers fast response to design modification. Basic functions performed by the Automatic Design Synthesizer (ADS) include:

- Pruning design alternatives.

- Tuning design parameters if necessary.
- Synthesizing design models.

6.6.1 Pruning Phases of KBDSE

Once system requirements are specified, the Automatic Design Synthesizer (ADS) is employed to derive all possible design alternatives. The overall design pruning process is accomplished by conducting the following pruning tasks:

- Quantity pruning: The major goal of quantity pruning is to employ the built-in inferencing engine (i.e., WOFIE) to interpret selection rules associated with each specialization frames (i.e., Entity Information Frames (EIF) associated with specialization nodes of a FRASES tree). On each inferencing cycle, pruning decisions are made based on evaluating premises of selection rules. As mentioned before, values of static attributes are retrieved directly. For dynamic attributes, heuristic rules, CAD tools, or quantitative functions may be used to estimate rough values. For example, in designing a parallel system, values for cost, processing time, and chip area should be considered and estimated in the design process. However, these values cannot be calculated exactly at the design stage, so instead, rough estimates of these values must be used to eliminate alternative components which are apparently not qualified for further processing. This process will limit the number of alternative design models to a manageable level. There are several criteria used to estimate the value of a performance index. For instance, the standard value for the performance index of a system component can be:

- related to the data of commercial products.
- expressed as a quantitative function for calculation.
- related to the supported functions or architectures.
- related to technology used for implementation.

Even though quantitative values cannot be calculated, qualitative characteristics can be estimated for pruning purposes. Quantity pruning will result in a pruned FRASES.

- Synthesis pruning: The major task performed by synthesis pruning is to convert the pruned FRASES into one or more composition trees. Each composition tree stands for a design structure. At each level of design abstraction, synthesis rules of aspect frames (i.e., Entity Information Frames associated with decomposition nodes of a FRASES tree) are examined to determine if technical constraints are satisfied by the synthesized structures. This, in turn, validates the configurations of design models. Synthesis pruning may result in zero or more design models. If there exists no design models (i.e., too strict design constraints), a new design specification must be given.

- Quality pruning: With KBDSE, performance evaluation is accomplished by simulation methods. The purpose of quality pruning is to remove design models whose performance does not meet the system requirements. The approach is to transform each composition tree obtained from synthesis pruning into a simulation model structure. Performance statistics of each design model are then collected and evaluated after simulation. Design models whose performance do not meet the desired quality are removed from the list of design alternatives. In the worst case, there may

exist none design model satisfying all the performance requirements. This can be resolved by adjusting design parameters related to the interested performance index. On the other hand, if multiple design models exist, the Best Design Selector (BDS) is invoked to recommend the best design model.

6.7 Best Design Selector (BDS)

Multi-Criteria Decision Making (MCDM) studies how to make the right decision under condition of conflicting situations, i.e. in a situation where several objectives must be satisfied. In order to make the best decision, trade-offs must be made between objectives. The most usual definition of "best" is the definition called "*Pareto Optimality*" [Marzollo 1975]:

> *A decision X^o is Pareto-optimal if no feasible X' exists such that $f(x') > f(x^o)$, where the 'f' is interpreted as the objective function.*

Given a set of design constraints and objectives (which could be in conflict with each other), there may exist more than one design alternative. To determine the best design model based on certain design specifications is not a trivial problem. In order to provide flexible specifications of criteria preference, four weighting schemes are defined: unknown weighting, complete weighting, ranked weighting, and fuzzy weighting. Each category is associated with a number of MCDM methods for solving the user's problems. The main functions performed by the Best Design Selector (BDS) are:

- Normalizing values of criteria
- Assigning minus to minimized criteria
- Selecting an appropriate MCDM method

- Reporting the best design model

6.8 Experiment Specification Expert (ESE)

To construct a simulation experiment for verifying the performance of a design model, simulation requirements must be specified carefully. The major task performed by ESE is to acquire essential information about simulation circumstances (e.g., arrival processes, workload processes, and simulation control schemes) and save the acquired information in the Experiment Specification Form (ESF) for future processing. Basic functions performed by ESE include:

- Suggesting possible simulation controls.
- Recommending appropriate service and workload processes.
- Instructing specification of simulation requirements.
- Detecting the inconsistency of information.

6.9 Atomic Frame Generator (AFG)

After composition trees are generated by ADS, each can be transformed into a model structure for simulation. To evaluate the performance of each design model, experimental frames corresponding to the desired performance indices must be generated and coupled to the design model structure. To relieve designers from programming and debugging math-intensive models for performance evaluation, the construction of experimental frames is automated by the Automatic Frame Generator (AFG). Basic functions performed by AFG are:

- Identifying performance indices from DSF

- Formulating simulation experiments
- Constructing experimental frames

6.10 Simulation Model Manager (SMM)

Upon invoking a transformation algorithm, the Simulation Model Manager (SMM) searches the behavior model base for component models denoted by leaf entities of a composition tree and synthesizes these component models with experimental frames by coupling them together in a hierarchical manner. Basic functions performed by SMM are:

- Coordinating the generation of experimental frames.
- Converting composition trees into simulation model structures.
- Conducting simulation and reporting performance statistics.

6.11 System Operation Manager (SOM)

The overall system operation sequence is controlled by the System Operation Manager (SOM). Major functions performed by the system operation manager include:

- Monitoring transitions between different expert systems. This includes exiting previous expert systems and activating an appropriate expert system based on the design phase.
- Facilitating communication between different expert systems. This will allow parameters to be exchanged between expert systems in different design phases.

- Handling error. This function enables the System Operation Manager (SOM) to explain an error and propose possible correction.

6.12 Organization of Knowledge Bases

To provide sufficient heuristics and information for design processing, a number of knowledge/data bases are associated with KBDSE. They are:

1. Behavior model base.
2. FRASES base.
3. Experimental frame base.
4. MCDM model base.
5. CAD tools library.
6. KAR rule base.

The behavior model base contains simulation models characterizing the dynamic behavior of system components.

The FRASES base stores a number of FRASES trees which in turn can be categorized into two types: non-pruned and pruned. A non-pruned FRASES denotes a design problem domain and a family of design configurations. Pruning a non-pruned FRASES based on a certain design specification will result in a pruned FRASES. Each pruned FRASES can be converted into one or more composition trees. Each composition tree represents an alternative design model. Each leaf entity of a composition tree must have a corresponding model in the behavior model base.

The experimental frame base contains special purpose models designed for simulation control and performance analysis. Two types of frames are included:

(a.) Generic atomic experimental frames.

(b.) Aggregated experimental frames.

Atomic experimental frames are basic elements used to compose an aggregated experimental frame. An aggregated experimental frame is capable of measuring a performance index (transducer), formulating input segments (generator), or controlling a simulation run (acceptor).

The MCDM model base contains mathematical models for solving multi-criteria decision making problems.

The CAD tool library provides algorithmic analysis on design attributes and calculates design parameters as requested.

The KAR rule base provides data and information required to generate question patterns for knowledge acquisition or to explain any system-initiated questions.

6.13 Example

To illustrate the whole process of the KBDSE design methodology, let us use the design of distributed systems (Figure 3.4) as an example.

According to the design phases of KBDSE in Figure 6.2, the desired FRASES tree (Figure 3.5) including the entity information frames for each node (Appendix 3.1) is first retrieved by the System Operation Manager (SOM).

The Design Specification Expert (DSE) then help the user complete his design specification. Assume the following Design Specification Form (DSF) is specified for the root node:

(DSF (constraint (value (> thruput 0.098) (< cost 300)))
(objective (Max (value thruput)) (Min (value cost)))
(criteria-weighting (value (rw thruput cost))))

After the design specification is defined, the Automatic Design Synthesizer (ADS) is activated to derive all possible alternative design models. This is accomplished by three pruning tasks: quantity, synthesis, and quality. To conduct quantity pruning, WOFIE is used.

According to the weight-oriented search, the first rule module to be interpreted is the "MTS-technology". The message transfer system is determined by asking information about the degree of interaction among computer modules. If the user indicates that the interaction among computer modules is low and resource sharing capability is required then the "local area network" will be selected for "MTS-technology". The "Interconnection Network" and its subtrees is then removed from the working FRASES.

The next rule module to be activated will be the "medium-access-protocol". In order to select appropriate medium access protocols, WOFIE asks the user what is the length of the individual transmission and what does the traffic pattern look like? If the length of the individual transmission is long and the traffic pattern between computer nodes is sporadic, then the CSMA/CD will be selected for medium access protocol.

WOFIE continues to traverse other rule modules (i.e., "line-protocol", "medium", "topology", "access", "address-type" in sequence) until all selection rules associated with specialization nodes are processed. Let us assume the following selections have been made during design reasoning:

line-protocol: *full-duplex*
medium: *optical-fiber*
topology: *bus, ring*
access: *direct-access, cache*
address-type: *virtual-address*

Quantity pruning by WOFIE results in a pruned FRASES as shown in Figure 6.4. After quantity pruning, the Automatic Design Synthesizer (ADS) activates synthesis pruning to transform the pruned FRASES into two composition trees (Figure 6.5). Notice that another two composition trees are removed during synthesis pruning by detecting the constraint, "If LAN- segment.medium = optical-fiber then LAN.topology != bus".

After synthesis pruning, quality pruning must be applied to remove design models that do not meet performance requirements (e.g., thruput in our case). In order to perform quality pruning, design models must be simulated under the desired simulation circumstance.

The Experiment Specification Expert (ESE) is activated to accept simulation requirements. Assume the user defines the Experiment Specification Form (ESF) for the distributed system (root) as follows:

```
(ESF (ap (value (cond (t (normal 20)))))
     (ef (value (cond (t (list (symbol) (number 1.0) )))))
     (sc (value (cond ((> event 100) (stop))))) )
```

Referring to Figure 6.2 and 6.3, the Automatic Frame Generator (AFG) is then activated to construct experimental frames for performance evaluation. Assume the DEVS-Scheme simulator [Zeigler 1986] is used. The experimental frame conforming to the simulation requirements is automatically constructed and coupled to the design model for simulation.

After the desired experimental frame is generated, the Simulation Model Manager (SMM) is activated to synthesize the simulation model structure by coupling models and frames in a hierarchical manner. Simulation is then conducted to evaluate alternative design models. Assume the simulation results for the interested performance index (i.e., thruput) are:

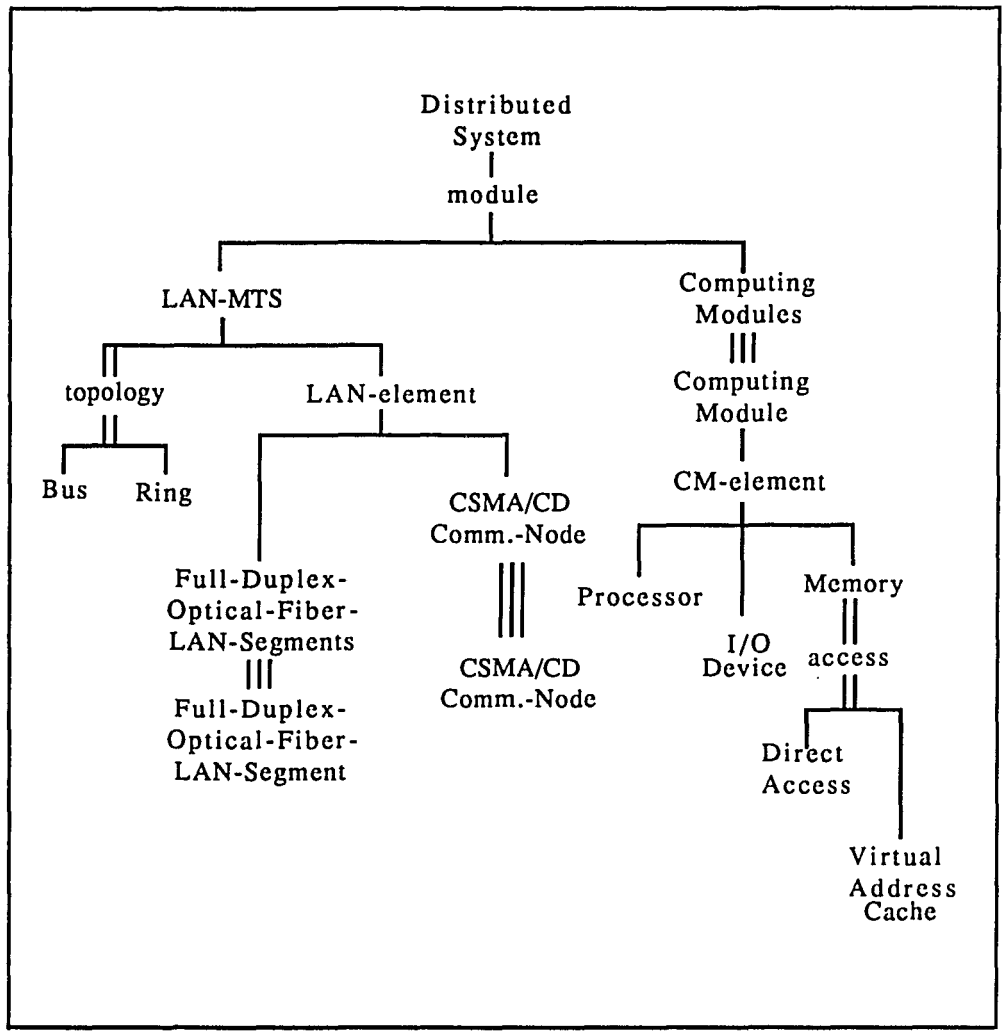
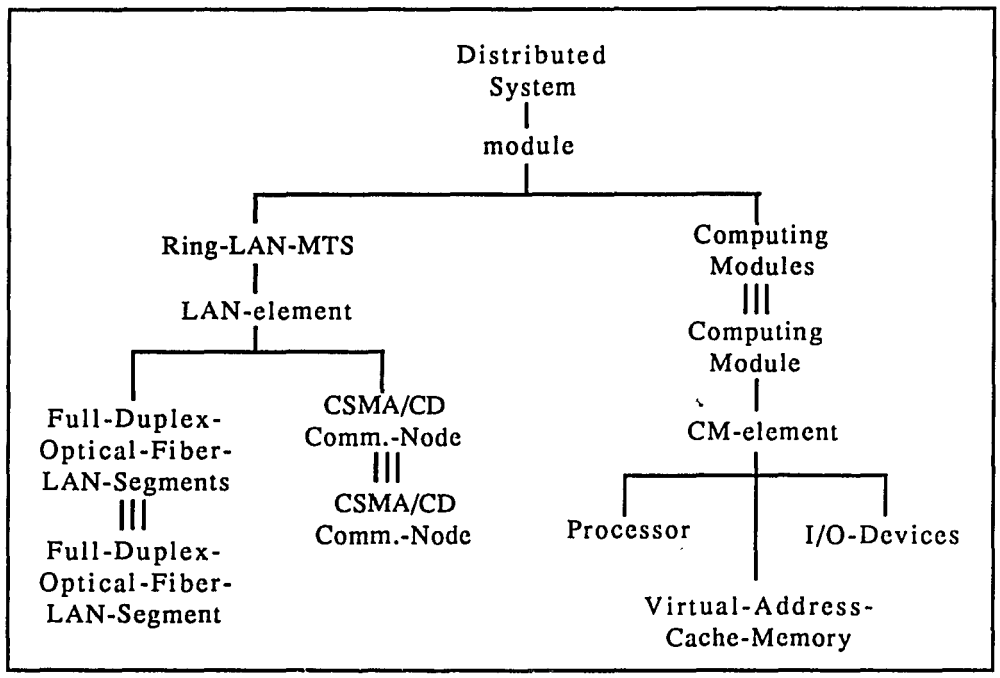


Figure 6.4 A pruned FRASES for distributed systems



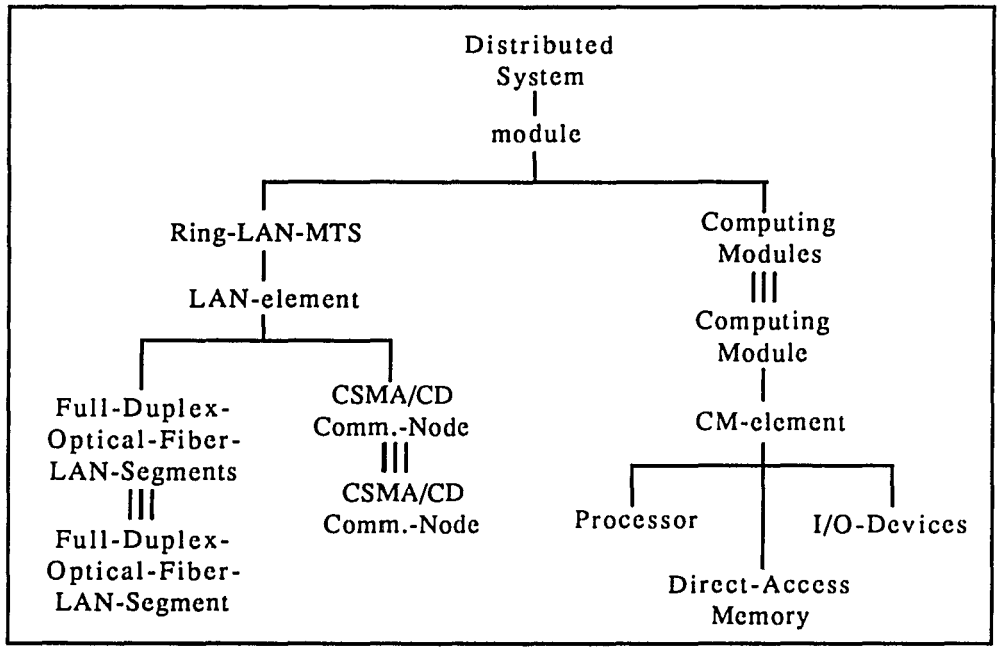


Figure 6.5 Composition trees for distributed systems

system-1 (design with direct access memory): 0.13

system-2 (design with virtual address cache): 0.15

Since both design models satisfy design constraints, the Best Design Selector (BDS) will be used to recommend the best design model. Assume design cost for both models are estimated as follows:

system-1 (design with direct access memory): 180

system-2 (design with virtual address cache): 250

After rating parameters and assigning negative signs to the second set of parameters (i.e., to minimize the cost), the MCDM model becomes:

	thruput	cost
system-1	0.867	-0.72
system-2	1.0	-1.0

Since the criteria preference is expressed by weak ranking [Kmietowicz 1981], we may use the extreme expected payoff method to solve the MCDM problem. The partial average for each system is computed as follows:

	thruput	cost
system-1	0.867	0.0735
system-2	1.0	0.0

Finally, the system-2 will be recommended, but only by a small margin (1.0 as against 0.941). Other MCDM methods may cause the system-1 to be recommended. This denotes the best design model may vary with the selected MCDM method. Different types of decision making problems may require different MCDM methods. There is no unambiguous MCDM method with a universal validity. It is hard to say which MCDM method is the best approach. To select the most appropriate MCDM method that conforms to the characteristics

of the decision making problem requires domain expertise. This motivates us to employ the knowledge-based approach for the realization of the Best Design Selector (BDS).

CHAPTER SEVEN

CONCLUSION AND FUTURE RESEARCH

7.1 Conclusion

In this dissertation an integrated Knowledge-Based Design Support Environment (KBDSE) is presented to facilitate the complex engineering design process. With KBDSE, the system design process is formalized as follows:

1. Automating knowledge elicitation, including both declarative and procedural knowledge, using the method of Knowledge Acquisition based on Representation (KAR).

2. Organizing acquired knowledge in a hierarchical and modular form managed under the entity-based Frames and Rules Associated System Entity Structure (FRASES).

3. Facilitating design specification for system requirements, including design constraints, objectives, and criteria preference, with the help of on-line explanation.

4. Reasoning all possible alternative design models via the Weight-Oriented FRASES Inferencing Engine (WOFIE) and adjusting design parameters for each model component by the application of design demons. The hierarchical, entity-based organization of procedural knowledge on FRASES reduces computation cost significantly and distinguishes it from conventional approaches.

5. Providing transparent schemes for specifying simulation requirements including arrival process, service process, and simulation controls.

6. Automating the generation of experimental frames for design verification and performance evaluation. The construction of the desired experimental frames is mainly driven by design objectives.

7. Applying Multi-Criteria Decision Making (MCDM) methods with performance statistics and criteria preference for selecting the best design model from multiple design alternatives.

As the complexity of systems increases, conventional acquisition methods become inhibited. To increase the efficiency of knowledge acquisition, it is necessary to facilitate the query process and the generation of query patterns. This motivates us to develop a new knowledge acquisition approach, termed Knowledge Acquisition based on Representation (KAR).

FRASES permits design knowledge to be organized in a highly modular and hierarchical manner. In other words, FRASES allows experts to hierarchically depict complex design knowledge into modules. This capability conforms not only to the common trait of modern engineering design but also enables an efficient management of design knowledge. In order to a.) increase the efficiency of design reasoning; b.) capture all dynamics of a design process; and c.) reduce the computation cost, a Weight-Oriented FRASES Inferencing Engine (WOFIE) was developed. In addition, FRASES facilitates knowledge refinement by exploiting its axioms and operations. Generally speaking, the development of FRASES enables an expert system to manage the embedded knowledge efficiently and intelligently.

The methodology for automatic generation of experimental frames relieves designers from programming and debugging math-intensive models for design verification and performance evaluation. This automation facility allows design models to be tested under different simulation circumstances without

wasting extraordinary labor in updating simulation software. This, in turn, reduces the overall design turnaround and increases the reliability of products for simulation-based system design.

Because of the increasing complexity in system structures, resources, and technologies, engineering design is becoming difficult without an expert's knowledge. The knowledge-based approach promises to solve knowledge- intensive problems which can not be solved easily by conventional algorithmic approaches. KBDSE offers designers of complex systems a multilevel design and simulation environment, lower design cost and faster simulation; and multi-criteria decision making. KBDSE applies AI techniques to automate design synthesis and performance modeling. This, in turn, increases the productivity and quality of design products.

7.2 Future Research

Topics for future research include:

- To incorporate psychological methods in the KAR methodology for a.) developing the best query patterns with which domain experts are more willing to share their knowledge; b.) determining the reason a user has asked a question so that a customized response can be generated; and c.) verifying the degree of validity of knowledge with psychological factors.
- To incorporate a natural language processor to upgrade the KAR interface, to facilitate design specification, and simulation circumstances.
- To extend program automation for behavioral modelling of system components. It is preferred to have a natural language as the input media to the program generator.

APPENDIX 2.1

IMPLEMENTATION OF A FRAME SYSTEM

```
;;*********************************************************;;
;;* Author   :  Jhyfang  Hu                               *;;
;;* Language :  Common-Lisp                               *;;
;;* Package  :  KBDSE                                     *;;
;;* Objective:  Management of frame objects               *;;
;;*********************************************************;;

;;*=======================================================*;;
;;* A basic frame object has the following structure:     *;;
;;*     (HENRY  (ako     (value man) )                    *;;
;;*             (height  (value 5.4) )                    *;;
;;*             (weight  (value 120)                      *;;
;;*                      (if-needed  frame-ask) )         *;;
;;*             (hobbies (default jogging skiing) )       *;;
;;*             )  ;;end of HENRY                         *;;
;;*                                                       *;;
;;*   Slot  -> variable type...                           *;;
;;*   Facet -> value, default, if-needed, if-added,..     *;;
;;*   Value -> any numerical or symbolic expression       *;;
;;*=======================================================*;;

;;*-------------------------------------------------------*;;
;;* Get value from the specified slot and facet.          *;;
;;* Example: -> (frame-get 'paul 'weight 'value)          *;;
;;*-------------------------------------------------------*;;
(defun FRAME-GET (frame slot facet)
   (let ( (frame (if (listp frame) (eval frame) frame)) )
       (cdr (assoc facet (cdr (assoc slot
                                     (cdr (get frame 'eif))
                               ))
                  ))))
;;*-------------------------------------------------------*;;
;;* Get value from the specified path.                    *;;
;;* Example: -> (frases-get 'paul 'weight 'cm 'value)     *;;
;;*-------------------------------------------------------*;;
(defun FRASES-GET (frame &rest keys)
   (let ( (frame (if (listp frame) (eval frame) frame)) )
       (do (
            (keys keys (cdr keys))
            (str (cdr (get frame 'eif)))
           )
           ((or (null keys) (null str)) str)
           (setq str (cdr (assoc (car keys) str))) )))
```

```
;;*-------------------------------------------------------*;;
;;* Get value for the specified slot.                     *;;
;;* Both VALUE & DEFAULT facets are checked.              *;;
;;* Example: -> (frame-get-vd 'paul 'weight)  => (58.41)  *;;
;;* Example: -> (frases-get-vd 'paul 'weight 'value)      *;;
;;*-------------------------------------------------------*;;
(defmacro FRASES-GET-VD (frame &rest slots)
    `(cond ((frame-get ',frame ,@slots 'value))
           ((frame-get ',frame ,@slots 'default)) ))

(defun FRAME-GET-VD (frame slot)
    (cond ((frame-get frame slot 'value))
          ((frame-get frame slot 'default)) ))
;;*-------------------------------------------------------*;;
;;* Get slot value by checking VALUE, DEFAULT, and        *;;
;;* IF-NEEDED facets.                                     *;;
;;* Example: -> (frame-get-vdp 'paul 'weight)             *;;
;;* Example: -> (frases-get-vdp 'paul 'weight 'cm)        *;;
;;*-------------------------------------------------------*;;
(defmacro FRASES-GET-VDP (frame &rest slots)
    `(cond ( (frame-get-vd ',frame ,@slots) )
           (t (mapcan
                #'(lambda (demon) (when demon
                      (funcall demon ',frame ,@slots)))
                (frame-get ',frame ,@slots 'if-needed) )) ))

(defun FRAME-GET-VDP (frame slot)
    (cond ( (frame-get-vd frame slot) )
          (t (mapcan
               #'(lambda (demon)
                    (when demon (funcall demon frame slot)))
               (frame-get frame slot 'if-needed) )) ))
;;*-------------------------------------------------------*;;
;;* Get slot value by using Z-inheritance. On each frame, *;;
;;* VALUE, DEFAULT, and IF-NEEDED facets are searched.    *;;
;;* Example: -> (frame-z-get 'paul 'weight)               *;;
;;*-------------------------------------------------------*;;
(defun FRAME-Z-GET (frame slot)
    (do* (
          (frames (frame-get-inheritances frame)
                  (cdr frames) )
          (value nil (frame-get-vdp (car frames) slot) )
         )
        ((or value (null frames)) value)))
;;*-------------------------------------------------------*;;
;;* Get slot value  by using I-inheritance.               *;;
;;* On each frame, only VALUE facet is searched.          *;;
;;* Example: -> (frame-i-get 'paul 'weight)               *;;
;;*-------------------------------------------------------*;;
```

```
(defun FRAME-I-GET (frame slot)
    (let ( (frames (frame-get-inheritances frame)) )
       (frame-get-facet frames slot 'value) ))
;;*-------------------------------------------------------*;;
;;* Get slot value by using N-inheritance.               *;;
;;* I-inheritance with VALUE, I-inheritance with DEFAULT  *;;
;;* and, I inheritance with IF-NEEDED.                    *;;
;;* Example: -> (frame-n-get 'paul 'weight)               *;;
;;*-------------------------------------------------------*;;
(defun FRAME-N-GET (frame slot)
   (let ((frames (frame-get-inheritances frame)))
      (cond ((frame-get-facet frames slot 'value))
            ((frame-get-facet frames slot 'default))
            ((frame-get-facet frames slot 'if-needed))
            (t nil)  )))
;;*-------------------------------------------------------*;;
;;* Get slot/facet value from a set of frames.           *;;
;;* (frame-get-facet '(man paul) 'weight 'value)         *;;
;;*-------------------------------------------------------*;;
(defun FRAME-GET-FACET (frames slot facet)
   (do (
        (value nil)
        (classes frames (cdr classes))
       )
       ((or value (null classes)) value)
       (if (or (equal facet 'value) (equal facet 'default))
           (setq value (frame-get (car classes) slot facet))
           (setq value
              (mapcan #'(lambda (demon) (apply demon nil))
                      (frame-get (car classes) slot facet)))
       )))
;;*-------------------------------------------------------*;;
;;* Get all inherited frames through AKO relation        *;;
;;* Example: -> (frame-get-inheritance 'paul)            *;;
;;*-------------------------------------------------------*;;
(defun FRAME-GET-INHERITANCES (frame)
   (do (
        (frames '())
        (tmp-frames (list frame) (cdr tmp-frames))
       )
       ((null tmp-frames) frames)
       (setq frames (append frames (list (car tmp-frames))))
       (setq tmp-frames
             (append tmp-frames
                (frame-get-vdp (car tmp-frames) 'ako) )) ))
;;*-------------------------------------------------------*;;
;;* Set the value of particular slot/facet.              *;;
;;* Example: -> (frame-set 'john 'occupation 'teaching)  *;;
;;*-------------------------------------------------------*;;
```

```
(defun FRAME-SET (object &optional (var nil) (val nil) )
   (cond
      ((and var val)
       (frame-remove object var 'value)
       (frame-put object var 'value val) )
      (var (frame-remove object var))
      (t (setf (get object 'eif) nil) ))  )
;;*-------------------------------------------------------*;;
;;* Set slot/facet value of a frame object               *;;
;;* -> (frame-vector-set 'john '((occupation teaching)    *;;
;;*                              (hobby swimming) ))      *;;
;;*-------------------------------------------------------*;;
(defun FRAME-VECTOR-SET (object p-lst)
   (mapcar
      #'(lambda (e) (frame-set object (car e) (cadr e)) )
      p-lst) object )
;;*-------------------------------------------------------*;;
;;* Find out which frame contains the indicated slot      *;;
;;* Example: -> (frame-find-object 'man 'occupation)      *;;
;;*-------------------------------------------------------*;;
(defun FRAME-FIND-OBJECT (source var)
   (do* ( (object nil)
          (lst (frame-get-inheritances source) (cdr lst))
          (inst-vars nil)
          (class-vars nil) )
     ((or object (null lst)) object)
     (setq inst-vars (frame-get-vdp (car lst) 'inst-vars))
     (setq class-vars (frame-get-vdp (car lst) 'class-vars))
     (if (or (member var inst-vars)
             (member var class-vars)
             (frame-get-vd (car lst) var) )
         (setq object (car lst)) )) )
;;*-------------------------------------------------------*;;
;;* Find out which frame contains the indicated slot      *;;
;;* -> (frame-locate-attribute 'man 'methods 'earings)    *;;
;;*-------------------------------------------------------*;;
(defun FRAME-LOCATE-ATTRIBUTE (source type var)
   (do ( (object nil)
         (lst (list source) (append (frame-get-vd (car lst)
                                                  'ako) (cdr lst)))
       )
     ((or object (null lst)) object)
     (if (assoc var (frame-get-vd (car lst) type))
         (setq object (car lst)) )))
;;*-------------------------------------------------------*;;
;;* Create a new frame object                            *;;
;;*-------------------------------------------------------*;;
(defun FRAME-SET-NEW (object frame)
   (setf (get object 'eif) frame) )
```

```
;;*-------------------------------------------------------*;;
;;* Find out which frame contains the indicated slot      *;;
;;* -> (frame-get-attribute 'man 'methods 'earings)       *;;
;;*-------------------------------------------------------*;;
(defun FRAME-GET-ATTRIBUTE (source type var)
   (let ( (pair (assoc var (frame-get-vd source type))) )
      (if pair (cdr pair) nil) ))
;;*-------------------------------------------------------*;;
;;* Find out which frame contains the indicated slot      *;;
;;* -> (frame-z-get-attribute 'man 'methods 'earings)     *;;
;;*-------------------------------------------------------*;;
(defun FRAME-Z-GET-ATTRIBUTE (source type var)
   (do* ( (value nil)
          (lst (list source) (append (frame-get-vd (car lst)
                                          'ako) (cdr lst)))
        )
      ((or value (null lst)) (if value (cdr value)) )
      (setq value (assoc var (frame-get-vd (car lst) type)))
   ))
;;*-------------------------------------------------------*;;
;;* Find out which frame contains the indicated slot      *;;
;;* -> (frame-set-attribute 'cpu 'attributes 'thruput 20) *;;
;;*-------------------------------------------------------*;;
(defun FRAME-SET-ATTRIBUTE (source type var value)
   (let ( (pair nil) (exist? nil) )
      (do ((classes (frame-get-inheritances source)
                    (cdr classes) ))
        ((or exist? (null classes))
         (if exist?
             (progn
               (frame-remove source type 'value exist?)
               (frame-put source type 'value
                                  (cons var value)))
             (error "->> Undefined variable ~S <<-" var) ))
        (setq pair (frame-get-vd (car classes) type))
        (setq exist? (assoc var pair)) )))

;;*-------------------------------------------------------*;;
;;* Destroy the frame structure                           *;;
;;*-------------------------------------------------------*;;
(defun FRAME-DEL-STRUCTURE (frame)
   (setf (get frame 'EIF) nil) )

;;*-------------------------------------------------------*;;
;;* Set the slot/facet with a value. If the value existed *;;
;;* then return nil, else insert the new value.           *;;
;;* Example: -> (frame-put 'paul 'weight 'value 45)       *;;
;;* Example: -> (frases-put 'paul 'weight 'cm 'value 180) *;;
;;*-------------------------------------------------------*;;
```

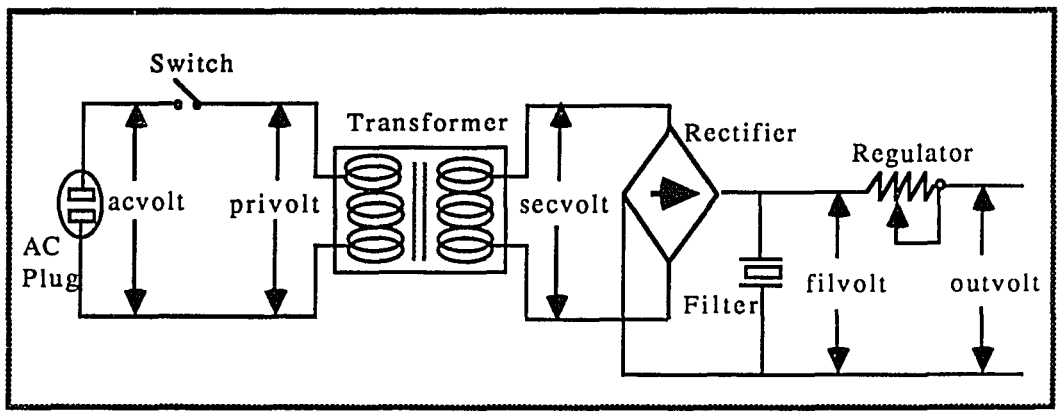
```
(defun FRASES-PUT (frame &rest keys)
   (let* (
          (value (car (last keys)))
          (ptr (butlast keys))
          (value-lst (frame-follow-path ptr
                          (frame-get-structure frame) ))
         )
      (cond
        ((flat-member value value-lst) nil)
        (t (rplacd (last value-lst) (list value)) value ) )))

(defun FRAME-PUT (frame slot facet value)
   (when value
      (let ( (value-lst
               (frame-follow-path (list slot facet)
                   (frame-get-structure frame) ))
           )
      (cond ((flat-member value value-lst) nil)
            (t (rplacd (last value-lst) (list value)) value)
      ))))
;;*-------------------------------------------------------*;;
;;* Set the slot/facet with a value and execute the       *;;
;;* procedure associated with IF-ADDED facet              *;;
;;* Example: -> (frame-put-p 'paul 'weight 'value 45)     *;;
;;*-------------------------------------------------------*;;
(defun FRAME-PUT-P (frame slot facet value)
    (cond ((frame-put frame slot facet value)
           (mapcar
               #'(lambda (f)
                     (mapcar
                         #'(lambda (demon)
                               (funcall demon frame slot) )
                         (frame-get f slot 'if-added) ))
               (frame-get-inheritances frame) )
          value ) ))
;;*-------------------------------------------------------*;;
;;* Remove a value from the indicated slot/facet.         *;;
;;* Example: -> (frame-remove 'paul 'weight 'value 54)    *;;
;;* Example: -> (frases-remove 'paul 'weight 'value 180)  *;;
;;*-------------------------------------------------------*;;
(defun FRAME-REMOVE (frame slot &optional facet value)
    (let ((slot-lst (frame-follow-path (list slot)
                        (frame-get-structure frame) )))
     (cond
       ((or facet value)
        (let ((value-lst (frame-follow-path
                             (list facet) slot-lst)))
           (if value
               (if (flat-member value value-lst)
```

```
                (delete value value-lst) )
              (rplacd value-lst nil)  ;;remove a facet
          )))
       (t (rplacd slot-lst nil)) ;;remove a slot
     )))

(defun FRASES-REMOVE (frame &rest slots)
    (let* (
           (value (car (last slots)))
           (keys (butlast slots))
           (value-lst (frame-follow-path keys
                        (frame-get-structure frame) ))
         )
       (delete value value-lst) ))
;;*-----------------------------------------------------*;;
;;* Remove a slot/facet value and execute the procedure  *;;
;;* associated with IF-REMOVED facet                     *;;
;;* Example: -> (frame-remove-p 'paul 'weight 'value 45) *;;
;;*-----------------------------------------------------*;;
(defun FRAME-REMOVE-P (frame slot &optional (facet nil)
                                            (value nil))
    (cond ((frame-remove frame slot facet value)
           (mapcar
               #'(lambda (f)
                     (mapcar
                         #'(lambda (demon)
                               (apply demon nil) )
                         (frame-get f slot 'if-removed) ))
                 (frame-get-inheritances frame) ) value) ))
;;*-----------------------------------------------------*;;
;;* Get the Entity Information Frame for an object.      *;;
;;* Example: -> (frame-get-structure 'cpu)               *;;
;;*-----------------------------------------------------*;;
(defun FRAME-GET-STRUCTURE (frame)
    (cond ((get frame 'EIF))
          (t (setf (get frame 'EIF) (list frame)) )))
;;*-----------------------------------------------------*;;
;;* Search the path, if existed return the CDR part,     *;;
;;* else make one and add it to the frame structure.     *;;
;;* (frame-extend 'hobby '(john (hobby (value jogging)))) *;;
;;*-----------------------------------------------------*;;
(defun FRAME-EXTEND (key lst)
    (cond ((assoc key (cdr lst)))
          (t (cadr (rplacd (last lst) (list (list key)))))) ))
;;*-----------------------------------------------------*;;
;;* Follow the searching path and get the value list     *;;
;;* -> (frame-follow-path '(hobbies value)               *;;
;;*        '(john (hobbies (value jogging skiing))) )    *;;
;;*-----------------------------------------------------*;;
```

```
(defun FRAME-FOLLOW-PATH (path frame-str)
    (cond ((null path) frame-str)
          (t (frame-follow-path
                 (cdr path)
                 (frame-extend (car path) frame-str)) )
    ))
;;*-----------------------------------------------------*;;
;;* An ASK-procedure associated with the IF-NEEDED facet. *;;
;;* -> (frame-put 'henry 'hobbies 'if-needed 'frame-ask)  *;;
;;* -> (frame-get-vdp 'henry 'hobbies)                    *;;
;;*-----------------------------------------------------*;;
(defun FRASES-ASK (frame &rest slot)
    (princ " What is the value for ")
    (write (car (last slot)))
    (princ " of ") (write frame) (princ " => ")
    (frame-put frame slot 'value (read)) )

(defun FRAME-ASK (frame slot)
    (princ " What is the value for ") (write slot)
    (princ " of ") (write frame) (princ " => ")
    (frame-put frame slot 'value (read)) )
;;*-----------------------------------------------------*;;
;;* Check if value is defined in particular facet of slot *;;
;;* Example: -> (frame-value-ck 'henry 'occupation 'value *;;
;;*                              'teacher )               *;;
;;*-----------------------------------------------------*;;
(defun FRAME-VALUE-CK (frame slot facet value)
    (if (flat-member value (frame-get frame slot facet))
        t nil ))
;;*-----------------------------------------------------*;;
;;* Check if the specified slot/attribute exists in the   *;;
;;* current  class or inherited classes.                  *;;
;;* -> (frame-z-check 'cpu 'class-vars 'clock-rate)       *;;
;;*-----------------------------------------------------*;;
(defun FRAME-Z-CHECK (frame slot attribute)
    (do (
         (value nil)
         (frames (frame-get-inheritances frame) (cdr frames))
        )
        ((or value (null frames)) value)
       (if (flat-member attribute
              (frame-get-vdp (car frames) slot))
           (setq value t) )
    ))
;;*************** End of frame system *********************;;
;;*****************************************************;;
```

APPENDIX 2.2

FRASES FOR OTHER APPLICATIONS



EIF: pc-module

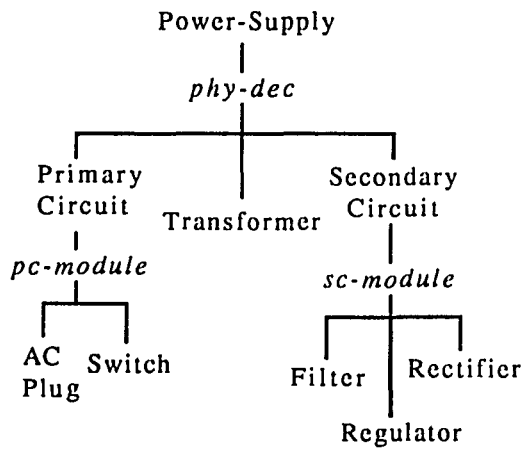
```
(CRS
   (if acvolt = ok and privolt = 0
   then the problem is bad switch)
   (if acvolt = 0
   then the problem is bad AC plug))
```

EIF: phy-dec

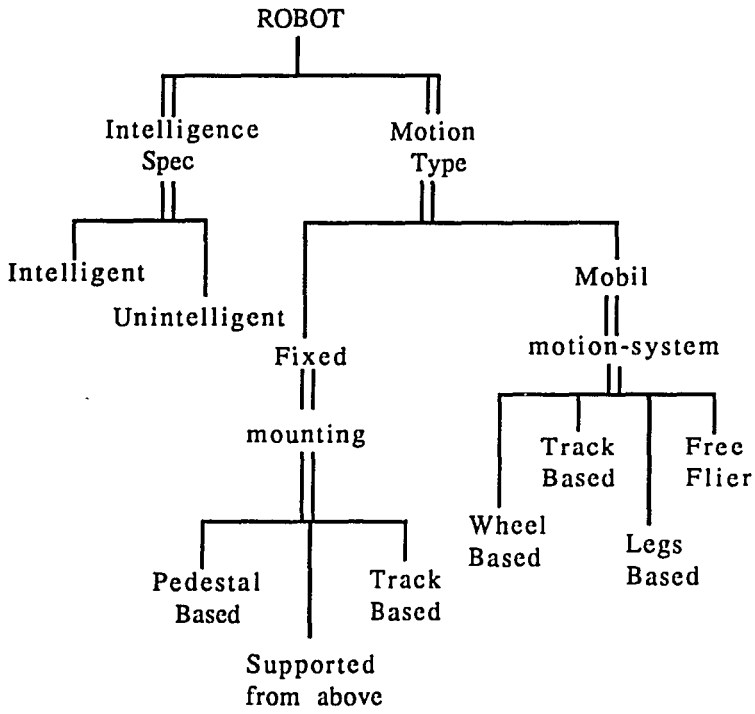
```
(CRS
   (if privolt = 0
   then the problem is primary circuit)
   (if privolt = ok and secvolt = 0
   then the problem is bad transformer)
   (if secvolt = ok and outvolt = 0
   then the problem is secondary circuit))
```

EIF: sc-module

```
(CRS
  (if filvolt = low
  then the problem is bad filter)
  (if outvolt = 0 and filvolt = ok
  then the problem is open regulator)
  (if filvolt = ok and outvolt = high
  then the problem is short regulator)
  (if secvolt = ok and filvolt = 0
  then the problem is bad rectifier))
```



Fault diagnosis with FRASES



Intelligence-Spec.EIF.CRS:
((if desired autonomy is high or medium then select Intelligent Robot)
(if desired autonomy is low then select Unintelligent ROBOT))

Motion-Type-EIF.CRS:
((if budget is low and working area is less than 25 square feet or
arm carrying capacity is larger than 1000 lbs
then select Fixed ROBOT)
(if budget is high and working area is larger than 25 square feet or
arm carrying capacity is less than 1000 lbs
then select Mobil ROBOT))

Mounting-.EIF.CRS:
((if Motion-type is fixed and power consumption is low and
working area has a solid ground then select Pedestal-Based mounting)
(if Motion-type is fixed and power consumption is medium and
degree of freedom is low and working area has a soft ground
then select Support-from-above mounting) ....)

Robot selection with FRASES

APPENDIX 3.1

ENTITY INFORMATION FRAMES
FOR AN ABSTRACT DISTRIBUTED SYSTEM

```
********
  (DISTRIBUTED-SYSTEM
        (M (Value Dist-Sys))
        (ATTS (SATT (TYPE (Value e))
                    (IPORTS (Value In))
                    (OPORTS (Value Out))
                    (WEIGHT (Value 0)) )
              (DPARA (COST (Default Compute-Cost))) ;;demon
              (PIX  (THRUPUT
                       (Default (/ (counter Out) (timer) )))))
        (CH  (Value module)) )

********
  (MODULE
     (ATTS (SATT (TYPE (Value a)) ... )
           ...
     (CH (Value message-transfer-system computer-modules)) )

********
  (MESSAGE-TRANSFER-SYSTEM
     (M (Value MTS))
     (ATTS (SATT (TYPE (Value e))
                 (IPORTS (Value Msg-in))
                 (OPORTS (Value Msg-out))
                 (WEIGHT (Value 2)) )
           (PIX (RESPONSE-TIME
                   (Default (- (time-reader msg-out)
                               (time-reader msg-in) )))) ...)
     (CH (Value mts-technology)) )

 ********
  (COMPUTER-MODULES
     (M (Value CM))
     (ATTS (SATT (TYPE (Value e))
                 (IPORTS (Value cm-in))
                 (OPORTS (Value cm-out))
                 (WEIGHT (Value 1)) ... )
           ... ) ... )

********
  (MEMORY
     (M (Value MEM))
```

```
      (ATTS (SATT (TYPE (Value e))
                  (WEIGHT (Value 1)) ... )))

********
  (I/O-DEVICES
     (M (Value I/O))
     (ATTS (SATT (TYPE (Value e))
                 (WEIGHT (Value 2)) ... )))

********
  (PROCESSOR
     (M (Value P))
     (ATTS (SATT (TYPE (Value e))
                 (WEIGHT (Value 3)) ... )))

********
  (MTS-TECHNOLOGY
    (ATTS (SATT (TYPE (Value s)) ... ) ... )
    (CRS (SEL
       (Value
        (s1 (if interaction among computer modules = high and
                resource sharing capability = no
             then select IN))
        (s2 (if interaction among computer modules != high and
                resource sharing capability = yes
             then select LAN))
      ))) ... )

********
  (Local-Area-Network
     (ATTS (SATT (TYPE (Value e))
           (Constraints
              (if LAN-segment.medium = optical-fiber
               then LAN.topology != bus))
           ... ))

********
  (TOPOLOGY
    (ATTS (SATT
             (TYPE (Value s))
             (WEIGHT (Value 1) ))
    (CRS (SEL
       (Value
          (s1 (if desired network reliability = high and
                  relationship between stations = peer-to-peer
               then select bus))
          (s2 (if desired network reliability = low and
                  relationship between stations = peer-to-peer
               then select ring))
```

```
          (s3 (if desired network reliability = medium and
                  relationship between stations = master-slave
               then select star))
       ))) ... )

********
  (LAN-ELEMENT
     (ATTS (SATT (TYPE (Value a))
                 (WEIGHT (Value 2)) )) ... )

********
  (LAN-SEGMENTS
     (ATTS (SATT (TYPE (Value e))
                 (WEIGHT (Value 1)) )) ... )

********
  (COMMUNICATION-NODES
     (ATTS (SATT (TYPE (Value e))
                 (WEIGHT (Value 2)) )) ... )

********
  (ACCESS-PROTOCOL
    (ATTS (SATT (TYPE (Value s)) ))
    (CRS (SEL
       (Value
        (s1 (if length of individual transmissions = long and
                traffic pattern between stations = sporadic
             then select CSMA/CD))
        (s2 (if length of individual transmissions = short and
                traffic pattern between stations = heavy
             then select Token-Passing))
        ))) ... )

********
  (MEDIUM
    (ATTS (SATT
             (TYPE (Value s))
             (WEIGHT (Value 1)) ))
    (CRS (SEL
       (Value
          (s1 (if noise-immunity-requested = low and
                  maximum nodes in network < 1024 and
                  maximum bandwidth <= 4 * 10^6
               then select twisted-pair))
          (s2 (if type of signal transmitted = analogy and
                  maximum nodes in network < 1024 and
                  noise immunity requested = low-to-medium
               then select baseband-coaxial-cable))
          (s3 (if maximum bandwidth <= 400*10^6 and
```

```
                    noise-immunity-requested = medium and
                    maximum nodes in network < 25000
                 then select broadband-coaxial-cable))
           (s4 (if maximum-nodes-in-network < 1024 and
                    maximum-bandwidth <= 10^9 and
                    noise-immunity-requested = high
                 then select optical-fiber))
        ))) ... )

********
  (LINE-PROTOCOL
    (ATTS (SATT (TYPE (Value s))
                (WEIGHT (Value 2)) ))
    (CRS (SEL
       (Value
          (s1 (if two-way transmission at the same time = yes
               then select full-duplex))
          (s2 (if two-way transmission = yes
               then select half-duplex))
          (s3 (if two-way transmission = no
               then select simplex))
       ))) ... )

********
  (IN-TECHNOLOGY
    (ATTS (SATT (TYPE (Value s)) ))
    (CRS (SEL (Value
      (s1 (if desired network complexity = low and
              active nodes to be supported <= 30
           then select shared-bus))
      (s2 (if desired network complexity = high and
              active nodes to be supported < 10
           then select crossbar))
      (s3 (if desired network complexity = medium and
              active nodes to be supported > 30
           then select multistage interconnection network))
      ))) ... )

********
  (TWISTED-PAIR-WIRE
     (M (Value twist-pair))
     (ATTS (SATT
              (TYPE (Value e))
              (UNIT-PRICE (Value 0.25)) ;dollar/foot
              (MAX-ALLOWABLE-DISTANCE (Value 1000)) ;feet
              (MAX-DATA-RATE (Default 10^6)) ;bps
              (IPORTS (Value in))
              (OPORTS (Value out)) )
           (DPARA
```

```
                (LENGTH (if-needed ask)) ;feet
                (DATA-RATE (Value max-data-rate))
                (COST (Value (* length unit-price))) )
             (PIX  (RESPONSE-TIME
                      (Estimate (/ 1 data-rate))))
      ) ... )

********
(BASEBAND-COAXIAL-CABLE
   (M (Value base-cable))
   (ATTS (SATT
       (TYPE (Value e))
       (UNIT-PRICE (Value 3)) ;dollar/foot
       (MAX-ALLOWABLE-DISTANCE (Value 5000)) ;feet/repeater
       (MAX-CHANNELS (Value 1))
       (MAX-DATA-RATE/CHANNEL (Value 10^6))
       (TYPE-OF-SIGNAL-SUPPORTED (Value analog))
       (BITS-INDICATION (Value voltage))
       (BROADCASTING (Value two-way))
       (IPORTS (Value in))
       (OPORTS (Value out)) )
    (DPARA
       (LENGTH (if-needed ask)) ;feet
       (COST (Value (* length unit-price)))
       (DATA-RATE (Value
              (* max-channels max-data-rate/channel)) )
    (PIX  (RESPONSE-TIME (Estimate (/ 1 data-rate))))
    ) ... )

********
(BROADBAND-COAXIAL-CABLE
   (M (Value broadband-coaxial-cable))
   (ATTS (SATT
     (UNIT-PRICE (Value 1)) ;dollar/foot
     (MAX-ALLOWABLE-DISTANCE (Value 15000)) ;feet/amplifier
     (MAX-CHANNELS (Value 30))
     (MAX-DATA-RATE/CHANNEL (Value 5^6))
     (BITS-INDICATION
        (Value signal-frequency signal-strength))
     (BROADCASTING (Value one-way))
     (AUXILIARY-DEVICES (Value modem headend)) )
   (DPARA
     (LENGTH (if-needed ask)) ;feet
     (COST (Value (* length unit-price)))
     (DATA-RATE (Value
                   (* max-channels max-data-rate/channel)) )
     (PIX  (RESPONSE-TIME (Estimate (/ 1 data-rate))))
    ) ... )
```

REFERENCES

Anderson, G. A. and E. D. Jensen 1975, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," *Computing Surveys*, December, 7(4), pp.197-213.

Barker, Virginia E. and Dennis E. O'Connor 1989, "Expert Systems for Configuration at Digital: XCON and Beyond," *Communications of the ACM*, vol.32, no.3, pp.298-318, March.

Boose, J.H. 1985, "A Knowledge Acquisition Program for Expert Systems based on Personal Construct Psychology", *International Journal of Man-Machine Studies*, vol.20, pp.21-43.

Canon, C.M. 1974, "Decision Theory and Incomplete Knowledge", *Journal of Management Studies*, Vol.11, No.3.

Chang, C. L. and R. C. Lee 1973, *Symbolic logic and Mechanical Theorem Proving,* Academic Press, New York.

Duh, Jang 1987, *Realization of Distributed Experimental Frames in DEVS Scheme Environment,* Master Thesis, University of Arizona, Tucson, Arizona.

Enslow, P. H. 1978, "What is a Distributed Data Processing System?" *IEEE Computer*, 11(1), January, pp.13-21.

Eshelman, L. and J. McDermott 1986, "MOLE: A Knowledge Acquisition Tool That Uses Its Head", in *Proceeding AAAI* 86, 5th National Conference on AI, Philandephia, PA.

Eshelman, L. 1988, "MOLE: A Knowledge Acquisition Tool That Buries Certainty Factors," *Int. J. Man-Machine Studies,* no.29, p.563-577.

Flynn, M. J. 1966, "Very High Speed Computing Systems," *Proceedings of the IEEE*, vol.54, pp.1901-1909.

Forbus, K. 1988, "Intelligent Computer-Aided Engineering", *AI magazine*, vol.9, no.3, pp.23-36.

Gaines, B. R. 1987, "An Overview of Knowledge Acquisition and Transfer," *Int. J. Man-Machine Studies*, No.26, p.453-472.

Hart, Anna 1985, "The Role of Induction in Knowledge Elicitation," *Expert Systems*, no.2, pp.24-28, January 1985.

Hart, Peter E., Richard O. Duda, and M.T. Einaudi 1978, "PROSPECTOR - A Computer-based Consultation System for Mineral Exploration," *Mathematical Geology*, vol.10, no.5.

Hillis, D. 1985, *The Connection Machine,* MIT Press, Cambridge, Mass.

Hu, Jhyfang and Jerzy W. Rozenblit 1988, "Towards Automatic Generation of Experimental Frame in Simulation-Based System Design," *SCS AI Papers*, vol.20, no.1, pp.1-6.

Hu, Jhyfang and Jerzy W. Rozenblit 1989, "Knowledge Acquisition Based on Representation (KAR) for Design Model Development," to appear in: *Knowledge-Based Simulation: Methodology and Applications*, (ed. P. Fishwick and R. Modejeski), Springer-Verlag, NY 1989.

Hu, Jhyfang, Y.M. Huang, and J. W. Rozenblit 1989, "FRASES - A Knowledge Representation Scheme for Engineering Design," *Advances in AI and Simulation*, SCS Simulation Series, vol.20, no.4.

Hwang, Kai and F. A. Briggs 1984, *Computer Architecture and Parallel Processing,* McGraw-Hill, Inc.

Kahn, G., Nowlan, S. and J. McDermott 1985, "MORE: An Intelligent Knowledge Acquisition Tool", *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pp.581-584.

Kawato, N., Saito, T. and Sugimoto, H. 1985, "DDL/SX: A Rule-Based Expert System for Logic Circuit Synthesis," *IEEE Proc. ISCAS-85*, pp.885-887.

Kelly, G.A. 1955, *The Psychology of Personal Constructs,* New York: Norton.

Kessel, K. L. 1986, "Methodological Tools for Knowledge Acquisition," *Proceeding of the 1986 IEEE International Conference on System, Man and Cybernetics*, Atlanta, GA.

Kmietowicz, Z. W. 1981, "Decision Theory and Incomplete Knowledge," Gower Publishing Company Ltd., England.

Kowalski, T. J. and Thomas, D. E. 1985, "The VLSI Design Automation assistant: What's in a Knowledge Base", *IEEE Proc. 22nd Design Automation*

Conf., pp.252-258.

Kuratani, Y. 1986, "The Intelligent Decision Support System: Synthesis of a Decision Support System and an Expert System," preprints of VII-th International Conference on Multiple Criteria Decision Making, Kyoto, Japan.

Loo, William V. 1987, "Maximize Performance by Choosing Best Memory," *Computer Design*, Aug. 1, p.89-94.

Marcus, S. and J. McDermott 1986, *SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems,* Technical Report, CMU-CS-86-170, Dept. of Computer Science, Carnegie-Mellon University.

Marcus, S. and J. McDermott 1989, *SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems, Artificial Intelligence*, vol.39, no.1, May.

Marzollo, A. 1975, "On Some Broad Classes of Vector Optimal Decision and Their Characterization," reading: *Multicriteria Decision Making*, Spring-Verlag, New York, pp.281-323, 1975.

McDermott, John 1982, "R1: A Rule-Based Configurer of Computer Systems," *Artificial Intelligence*, vol.19, no.1, 1982.

Minsky, M. 1975, "A Framework for Representing Knowledge," in Winston, P.H. (ed.), *The Psychology of Computer Vision*, New York: McGraw-Hill, pp.211-277.

Mori, H. et al 1985, "WIREX: VLSI Routing Design Expert System," *Proc. IFIP Int. Conf., VLSI 85*, pp.297-306.

Newell, A. and H. A. Simon 1972, *Human Problem Solving,* Englewood Cliffs, NJ: Prentice-Hill.

Nilsson, N.J. 1980, *Principles of Artificial Intelligence,* Palo Alto, CA: Tioga Press; (Also Los Altos, CA: William Kaufman, 1983).

Olson, J. R. and Henry H. Rueter 1987, "Extracting Expertise from Experts: Methods for Knowledge Acquisition," *Expert System*, August 1987, Vol.4, No.3, p.152-168.

Osyczka, A. 1984, *Muticriterion Optimization in Engineering,* Ellis Horwood Ltd., England.

Paulin, P. G., Knight, J. P. and Circzyc, E. F 1986, "HAL: A Multiparadigm

Approach to Automatic Data Path Synthesis," *IEEE Proc. 23nd Design Automation Conf.*, pp.263-270.

Quillian, M.R. 1968, "Semantic Memory," in Minsky, M. (ed.), *Semantic Information Processing*, Cambridge, MA:MIT press, pp.27-70.

Rich, Elaine 1983, *Artificial Intelligence*, McGraw-Hill, Inc., New York.

Richie, I. C. 1984, "Knowledge Acquisition by Computer Induction," *Proceeding of UNICOM Seminar*, London, England.

Rozenblit, Jerzy W. 1986, *A Conceptual Basis for Integrated, Model-Based System Design,* Technical Report, Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona.

Rozenblit, J. W. and B. P. Zeigler 1988, "Design and Modeling Concepts," in *International Encyclopedia of Robotics Applications and Automation*, John Wiley and Sons, Inc.

Rozenblit, J. W. and Jhyfang Hu 1988, "Experimental Frame Generation in a Knowledge-Based System Design and Simulation Environment," In: *Modeling and Simulation Methodology: Knowledge System Paradigms*, (M. Elzas et. al., eds), North Holland, Amsterdam.

Schank, R.C. and Abelson, R.P. 1977, *Scripts, Plans, Goals, and Understanding*, Hillsdale, Lawrence Erlbaum, Hillsdale, NJ.

Scheifler, R. W. and J. Gettys 1986, "The X Window System", *ACM Transactions on Graphics*, vol.5, no.2, pp. 76-109.

Shastri, L. 1988, *Semantic Networks: An Evidential Formalization and its Connectionist Realization*, Research Notes in Artificial Intelligence, Morgan Kaufman Publishers, San Mateo, California.

Shortliffe, E. H. 1976, *Computer-Based Medical Consultations: MYCIN,* New York: American Elsevier.

Southard, J. R. 1983, "MacPitts: An Approach to Silicon Compilation," *IEE Comput.*, Vol.16, No.12, pp.74-82.

Stallings, William 1985, *Data and Computer Communications,* Macmillan.

Stallings, William 1987, *Local Area Networks: An Introduction,* Macmillan.

Sun Microsystems 1987, *Sun Network-extensible Window System (NeWS) Technical Overview and Reference Manual*, Sun Microsystems, Mountain View, California.

Tanimoto, Steven L. 1987, *The Elements of Artificial Intelligence, An Introduction Using LISP,* Computer Science Press, Rockville, Maryland.

Texas Instruments 1986, *TI Explorer Manual,* Dallas: Texas Instruments.

Waterman, D. A. and A. Newell 1971, "Protocol Analysis as a Task for Artificial Intelligence," *Artificail Intelligence*, 2, p.285.

Weste, Neil and Kamran Eshraghian 1985, *Principles of CMOS VLSI Design - A System Perspective,* Addison-Wesley Publishing Company.

Winston, Patrick Henry 1984, *Artificial Intelligence,* 2nd ed., Addison-Wesley Publishing Company, MA.

Winston, Patrick Henry and B.K.P. Horn 1984, *LISP*, 2nd ed., Addison-Wesley Publishing Company, Massachusetts.

Zeigler, B. P. 1976, *Theory of Modelling and Simulation,* Wiley, New York (Reissued by Krieger Pub. Co. Malabar, Fla. 1985).

Zeigler, B. P. 1984, *Multifacetted Modelling and Discrete Event Simulation,* Academic Press, London and Orlando, Fla.

Zeigler, B.P. 1986, *DEVS-Scheme: A Lisp-based Environment for Hierarchical, Modular Discrete Event Models,* Technical Report AIS-2, Department of Electrical and Computer Engineering, the University of Arizona, Tucson, Arizona.

Zeigler, B. P. 1987, "Hierarchical, Modular Discrete Event Modelling in an Object-Oriented Environment," *SCS Simulation*, vol.49, no.5, November 1987.

Zeigler, B. P. 1987, "Knowledge Representation from Newton to Minsky and Beyond," *Applied Artificial Intelligence*, no.1, pp. 87-107.

Zhang, G. and B. P. Zeigler 1989, "DEVS-Scheme Supported Mapping of Hierarchical Models onto Multiple Processor Systems," *Distributed Simulation 1989*, SCS Simulation Series, vol.21, no.2.