NIST Special Publication 500-188

Computer Systems Technology

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

Guide to Expert System Building Tools for Microcomputers

Christopher E. Dabrowski
Elizabeth N. Fong

NIST Special Publication 500-188

Guide to Expert System Building Tools for Microcomputers

Christopher E. Dabrowski
Elizabeth N. Fong

Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

July 1991

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director

Reports on Computer Systems Technology

The National Institute of Standards and Technology (NIST) has a unique responsibility for computer systems technology within the Federal government. NIST's Computer Systems Laboratory (CSL) develops standards and guidelines, provides technical assistance, and conducts research for computers and related telecommunications systems to achieve more effective utilization of Federal information technology resources. CSL's responsibilities include development of technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive unclassified information processed in Federal computers. CSL assists agencies in developing security plans and in improving computer security awareness training. This Special Publication 500 series reports CSL research and guidelines to Federal agencies as well as to organizations in industry, government, and academia.

National Institute of Standards and Technology Special Publication 500-188
Natl. Inst. Stand. Technol. Spec. Publ. 500-188, 147 pages (July 1991)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1991

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

PREFACE

The Computer Systems Laboratory (CSL) (formerly the Institute for Computer Sciences and Technology (ICST)) within the National Institute of Standards and Technology (NIST) has a mission under Public Law 89-306 (Brooks Act) to promote the "economic and efficient purchase, lease, maintenance, operation, and utilization of automatic data processing equipment by Federal departments and agencies." When a potentially valuable technology first appears, CSL may be involved in research and evaluation. Later on, standardization of the results of this research, in cooperation with voluntary industry standards bodies, may best serve Federal interests. Finally, CSL helps Federal agencies make practical use of existing standards and technology through consulting services and the development of supporting guidelines and software.

This report provides system managers, planners, and potential expert system developers with a readable description of expert system building tools for the microcomputer environment. Certain commercial software products and companies are identified in this report. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

ACKNOWLEDGMENTS

The technical work for this report was done through extensive review of published literature and hands-on analysis of commercial expert system building tools. We would like to thank the vendors for their cooperation and for their donation of these tools to the CSL Knowledge-Based Systems Laboratory.

We would also like to acknowledge the contributions of David Jefferson, Steven Ray, Daniel Benigni, Joseph Collica, and Bruce Rosen, of NIST who reviewed earlier versions of this paper and provided valuable advice. Candy Leatherman and Jennifer Lindeman proofread the document and helped with preparing figures and tables. Special thanks is also due to Steven Oxman, whose advice in preparing the final version of this report was particularly valuable. We also wish to express appreciation to Kenneth Reddy who provided programming assistance during the analysis and evaluation of tools.

ABSTRACT

Microcomputer-based expert system building tools (microcomputer-based ESBTs), sometimes known as expert system shells, are software packages for development of expert systems that run on microcomputers. This report provides system managers, planners, and potential expert system developers with a readable description of ESBTs for microcomputers including a detailed description of specific tool features and the capabilities they support. The technical content of this report is based on analysis of commercially available ESBTs. However, individual commercial products are not described, compared, or ranked.

Keywords: artificial intelligence; expert system; expert system building tool; expert system shell; inference; knowledge base; knowledge-based system; knowledge engineering; knowledge representation; microcomputer; object-oriented programming; production rule.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Recent years have seen substantial growth in the number of expert systems being developed and fielded in the microcomputer environment. This growth is due, to a great extent, to the advent of expert system building tools (ESBTs) designed for use on microcomputers. ESBTs, sometimes referred to as expert system shells, are special purpose software packages that are used to develop expert systems. Currently, there are a large number of commercial ESBTs available for the microcomputer environment.

While the structure and appearance of many of these products are similar, there are differences in individual tool features and the specific capabilities the features support. These differences influence selection and use of ESBTs for expert system projects. Selecting and using an ESBT requires both an understanding of how ESBTs are used and an in-depth knowledge of many of the features and capabilities of currently available tools. This report fulfills these needs by providing the following:

(1) A description of expert system technology in the microcomputer environment including an overall picture of expert systems development and use of microcomputer-based ESBTs.

(2) A detailed analysis of microcomputer ESBTs, including individual features and their capabilities.

(3) An identification and explanation of important trends and developments in this area of technology.

The report was compiled as a result of extensive review of published literature on expert system technology and ESBTs, through experiences in developing expert systems, and through analysis of commercial ESBTs.

1.1 Motivation

This report is motivated by a desire to provide a broad range of computer professionals, including project managers and potential application developers, with an in-depth understanding of microcomputer-based ESBTs. This motivation arises from important trends occurring in the development and utilization of expert systems in the microcomputer environment.

Expert systems are computer programs that store human problem solving knowledge, or expertise, and use it to solve difficult problems. As such, expert systems have proven to be a successful means of automating human expertise. Increasingly, organizations are beginning to see the value of developing and implementing expert systems on microcomputers. In the decade of the 1980s, microcomputers have become an inexpensive, reliable, and convenient

mechanism for providing computer processing power. The combination of expert system technology and microcomputers, used when and where appropriate, is an effective means of raising productivity of an organization. In recent years, the proportion of expert systems being developed on microcomputers has increased [WEIN90].

ESBTs provide some important advantages in developing expert systems. ESBTs provide a customized software development environment and a set of prepackaged software components for constructing individual expert system programs. Using an ESBT can allow an expert system to be developed more easily and quickly than would be possible using a programming language such as LISP or FORTRAN. However, this is true only if the capabilities of the ESBT match the requirements of the application being developed. Therefore, an in-depth understanding of ESBTs, as well as expert system technology in general, is an important prerequisite to the success of an expert systems development project. In many cases this understanding is lacking.

ESBTs have been commercially available only since the middle of the 1980s. Being a relatively new and still evolving area of technology, there is a lack of a thorough understanding about the capabilities and limitations of existing ESBTs in general and about those of microcomputer-based ESBTs in particular. This is especially true when it comes to understanding tool capabilities at the level of detail necessary to judge their usefulness for specific applications.

The lack of understanding is due, in part, to long-standing misconceptions about the nature of expert system technology. It is also due to the relatively small number of publications that provide detailed descriptions of ESBT features and their capabilities. Finally, understanding of ESBTs is made difficult by continuing changes taking place in this new technology. Nowhere is the pace of change more rapid than in microcomputer-based ESBTs.

1.2 Scope

The capabilities of microcomputer-based ESBTs cannot be fully understood apart from other areas of expert system technology. Therefore, it is important to provide a general discussion of expert system technology. The discussion seeks to address some important questions listed below.

- What are expert systems, and how do they work? How do expert systems differ from conventional software? What benefits are provided by expert system technology?

- What are appropriate applications for expert systems on microcomputers? What are inappropriate applications? What kinds of applications are most suitable for development using ESBTs? When should other kinds of software be used instead?

- What is the role of ESBTs in the development of expert systems? What kinds of programming skills are necessary to learn to use ESBTs? To what extent can ESBTs be used by non-computer professionals?

This general discussion will be of particular value to managers, planners, and others in organizations with an interest in expert system technology. For readers less familiar with expert system technology, the discussion provides background for a detailed analysis of individual features of microcomputer-based ESBTs.

The detailed analysis seeks to address the following relevant questions.

- What is the state of the art in microcomputer-based ESBTs? What are the limitations? What features are commonplace and can be expected in a commercial microcomputer-based ESBT?

- What features are currently being introduced in the microcomputer ESBT market? What new features, or enhancements in current capabilities, can be expected in the future?

- How can particular ESBT capabilities be used to fulfill specific requirements and implement specific functions of an expert system?

- How do individual features in ESBT development environment support development of expert system applications? What features are most, or least, important in development?

- Which ESBT features require well-developed computer skills to use? Which features can be used by non-computer specialists?

The detailed descriptions of features are also useful for formulating criteria to select ESBTs for specific projects.

1.3 Organization

The first five chapters provide the foundation for understanding expert system technology in the microcomputer environment.

- Chapter 2 contains an overview of expert systems concepts, including a description of how expert systems work. The benefits offered by expert system technology are discussed.

- In chapter 3, the operation of an expert system is illustrated by means of an example.

- Chapter 4 focuses on applications of expert systems in the microcomputer environment. The major kinds of expert systems that can be developed are described. Criteria for selecting expert system applications are discussed.

- Chapter 5 provides an overview of the expert systems development process and introduces the major classes of software used in development. The role of software for development of expert systems, including ESBTs, is discussed.

- In chapter 6, the characteristics of the ESBT architecture are presented. The major components of ESBTs are identified and described. Categories of ESBTs are introduced. The criteria used in analyzing microcomputer-based ESBT features are described.

The remaining chapters concentrate on detailed description and analysis of ESBTs. Each major component of the ESBT architecture is covered in a separate chapter.

- Features of application development environments provided by ESBTs are covered in chapter 7.

- Expert system methods for applying knowledge to solve problems through inference procedures are covered in chapter 8.

- Features used to represent and store expert knowledge are covered in chapter 9.

- ESBT features for constructing interfaces between expert systems and humans who use them are covered in chapter 10.

- ESBT features for constructing external interfaces for communicating with other software systems are covered in chapter 11.

Chapter 12 contains a discussion about guidelines for selecting ESBTs. Finally, chapter 13 presents conclusions and discusses future trends.

1.4 How to Read This Report

The development of an expert system can involve different groups in an organization, including managers, application developers, persons who provide expertise for the expert system, and users of the finished product. The use of ESBTs can therefore be examined from a number of different perspectives. Managers and others engaged in program planning may need background in expert

system technology and general information about ESBTs. Application developers with experience in expert systems development may wish to focus on detailed descriptions of features. Application developers with no experience in expert systems development may need both general background as well as detailed information on ESBT features and their capabilities. For these persons, here are some guidelines about how this report may be read.

- Many managers and program planners are in the process of learning about expert systems and are considering future uses of this technology. These people may be most interested in chapters 2-5, parts of chapter 6, and chapter 12.

- Other people in managerial positions may be members of organizations that are currently developing expert systems. These persons may wish to focus on application development and use of ESBTs. For these people, chapters 4, 5, and 6 will be of interest. Portions of chapters 7-11 that describe specific features of interest may also be read. Chapter 12 on selection of ESBTs should be of special benefit.

Developers of expert systems may be divided into several categories based on background and interests. Therefore, each may view expert system technology and ESBTs differently.

- Experienced expert systems developers may need in-depth knowledge about ESBTs. These persons should read chapter 7 to the end. Chapter 12 on selection of ESBTs may be of special interest.

- For computer specialists with little experience in expert systems or AI, the entire report may be read.

- In some cases, persons who are not computer specialists, but who have some knowledge about computers, may wish to develop expert systems. These persons may need to learn about both expert systems and ESBTs. For these people, chapters 2-5 should be read for background purposes. Selected portions of chapters 6-13 may also be of interest, particularly those that discuss ESBT features requiring little or no programming skill.

Figure 1.1 provides a "road map" to assist readers of this report.

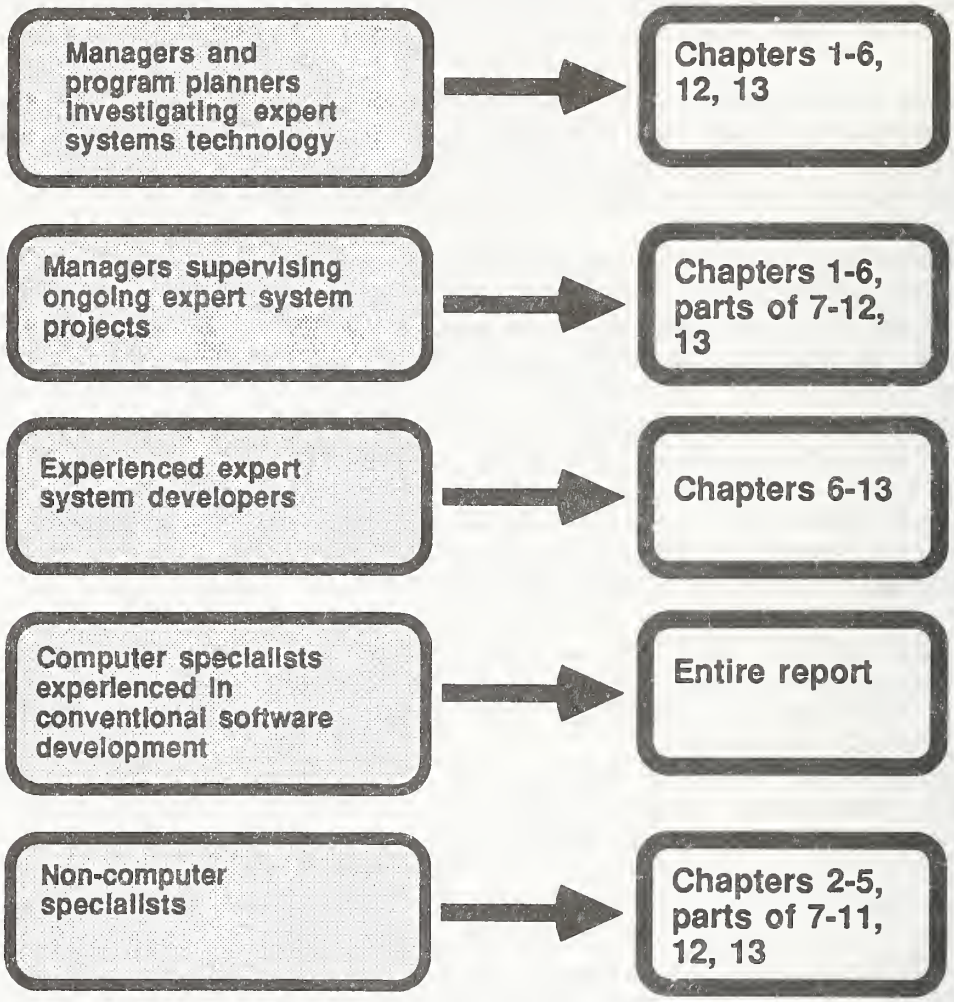

Figure 1.1. Road Map to Assist Readers of This Report.

2. EXPERT SYSTEM CONCEPTS

This chapter provides an introduction and overview of the main concepts underlying expert system technology. The methods employed by expert systems are described, and the features of a typical expert system architecture are summarized. The role of ESBTs is discussed. The next chapter will present an example of an expert system in operation and examine how this technology can be applied to solve real-world problems.

2.1 Background and History

Expert systems are a branch of the discipline of artificial intelligence. Artificial intelligence (AI) can be described as the study of theories and methods for automating "intelligent" behavior. AI has been a research discipline for over three decades, with other major subfields such as robotics, machine vision, and natural language understanding.

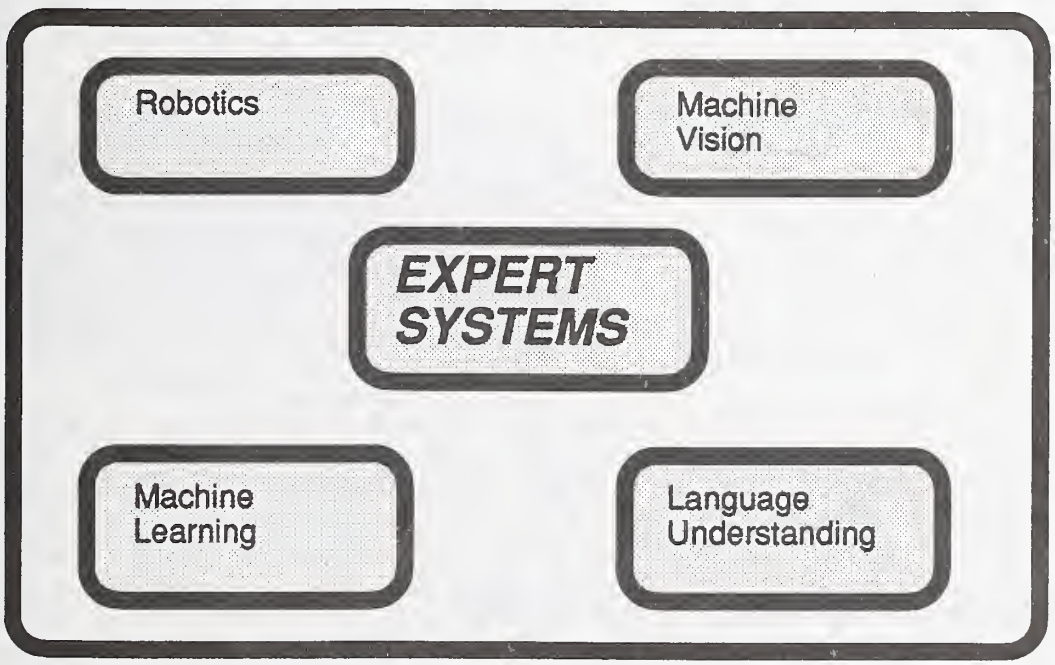

Figure 2.1. Expert Systems and Artificial Intelligence.

Research in expert systems began in the late 1960s and early 1970s. MYCIN, an experimental system for diagnosis of blood diseases [SHOR76], was an early demonstration of the potential of expert systems. The beginning of the 1980s saw the first practical applications of expert systems in real-world environments.

Since the middle of the 1980s, there has been a modest boom in the development of expert systems, both in government and industry. Initially, expert systems were perceived by some as holding forth the promise of revolutionizing the computer industry. Some hoped that expert systems could provide computers with a general capability for intelligent problem solving that could be applied to almost any problem. Such predictions proved to be misleading. As the technology matured, these views have gradually been corrected. More and more, expert systems are being realized for what they are: specialized software systems for automating expert problem solving for specific types of problems.

2.2 What Are Expert Systems?

An expert system is a computer program that provides assistance in solving difficult problems normally handled by human experts. An expert system stores knowledge about how a particular type of problem is solved. When an example of the problem is presented, the expert system uses the stored knowledge to find a solution.

Expert system programs differ from conventional computer programs. Conventional programs are designed to solve problems for which all the factors used in the decision-making process can be completely analyzed. Typically, this analysis can be expressed in an algorithm that, when its steps are executed, will arrive at a correct solution. A good example is a payroll calculation program. Algorithms for performing such a task can be encoded in a conventional programming language.

In contrast, expert systems are aimed at problems that cannot always be solved using a purely algorithmic approach. These problems are often characterized by irregular structure, incomplete or uncertain information, and considerable complexity. In such cases, the method of problem solving either cannot easily be expressed in algorithmic form or algorithms are altogether unknown. Solutions must be obtained by reasoning from available evidence and sometimes making subjective "best guesses." Examples of problems with these characteristics are medical diagnosis, equipment troubleshooting, and database design.

The ability to solve such problems requires considerable knowledge or expertise about a specific domain of endeavor. Typically, such ability is possessed by only a few human experts, who are referred to as <u>domain experts</u>.

2.3 The Role of Heuristic Knowledge

Much of the knowledge of domain experts in solving practical problems consists of heuristics acquired through learning and experience. A heuristic is a rule of thumb, fact, or even a procedure that can be used to solve some problem, but it is not guaranteed to do so. It may fail. Heuristics can be conveniently regarded as simplifications of comprehensive formal descriptions of a real-world system.

For example, it is conceivable that all aspects of the operation of a machine could be completely described in a complex physical or mathematical model, including circumstances under which machines malfunction. In principle, this model could be used to analyze machine problems and (algorithmically) determine malfunctions with virtual certainty. In practice, complete models are often difficult to develop due to lack of necessary information about the problem and its inherent complexity. Therefore, for many problems, domain experts find it practical and necessary to substitute heuristic knowledge for a complex model.

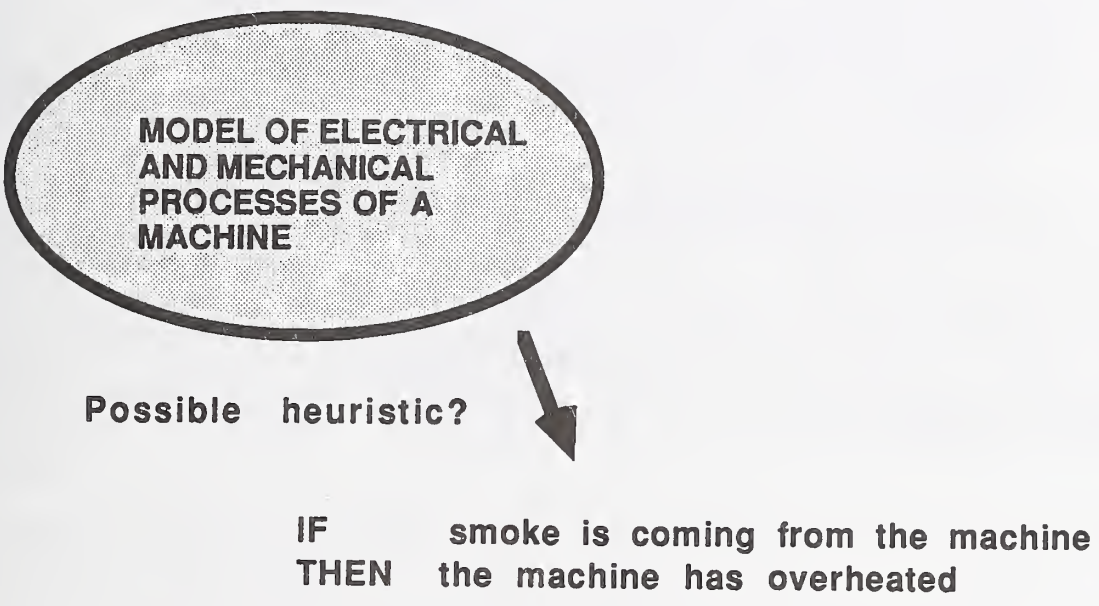

Figure 2.2. An Example of a Heuristic.

Though expert system programs do utilize conventional techniques, their overall problem-solving strategy is driven by application of heuristic knowledge. Expert systems can be thought of as software systems that (1) maintain heuristic knowledge about solving certain kinds of problems and (2) apply this knowledge to solve specific instances of the problem, usually using some form of automated inference. The expert system's problem-solving ability is a function of the quality and quantity of knowledge it internally represents and uses. To internally represent knowledge and carry out inference, expert systems utilize specialized programming techniques developed from AI research.

It should be pointed out that an expert system can seldom completely replace a human expert. Rather, its knowledge base contains knowledge about solving the most commonly encountered problems. Furthermore, the expert system's problem-solving ability cannot usually be extended to solve problems that are outside the scope of its knowledge.

2.4 The Elements of an Expert System

Expert systems store expert knowledge and apply it "on demand" to solve problems. Most often the user of an expert system is a person. The user may also be another software system or even a mechanical device. A human user, known as an end user, usually provides information to the expert system via a computer terminal. The expert system uses inference procedures to apply its stored knowledge to the facts describing a problem. The systematic application of inference leads to solutions that are then displayed at the terminal.

The operation of an expert system can be viewed in terms of the interaction of distinct components. The knowledge base stores knowledge about how to solve problems. Inference procedures are executed by a software module called the inference engine. If the user of the expert system is a person, communications with the end user are handled via an end user interface. Figure 2.3 provides a graphical illustration that summarizes the architecture of a typical expert system.

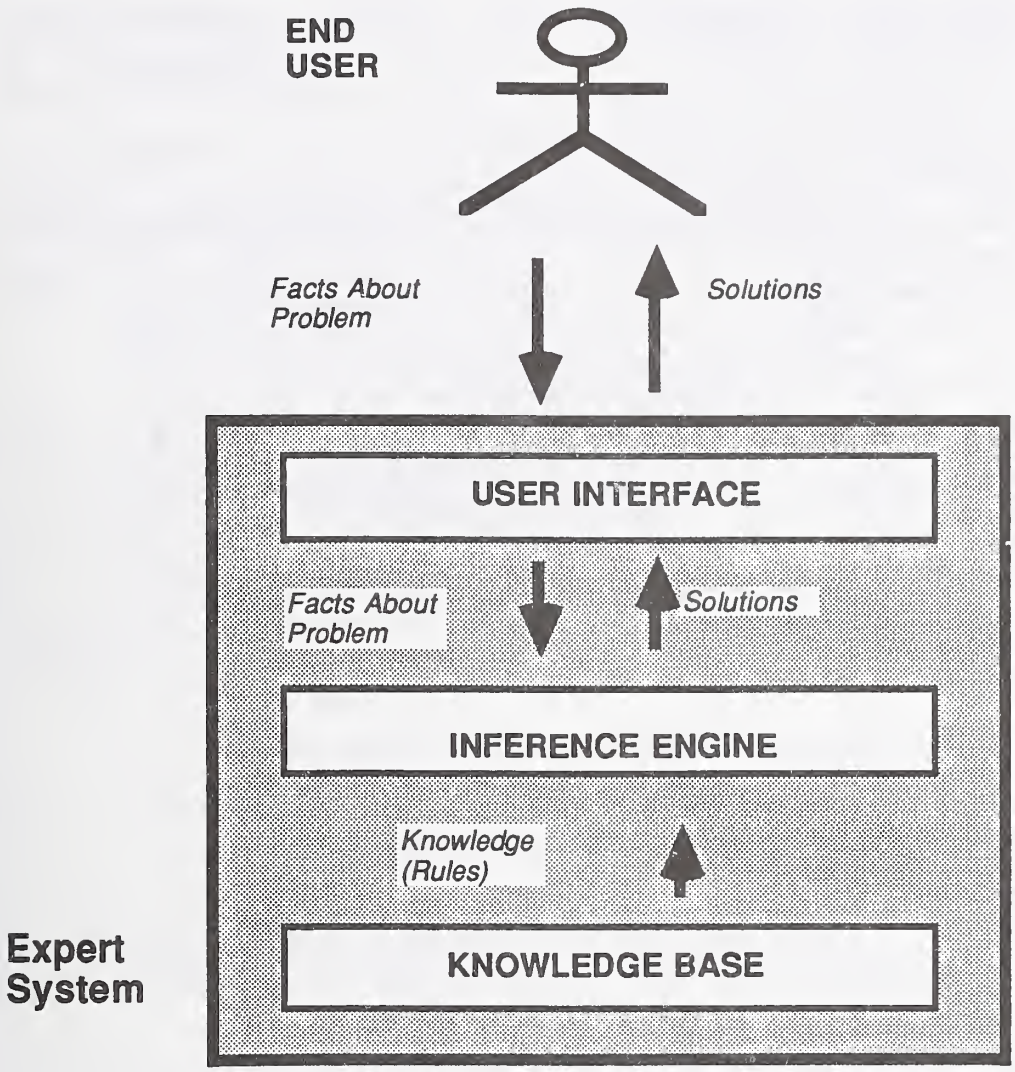

Figure 2.3. The Architecture of an Expert System.

If the user of the expert system is another software system instead of a person, different external interfaces will be required.

Each of the major parts of the expert system architecture can be further elaborated.

2.4.1 The Knowledge Base

Knowledge is stored in the knowledge base using symbols and data structures to stand for important concepts. The symbols and data structures are said to represent knowledge. Knowledge

representation can take many forms. The most common form is the production rule, shown below:

```
IF   MACHINE Is Smoking
THEN MACHINE Is Overheating.
```

Production rules are a particularly convenient way of expressing heuristic knowledge. The overall scheme for organizing and representing knowledge is sometimes called a knowledge representation system. The knowledge base refers to the actual store of knowledge for a particular expert system.

A knowledge representation system may be simple, consisting only of data structures for representing rules. Or knowledge representation may incorporate other more complex structures, which are discussed later in the report. Knowledge represented in data structures, such as rules, is said to be stated declaratively. Declarative knowledge is knowledge that is stated explicitly and is intended to be accessible to persons who may need to see it, such as domain experts. The ability to make its declarative knowledge accessible and understandable is one of the most important services provided by a knowledge representation system.

Declarative knowledge representation states what is known about how to solve a problem. As such, this knowledge is static and remains the same, as the expert system solves different problems. However, an expert system must also store information about the individual problems as it works on them. This information is maintained in a structure sometimes called a context file [MART88]. Information in a context file is specific to the problem the expert system is working on and changes as new information is acquired.

2.4.2 The Inference Engine

The inference engine is a software module that executes procedures for applying knowledge to produce new information about a problem. In production rule systems, an inference engine compares rules against known facts in the context file to determine if new facts can be inferred. The conditions in the premise, or IF part, of a production rule are compared against known facts. If these conditions are satisfied, the facts in the conclusion, or THEN part, can be inferred. The newly concluded facts are then added to the context file of the expert system. Figure 2.4 shows how inference takes place.

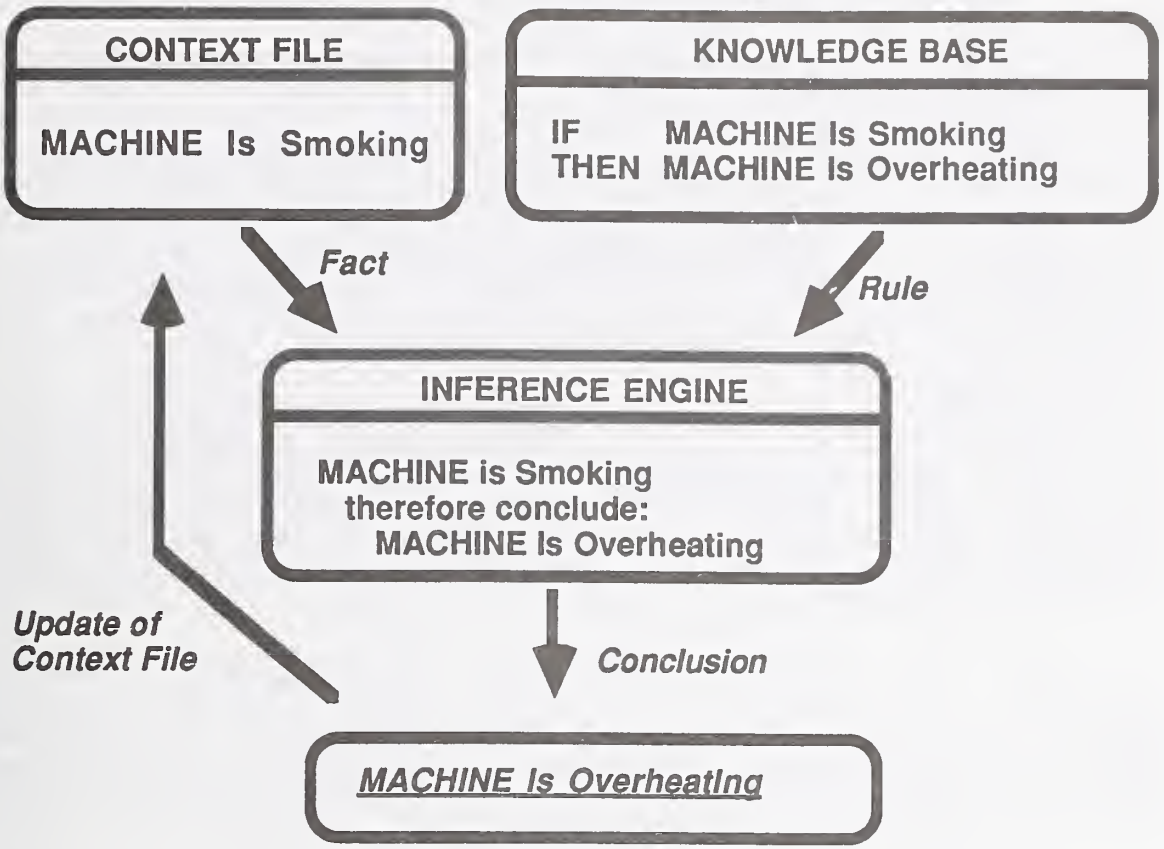

Figure 2.4. An Example of Using a Rule to Do Inference.

The process of inference is sometimes described in terms of reasoning with symbols, or reasoning symbolically. Symbolic reasoning refers to the manipulation of symbols that takes place during the process of inference. Symbolic reasoning is intended to emulate, albeit crudely, the way humans might manipulate concepts and ideas when reasoning. In the example above, symbolic reasoning is very simple, consisting of a simple comparison and update to a database of facts. Other forms of symbolic reasoning are more complex.

To solve a specific instance of a problem, a large number of rules may have to be examined. An inference engine uses an inference strategy to guide the order by which rules are examined and inferences made. Inference strategies are an important aspect of expert systems. Inference strategies provided by ESBTs are discussed more fully in chapters 8 and 9.

2.4.3 Expert System Interfaces

Expert systems communicate with human users as well as other software and hardware systems.

Expert systems communicate with human users via an end user interface. The purpose of the end user interface is to obtain information about the problem from the end user and to display solutions. To obtain information, the interface may display questions at a terminal and prompt the end user for answers. Solutions may consist of text statements. More elaborate end user interfaces may use graphics and hypertext.

A useful function of an expert system is the ability to explain its actions. While using an expert system, the end user may wish to know why questions are being asked or why certain facts were concluded. When the solution is displayed to the end user, the user may request an explanation of how the solution was reached. The end user interface contains procedures that generate explanations that can be shown to the end user.

In many practical applications, an expert system must interface, and exchange data, with other software and hardware systems. The number of expert systems that have nonhuman users, such as other software systems or electronic process control devices, is increasing. External interfaces between expert systems and other systems are discussed further in chapter 3.

2.5 Differences Between Expert System Programs and Conventional Computer Software

Expert system programs differ from conventional software in four important ways. First, knowledge is separated from program control; i.e., the knowledge base and inference engine are separate. Second, knowledge is represented declaratively. Third, expert systems perform computation through symbolic reasoning. And finally, expert systems can explain their actions. Each of these distinguishing capabilities are elaborated below.

- **Separation of Knowledge From Program Control**

  In conventional programs written in languages such as FORTRAN or C, knowledge about a problem domain is contained in programming language statements. As such, knowledge about how a problem is solved is combined directly with specifications for control of program execution. In an expert system, knowledge is represented in data structures, such as production rules, which are stored in a knowledge base. The knowledge base is separate from the inference engine, which controls program execution.

- **Declarative Representation of Knowledge**

  Knowledge represented using symbols and data structures, such as rules, is explicit in the sense that it states what knowledge exists, not how the knowledge is applied. By representing knowledge declaratively, the knowledge of an expert system can be more readily understood and accessed by individuals who are not specialists in programming languages.

- **Symbolic Reasoning and Inference**

  Symbolic reasoning refers to the manipulation of symbols and data structures by the inference engine. Symbolic reasoning is intended to emulate the way humans might manipulate concepts when solving a problem. In the simplest case, this may involve testing a rule to determine if its IF part is satisfied. If so, the conclusion of the rule may be added to the expert system's context file. Expert systems can also be characterized by use of defined inference strategies. These strategies are more fully described in chapters 8 and 9.

- **Explanation of Actions**

  Declarative knowledge and symbolic reasoning support the ability of an expert system to explain its actions by showing the chain of reasoning created by the rules used to solve a problem. An example is provided in the next chapter.

In addition to these capabilities, some expert systems represent knowledge that is uncertain as well as apply this knowledge to uncertain data. Reasoning under uncertainty, as this capability is known, can be used for heuristic knowledge. ESBT support for reasoning under uncertainty is discussed in chapter 8.

ESBTs are designed to support the development of software systems that require the capabilities described in this chapter.

2.6 The Role of Expert System Building Tools

To develop an expert system, it is necessary to implement the major components of the expert system architecture described in section 2.3. This work is often made easier by using an ESBT.

ESBTs provide a customized software development environment and a set of prepackaged software components, or building blocks, for implementing individual expert system applications. The development environment supports construction of the expert system program and includes facilities for entering the expert's knowledge into the knowledge base. The building blocks supplied by the ESBT are derived directly from the expert system architecture. They include:

- A knowledge representation system for creating a knowledge base.
- The inference engine.
- Software components for constructing interfaces between the expert system and its external environment.

ESBTs are often referred to as "shells." An expert's knowledge is stated declaratively. It is therefore distinguishable and separable from the knowledge representation system provided by the ESBT. The expert's knowledge may be inserted in, and removed from, the ESBT at the discretion of the developer. This is why ESBTs are referred to as "shells."[1]

ESBTs greatly speed implementation of individual expert systems. By providing prepackaged components, the ESBT spares the developer the necessity of programming a large part of an application from scratch. ESBTs can thus be distinguished from conventional programming languages and from artificial intelligence languages such as LISP or Prolog precisely because they provide a development environment and the building blocks necessary to support construction of expert systems.

---

[1]In this report, the term ESBT will be used instead of "shell." The term ESBT conveys the idea of a more comprehensive development tool for expert systems. This is consistent with ongoing developments in commercial microcomputer-based ESBTs.

3. EXPERT SYSTEMS IN OPERATION

This chapter describes how expert systems are employed in an operational environment. First, the circumstances under which an expert system might be used to solve real-world problems is illustrated. This is followed by an example of how an expert system solves a problem. The example reinforces the concepts presented in the preceding chapter. Then, this chapter discusses the increasingly important subject of expert system interfaces to other software systems, including issues pertaining to the integration of expert systems in larger computing environments.

3.1 The Need for Expert Systems

In many organizations, problem-solving expertise is scarce. Training people to become proficient in solving specialized problems takes time and requires substantial investment. Hence, experts are always in short supply. In an operational setting, the frequency with which problems occur often exceeds the capabilities of a limited number of experts. In some situations, experts may be geographically distant from the site of the problem, or problems may occur during times experts are unavailable. Consequently, problems must be handled by less qualified personnel. This can cause delays and lead to inconsistent or uneven decision making.

One example is a factory floor in which highly specialized equipment problems must be diagnosed and repairs effected. Normally, this task is performed by a trained expert having years of experience. Work proceeds normally if the frequency of malfunctions is relatively low and the expert is readily available. However, a company may have factories at different locations, or the factories may operate around-the-clock. Under these circumstances, demand for experts may quickly exceed their availability, resulting in delays and problems.

One solution is to develop an expert system to help identify frequently occurring machine malfunctions and suggest solutions. Such an expert system could be deployed on the factory floor and used to solve routine problems that would otherwise have to be handled by the expert. If the expert system could solve 80% of the problems the expert normally solves, delays could be eliminated and productivity improved. Copies of the expert system could be distributed throughout the company, making expertise available at several locations around the clock.

This is a simplified description of the productive use of an expert system. Certainly, there are potential problems in this scenario that could defeat effective use of expert system technology. Chapter 4 discusses criteria for selecting expert system applications for the microcomputer environment. But first, let's see how an expert system might diagnose a machine problem.

3.2 An Example of an Expert System

This section expands on the equipment failure example introduced above to illustrate an expert system in operation. Let us assume a simplified knowledge base, consisting of the rules depicted in figure 3.1 below.

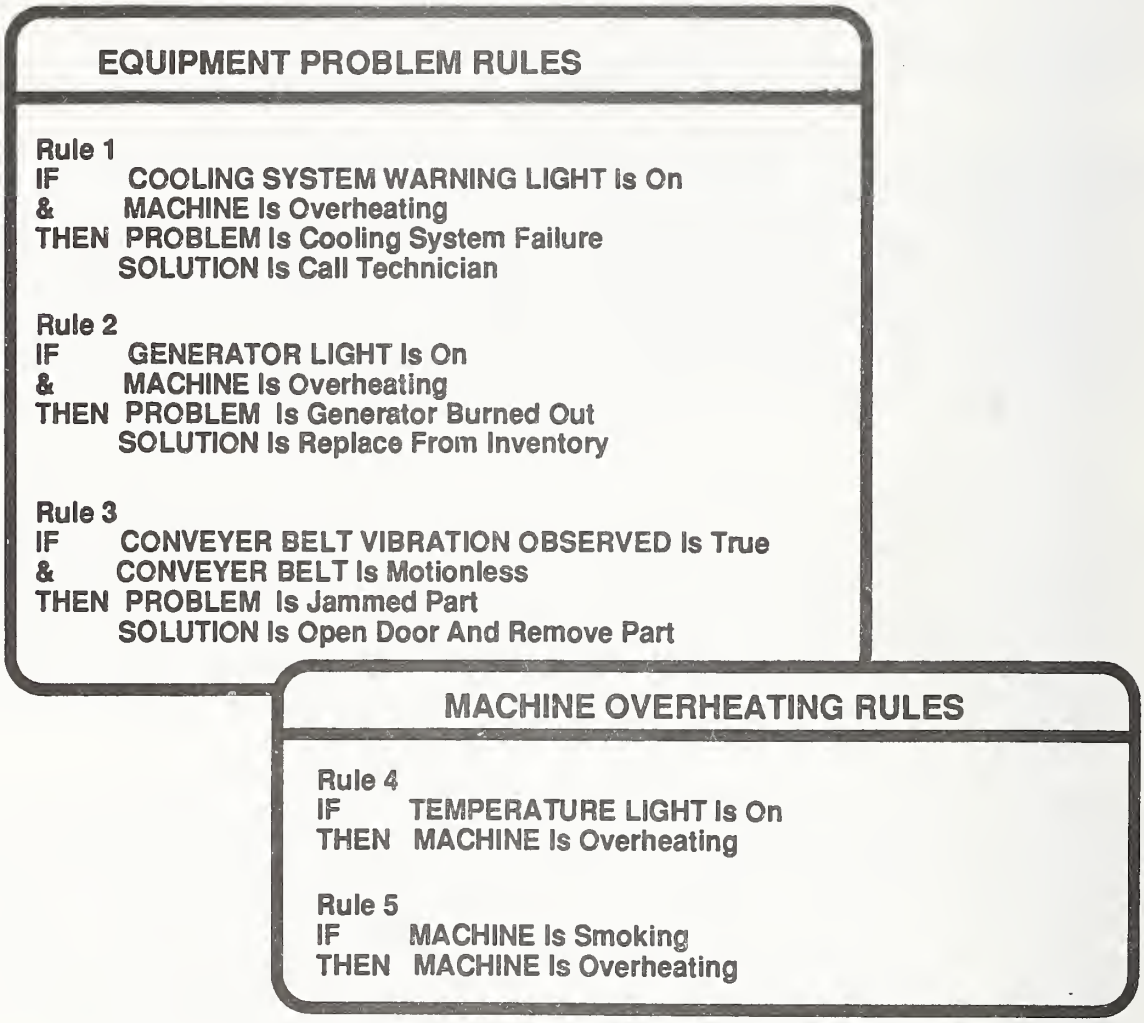

Figure 3.1. A Simple Knowledge Base.

A problem-solving session is initiated when a machine on the factory floor is observed to be malfunctioning. Let us assume that the following facts are observed: smoke is coming from the machine, and the generator light is on. To find out what has gone wrong, a factory employee sits down at a computer terminal and activates the expert system program.

3.3 How an Expert System Solves a Problem

A common problem-solving approach used by an expert system is to propose a solution and then attempt to prove it. In this case, possible causes of the machine's problem (and appropriate remedies) may be proposed and examined. To determine the cause, or causes, the inference engine accesses the knowledge base and systematically applies its rules to facts about the problem. These facts may be obtained from the end user or external software system and stored in the expert system's context file.

3.3.1 Use of an Inference Strategy

Many expert systems follow this problem-solving approach by using a strategy for inference known as backward chaining. (Forward chaining and other possible strategies are described in detail in chs. 8 and 9.) Backward chaining begins by first attempting to infer a solution using rules that conclude the cause of the malfunction (Rules 1, 2, and 3 in fig. 3.1). The inference engine attempts to satisfy the conditions of these rules with facts that are either available in its context file or that can be inferred. When a needed fact can be inferred by another rule, its conditions are, in turn, examined. The strategy thus continually "works backwards" from a rule's conclusion to its conditions, and "chains" to other rules that infer needed facts.

3.3.2 Arriving at a Solution

The sequence of steps below (illustrated graphically in fig. 3.2) shows how backward chaining is used to find the cause of the malfunction.

Step 1. The inference engine attempts to prove that the problem is in the cooling system by satisfying the conditions of Rule 1. The end user is asked if the cooling system warning light is on. The user replies no, causing Rule 1 to fail.

Step 2. The inference engine then tries to satisfy the conditions of Rule 2. In this case, the first condition is satisfied when the user replies that the generator light is

on. The inference engine must then determine if the machine is overheating, the second condition of Rule 2. Rules 4 and 5 can be used to infer that the machine is overheating. Therefore, the inference engine "backward chains" to each of these rules.

Step 3. Rule 4 fails because the temperature light was not on.

Step 4. Rule 5 is tried. The user is asked if the machine is smoking and replies yes, thus satisfying the single condition of this rule. The inference engine can then conclude that the machine is overheating and add this fact to the expert system's context file.

Step 5. Rule 2 is reexamined. Since its conditions are now satisfied, the inference engine concludes that the source of the problem is the generator, thus providing a solution to the problem. The conclusion is also added to the context file.

Step 6. This conclusion, together with the advice to check the inventory for a replacement part, is displayed at the terminal via the end user interface.

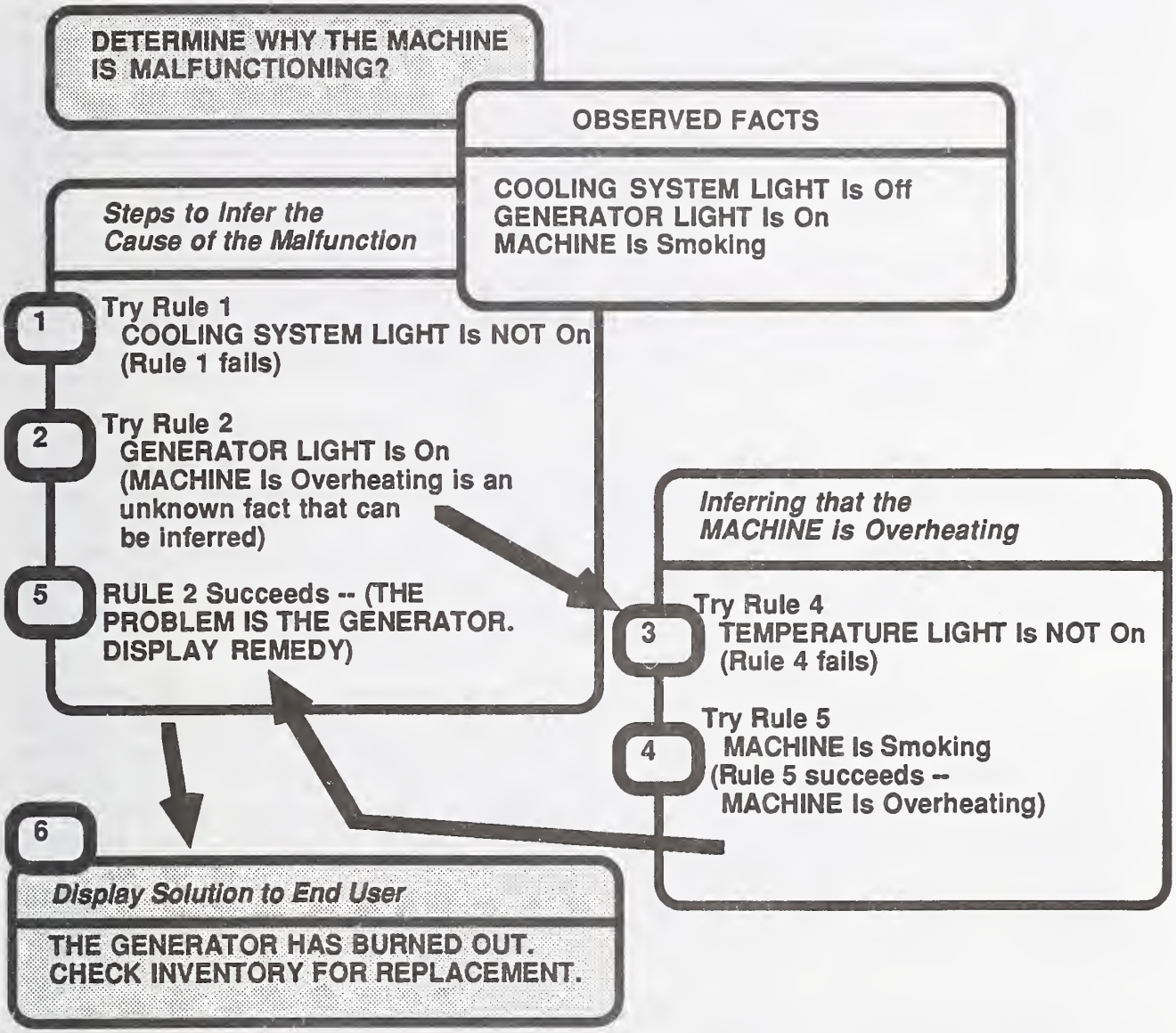

Figure 3.2. The Sequence of Inference.

This simple example is meant to give the reader some idea of how an expert system works. Actual expert systems may have hundreds or even thousands of rules.

3.4 Expert Systems and Problem Complexity

The difficulty of many real-world problems tackled by expert systems can often be described in terms of (1) the large number of facts that can potentially be examined, (2) the many different solutions that can be found, or both. Examining the consequence of each fact or combination of facts and investigating every possible solution could result in following many alternative lines of reasoning. The systematic examination of different lines of reasoning in an effort to find a solution can be viewed as a search process. For very large problems, exhaustively searching every possibility might be impractical. Solving such problems thus depends on reducing their size.

In many cases, the expert limits the size of the problem by focusing on a small subset of facts and a few relevant lines of reasoning. This allows the expert's knowledge to be captured in a succinct set of rules. The resulting "expert systems" are small, and the knowledge they contain can be expressed in decision tables or decision trees.[2]

For larger, more complex problems, finding a solution requires examining considerably more information and/or considering a much larger number of solutions. To reduce the size of the problem, an expert system may have to select only a few lines of reasoning to pursue from a much larger number that may be available. Heuristics provided by the expert are used to select those lines of reasoning with the best chance of producing a solution.

The use of heuristics to limit search is known as heuristic search. One way heuristics can reduce search is to shift the focus of the problem-solving effort on the most important evidence or the most likely solutions. In the example above, the domain expert may provide the knowledge that the machine is more likely to overheat than it is for the cooling system to fail. Hence, the expert system could reduce the amount of work it has to do by trying Rule 2 first. Other lines of reasoning would be tried if it was found that the machine had not overheated. In a larger expert system, heuristic search could be an important factor.

Search is an important area of research in AI [BARR81], [FOX90], [RICH91], [WINS84]. ESBTs provide direct support for only a subset of search techniques that have been developed. This topic is discussed in chapter 8.

---

[2]Such systems are often distinguished from expert systems because of their simplicity and small size. Some specialists prefer to call these systems knowledge systems.

3.5 How Expert Systems Explain Their Actions

Explanation is a helpful capability in understanding how a complex expert system works. For instance, the end user may request an explanation of the line of reasoning used to identify the source of the problem. Figure 3.3 shows how an expert system may display its line of reasoning.

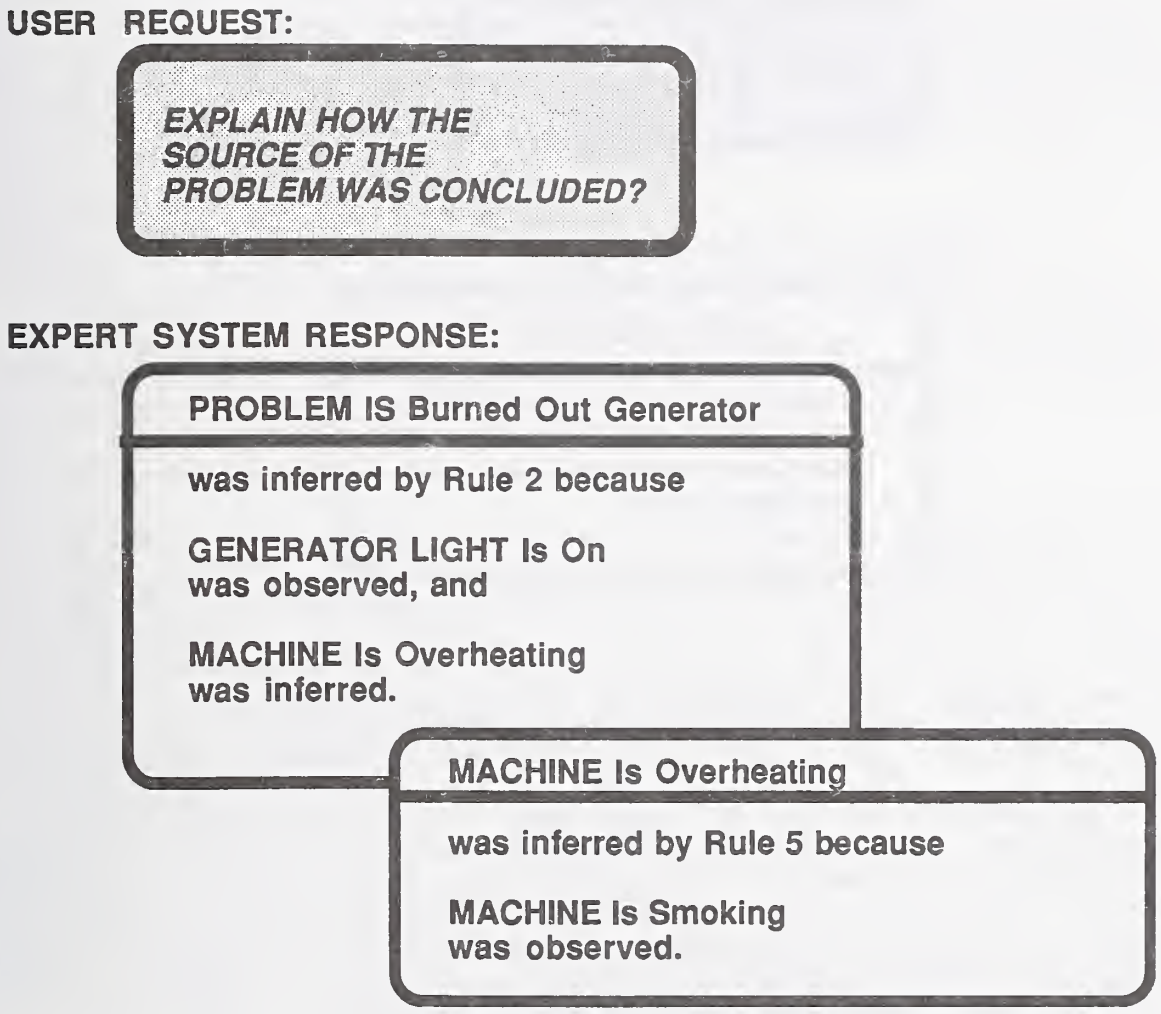

Figure 3.3. Explaining How a Conclusion Was Reached.

As the figure shows, explanation of concluded facts is accomplished by essentially "playing back" the sequence of inferences previously made by the expert system.

Similarly, an end user may not understand certain terms and phrases used by the expert system. For instance, the end user may

not understand what a Generator Light is, or where to find it. An expert system can allow the end user to interrupt execution of the system and request an explanation of a term. Figure 3.4 shows an explanation of what a Generator Light is and where it is located.

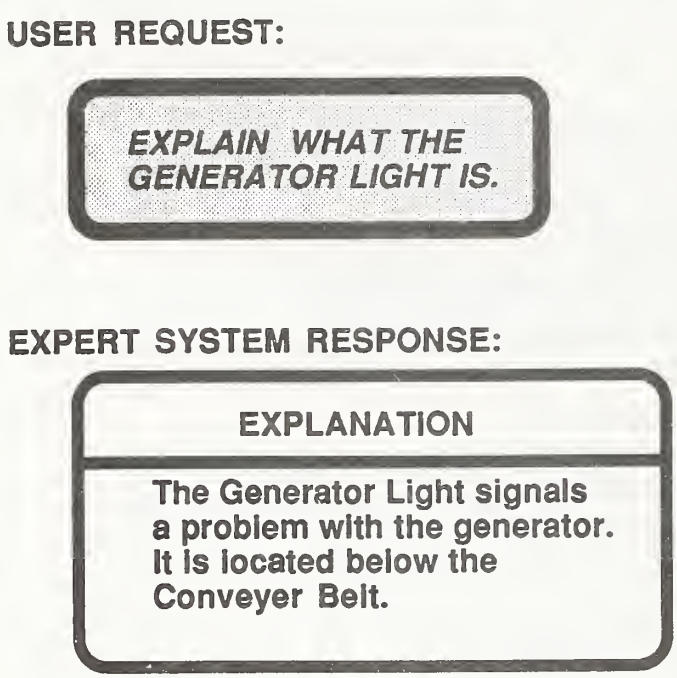

Figure 3.4. Explaining the Meaning of a Term.

Explanation can be an important capability provided by an expert system. In addition to explaining solutions to end users, explanation can be used to verify that the expert system is working correctly. For these reasons, ESBTs provide explanation facilities that can be used during the development of an expert system application and incorporated into the finished product. These facilities are discussed in chapters 7 and 10.

3.6 Interfacing Expert Systems to Other Software

Early expert systems, for the most part, were stand alone computer programs that did not interface with other software systems. However, this has changed in recent years. As more and more corporate information resources become computerized and as the demand for expert systems increases, the necessity and importance of expert system interfaces have also increased. Many problem-

solving tasks performed by expert systems now require communication and exchange of data with other software systems.

External interfaces must therefore be considered an important component of the overall expert system architecture. Expert systems may require a variety of external interfaces. A few are summarized below.

- **Interfaces to Database Management Systems**

  Many problem-solving tasks require accessing data residing in a DBMS. For instance, in the example above, the expert system might check an inventory database to find out if replacement parts are available. The expert system could initiate a call to a DBMS or access an ASCII file.

- **Computer Network Access**

  Many problem-solving tasks require interfacing to an organization's DBMS or special-purpose software system that is located on a large mainframe computer. Microcomputer-based expert systems may access these systems via a local area network (LAN) or other remote computing system.

- **Interfaces to Procedural Computer Programs**

  While expert systems carry out inference procedures well, they are not as good at performing other computations. For instance, inference procedures are inefficient for extensive numerical calculations or iterative processing. If such computations are required, the expert system should initiate calls to external software modules written in languages such as C or FORTRAN. The external modules can then perform the necessary computations and return the result. In some cases, ESBTs provide libraries of mathematical functions that can be used instead.

- **Graphics and Hypertext Packages**

  Interfaces to graphics software packages may be necessary for some expert system applications. Graphics packages are important aspects of engineering software such as computer-aided design (CAD) systems. Expert systems to perform tasks in the engineering domain have proven valuable. Hypertext systems enhance an expert system's explanatory capability and provide links to text information systems.

- **Other Commercial Software Packages**

  This includes spreadsheet programs and a variety of other special-purpose commercial software products.

Figure 3.5 summarizes the various types of interfaces.

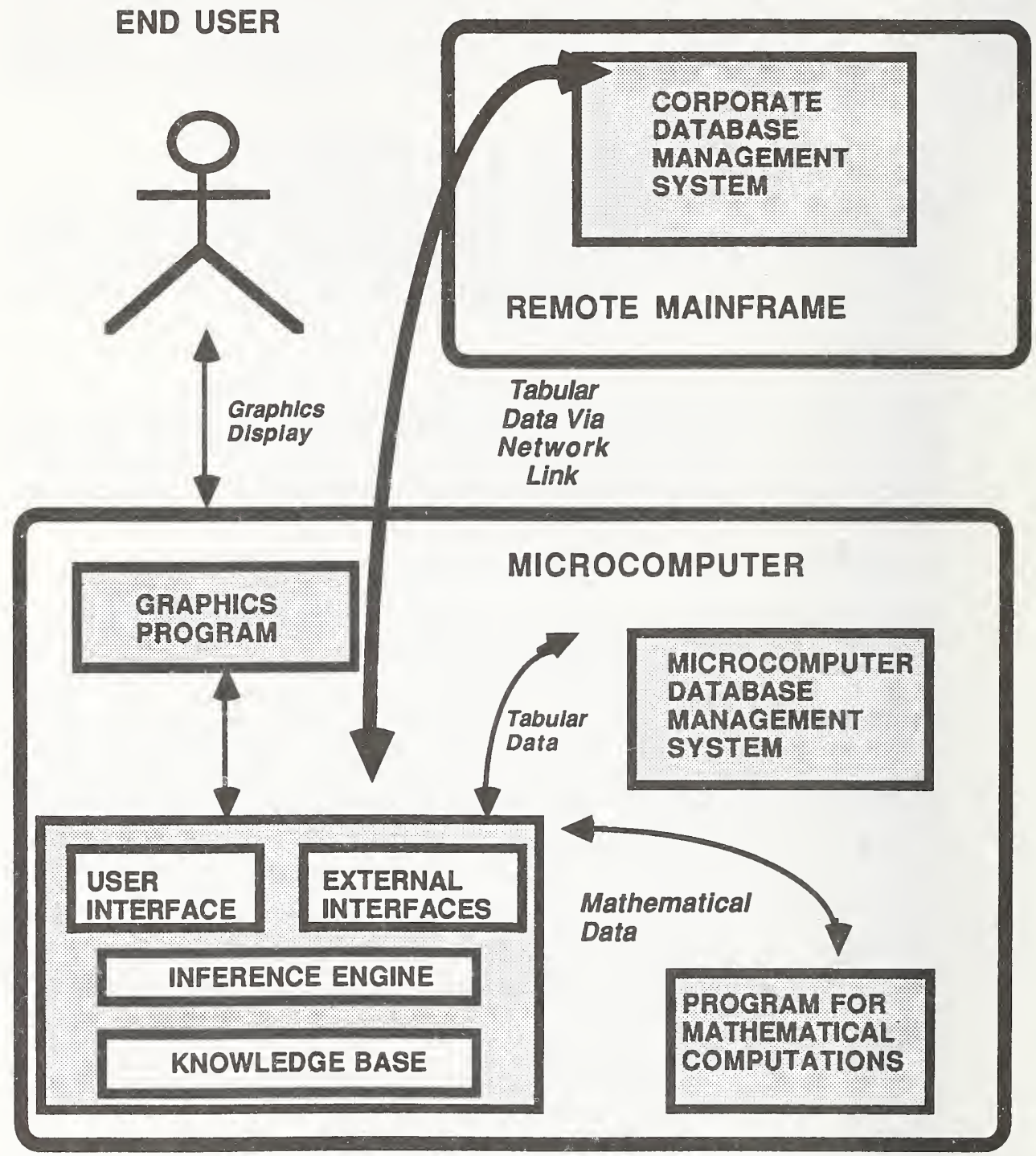

Figure 3.5. Expert System Interfaces.

Because of the important links between expert systems and other software systems, ESBTs should provide mechanisms to construct external interfaces. ESBT support for construction of external interfaces is discussed in chapter 11.

3.7 Embedded Expert Systems

In addition to expert systems that initiate calls to other software systems, it is possible that conventional software applications may initiate calls to expert systems to perform special-purpose problem solving. Increasingly, expert systems are being developed and configured as components in large complex software applications. These systems are referred to as embedded expert systems. In contrast to the example presented in section 3.2, the primary user of an embedded expert system is another software application. Figure 3.6 below illustrates this concept.

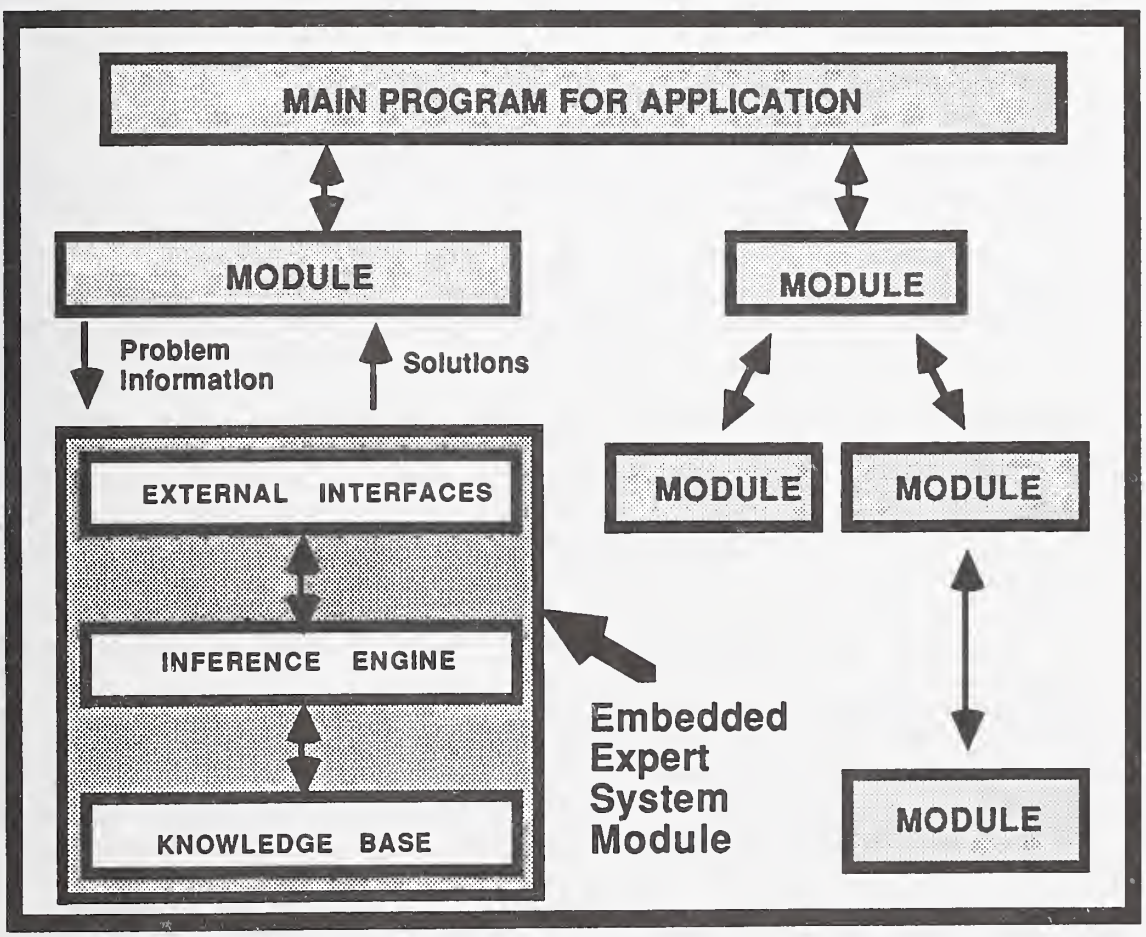

Figure 3.6. An Embedded Expert System.

Embedded expert systems are becoming increasingly important. ESBT facilities for constructing embedded expert systems are also discussed in chapter 11.

4. APPLICATIONS OF EXPERT SYSTEM TECHNOLOGY

Expert system technology can be used to automate certain kinds of tasks. For expert systems technology to be applied effectively, there must be a benefit to the organization in doing so. Also, certain factors and identifiable characteristics must generally be present in the problem to be solved. Not all tasks can be or should be implemented as expert systems. Some tasks rely on knowledge that cannot be easily captured by expert systems, while other tasks are more suitable for implementation using alternative computing methods.

This chapter begins with a discussion of the different types, or classes, of expert systems that can be implemented using ESBTs. Then, criteria for selecting applications of expert system technology are discussed. Characteristics of tasks that are appropriate for implementation as expert systems are presented. Finally, tasks that are not appropriate for development as expert systems are also discussed.

4.1 Classes of Expert System Applications

What kinds of expert systems can be developed? What kinds of tasks can they perform? To answer this question, it is helpful to categorize expert systems on the basis of application type.

Expert systems employ two broad categories of problem solving strategies that provide a basis for describing classes of expert system applications. The two strategies are solution selection and solution construction.

The two strategies can differ in terms of program size, program complexity, and amount of processing required. These differences influence which kinds of expert systems can best be implemented on microcomputers as well as the characteristics of the microcomputer-based ESBTs that are used.

4.1.1 Expert Systems That Select Solutions

The most basic strategy employed by expert systems is to select solutions from among a larger list of possible outcomes. Simple versions of such systems may work by essentially matching facts describing a problem to individual solutions. Rules can be used to directly infer individual solutions with little or no chaining.

More complex problems require use of abstract problem characterizations as intermediate steps in finding a solution. The expert system examines supporting evidence in an attempt to characterize, or classify, a problem according to abstracted

descriptions of problem types. The characterizations are then used to infer solutions. The abstract characterizations and rules of inference are based on heuristics obtained from domain experts. This method, often called heuristic classification [CLAN86], forms the basis for the way many expert systems solve problems. Section 3.2 presents a simple example of such a system.

Several kinds of expert system applications that select solutions using heuristic classification may be identified. A few are listed and discussed below.

- **Recognition of System Malfunctions**

  Determining causes of malfunctions is a generic task commonly performed by expert systems. Examples include medical diagnosis and troubleshooting equipment failure.

- **Evaluation of Applications for Benefits**

  These systems determine eligibility for credit applications, insurance claims, and pensions. There are several examples of such systems implemented on microcomputers. See [HARM88].

- **Selection of Items From a Catalog**

  In many organizations, expert systems are used to select goods or services from a larger list of possibilities to meet specific user needs [FONG88], [ALUR90], [POTT90], [RADA90].

- **Monitoring and Control**

  System monitoring, closely related to diagnosis, compares actual system behavior to a model of preferred behavior, operating on a continuous basis [TSUD90].

Examples of other kinds of expert systems that select solutions can be found in [HAYE83] and [CLAN86].

4.1.2 Expert Systems That Construct Solutions

A second method employed by expert systems is to construct solutions from individual components or pieces. This method is used to perform configuration, design, planning, or scheduling tasks. A simple example of a construction type expert system is presented in chapter 8.

Expert systems that construct solutions must first select individual components (or have them provided in the problem description). Components may have complex relationships, many of which recur in generic patterns. Similarly, constraints may be placed on relationships. The knowledge base may contain

generalized rules that specify allowable arrangements of components or express constraints.

The problem-solving strategy of some moderate to large construction type systems requires consideration of many alternative component combinations. For larger configuration problems, the number of potential combinations can exceed the computational resources of the largest computer systems. To solve these problems, many construction expert systems employ heuristics to help identify the most promising combinations, thus reducing the overall amount of search effort that must be expended (described in sec. 3.4). Types of construction tasks are listed below.

- **Design**

  Expert systems have been developed to perform a number of design tasks including: designing maps [ROBI85]; configuring hardware components of computer systems to satisfy customer requirements [MCDE82], [MCDE84]; physical database design [DABR88]; and other areas [COYN90], [LIEB90].

- **Planning and Scheduling**

  The goal of expert systems that plan and schedule is to find an acceptable arrangement of events so as to minimize use of time and resources. A typical example is finding optimal travel routes for delivery trucks [ROTH90].

Most, though not all, construction-oriented tasks require extensive processing. Small scale configuration tasks are largely procedural and do not require extensive computing to perform.

4.1.3 Selection vs. Construction in the Microcomputer Environment

The two strategies described above can be combined to some extent. For instance, design systems may employ heuristic classification to first select component parts. Recommending goods from a catalog may require configuring or assembling related items to suit a particular user's needs.

As a generalization, construction systems that must explore large numbers of alternatives require more computer processing and possibly more memory than do most systems that employ selection. Construction system programs also tend to be larger and more complex, often requiring more powerful microcomputer systems and additional memory capacity. Partially for this reason and partially because construction systems are generally less common, most microcomputer-based expert systems developed with ESBTs are selection systems.

4.2 Deploying Expert Systems Within the Organization

Determining how and where to use expert systems are based on analyzing the information processing needs of an organization. Identification of places to apply expert system technology is based on finding critical points within an organization where automation of expertise can lead to improvements in operational efficiency.

- **Alleviating "Knowledge Bottlenecks"**

  "Knowledge bottlenecks" occur when existing expertise cannot be brought to bear on regularly occurring problems that require expertise to solve. That is, "knowledge bottlenecks" happen when the number of experts is too small for the number of problems that need to be solved. Or experts may be geographically distant from the site of the problem. The example in chapter 3 illustrates such a "bottleneck."

- **Providing a Means for Consistent Decision Making**

  By deploying expert systems throughout an organization, expertise about narrowly focused problems can be disseminated. This can be used to ensure consistent implementation of policies and procedures.

- **Automating Repetitive Tasks That Are Difficult for Humans**

  In many operational situations, certain tasks are performed by continuous repetition over long periods. One example is monitoring computer systems for illegal access attempts [TSUD90]. Such tasks require a certain level of expertise, but they are performed poorly by people because they require constant attention. In these situations, expert systems have proved useful.

- **Freeing Experts for More Important Tasks**

  An expert system can perform many relatively mundane tasks normally handled by an expert. This allows the expert to devote time to other work that may also be important to the organization.

4.3 Characteristics of Appropriate Problems

In determining whether or not expert system technology should be used, it is important to examine the characteristics of the problem to be solved. Problems whose solution can be automated using expert systems methods generally have the following characteristics listed and discussed below.

- **Finding Solutions Requires Expertise**

  The level of difficulty involved in solving the problem must be high enough to pose some barrier to individuals who are not experts. Finding solutions to problems depends on the use of expertise accumulated through training and/or practice.

- **Problem Solving May Have Uncertain Aspects**

  Information about the problem may be uncertain, imprecise, and incomplete. A typical problem may have more than one solution, each of which may have a significant amount of uncertainty associated with it.

- **The Problem Cannot Be Solved by Established Computing Methods**

  Analysis of the problem may show that it can be completely solved by mathematical techniques, operations research methods, or other established computing methods. In this case, expert system technology should not be used. Problem-solving activity that is based on well-defined procedures or that involves only simple decision making can often be more easily automated using conventional software methods.

In summary, expert system methods should be considered when the problem is difficult, the problem domain is characterized by uncertainty, and when well-defined procedures that produce solutions do not exist or cannot easily be developed.

4.4 Characteristics of Knowledge Used to Solve the Problem

Closely related to the characteristics of the problem to be solved are the characteristics of the knowledge used to perform the problem-solving task. These characteristics may be summarized as follows.

- **The Knowledge Necessary to Solve the Problem Depends on Heuristics**

  Not all the knowledge that goes into an expert system is heuristic. Often, however, a large part of it is. An exception may be applications such as claims benefits processing where much of the knowledge consists of laws, regulations, and procedures.

- **The Knowledge Can Be Stated or Represented Declaratively**

  It must be possible to clearly state important concepts, rules, and procedures used to solve the problem of interest. It must also be possible to express this information symbolically in data structures.

- **The Problem-solving Process Is Based on Reasoning**

  As discussed earlier, expert systems are specifically designed to represent heuristic knowledge and reason with it in the manner of a human expert. In most cases, heuristics should be representable in rule form.

4.5 The Scope and Size of the Problem

In practice, the size of completed expert systems is often very large, consisting of hundreds or thousands of rules. If the task is too broad, the development effort may take an inordinate amount of time, or even be impossible. Two important guidelines on evaluating the scope and size of the problem are listed below.

- **The Task Must Be Narrowly Focused**

  The problem to be solved must be restricted and specific. An expert system should be dedicated to a limited domain. For example, an expert system to diagnose factory machine failures may be dedicated to problems for a very specific kind of machine. Or it may diagnose malfunctions of only very specific parts.

- **The Task Should Be Decomposable**

  A well-known problem-solving approach is to decompose a problem into components and work on each component separately. Tasks that utilize the "divide and conquer" approach can be implemented as expert systems in pieces, or phases. This greatly eases development of an application.

In general, selection of a problem having sufficiently narrow scope is critical to success in developing an expert system application.

4.6 The Source of Expertise

To develop an expert system, an established source of expertise must exist. Without an existing base of expertise to solve the problem, expert system technology cannot be applied. The necessary expertise is normally possessed by a recognized human expert or is found in a written source, such as a manual. Sometimes expertise can be obtained from both sources.

- **The Core of the Problem-solving Knowledge Must Be Stable**

  While specific aspects of the problem-solving knowledge might change over time, the underlying concept and basic problem-solving method must remain stable. It is also important to state that expert systems are designed to apply only existing expertise to solve problems. They are not intended to create or invent expertise that did not exist previously.

- **There Must Be Substantial Agreement on Solutions Between Experts**

  If several experts exist and disagree on the solutions for the most important or common problems encountered, this may indicate that the problem-solving knowledge is unstable.

- **The Knowledge Provided by the Source of Expertise Must Be Clear**

  The problem-solving method and the underlying knowledge must be understandable to developers.

- **The Domain Expert Must Be Able to Allocate Time**

  Too often, the amount of time required to develop an expert system is underestimated. In most cases, it is necessary to have a domain expert committed to a project for its duration.

The availability of the source of expertise depends greatly upon management decisions about allocation of resources.

4.7 Areas to Avoid

Some categories of problem-solving activity have been identified as being inappropriate for expert system methods. These are listed below.

- **Tasks Based on Common Sense or Real-World Knowledge**

  The amount of knowledge about the real world is so vast as to be virtually impossible to implement in a computer application. Representation of common sense knowledge is a topic of AI research.

- **Tasks Requiring Perceptual Knowledge**

  Problems that require actually seeing or touching information about the problem are beyond the scope of expert systems.

- **Creativity or Inventiveness**

  Expert systems are not creative in the same sense as humans. As a rule, they are not designed to invent new concepts or ideas beyond those that are encoded in its knowledge base.

Each of these categories involves processes that are too poorly understood to be implemented using current AI programming methods.

4.8 Summary and Further Sources

The discussion contained in this section can be summarized as a list of questions to consider when assessing a candidate task for development as an expert system.

- Can the task be clearly defined? Is the task narrow in focus?
- Does the task fall into one of the expert system categories described in section 4.1?
- Is there a benefit to the organization in automating the task? How will it help improve productivity?
- Does a domain expert exist who can perform the task? Can the domain expert clearly state the knowledge necessary to solve the problem? (If not, an expert system may not be possible.)
- Can the task be expressed in a straightforward algorithm that can be encoded using established computing methods? (If so, an expert system may not be necessary.)
- Does the task depend on heuristic knowledge? Does it require reasoning and inference?
- Does the task require skills that are difficult to automate? (If so, an expert system may not be possible.)
- Is the task small enough to automate? Is it decomposable?

Further references to books and articles that deal with this subject include [BECK90], [CUPE88], [HARM88], [LAUF90], [MART88], [MOCK90], [MURD90], and [PRER89].

5. DEVELOPING EXPERT SYSTEMS

This chapter provides a brief description of the expert systems development paradigm and presents an overview of the development stages. The major classes of software used to develop expert systems are introduced, and the role of expert systems development software is discussed. To those unfamiliar with expert systems development, the chapter provides background needed for detailed understanding of ESBTs, particularly in regard to the features of the development interface.

5.1 The Expert Systems Development Paradigm

The process of building an expert system is called knowledge engineering. Correspondingly, developers of expert systems are referred to as knowledge engineers.

Knowledge engineers develop expert systems by entering knowledge in the form of rules or other representation structures into the knowledge base. Knowledge engineers perform three essential functions:

(1) They interview domain experts to acquire knowledge about solving a particular problem (or in some cases, obtain information from text sources). This process is known as knowledge acquisition.

(2) Knowledge engineers determine system requirements and formulate an overall scheme or model of the problem-solving method employed by the expert. This scheme provides the basis for the design of the expert system.

(3) The third major task of the knowledge engineer is actual system development. Data structures must be chosen to represent knowledge, and inference procedures must be identified that mirror the problem-solving process employed by the expert. This provides a basis to encode knowledge into the selected data structures and develop the expert system software.

For the purpose of this discussion, two related aspects of knowledge engineering will be described: iterative development and prototyping.

Expert systems are developed iteratively in a series of repeated steps. These steps roughly consist of knowledge acquisition, followed by system design (or modification of an existing design), system development (including knowledge entry), and system testing and evaluation. As figure 5.1 illustrates, the relationship between these steps is cyclic.

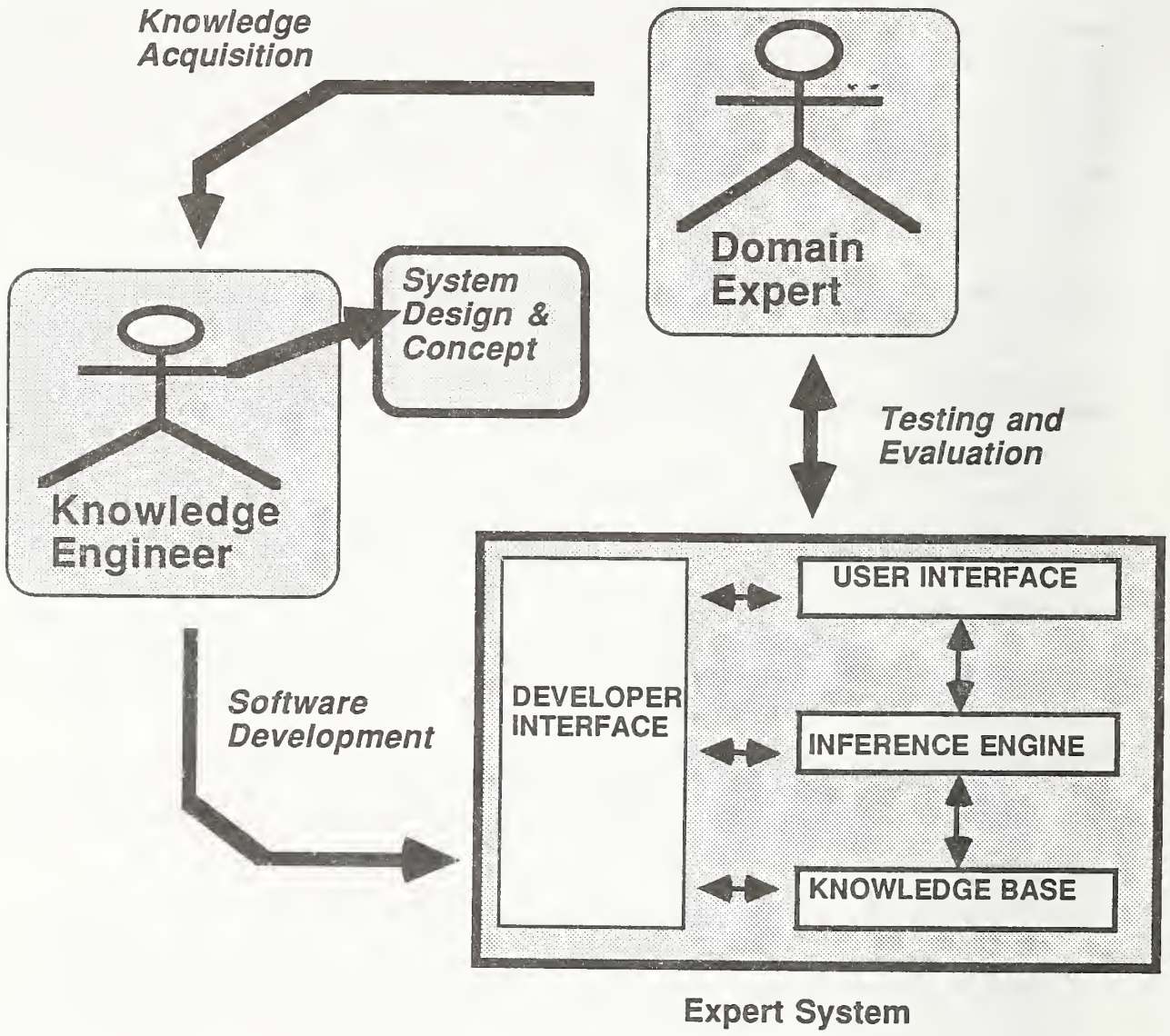

Figure 5.1. Iterative Development of an Expert System.

The process of testing and evaluation should involve both the domain expert and end users. Domain experts test the performance of the developing expert system, while end users test both performance and the effectiveness of the end user interface. In practice, user interface design is often a critical, time-consuming task that must be carefully done to ensure system delivery and acceptance.

Prototyping refers to the creation of limited or "scaled down" versions of the final system, designed to display essential system characteristics or performance. During the initial stages of the development effort, prototypes can be used to help understand system requirements and evaluate preliminary designs. Prototypes are also useful for providing demonstrations to managers and others interested in the progress of the development effort. In the early part of a project, it may be necessary to "sell" the project to managers. Therefore, support for prototyping facilities is an important aspect of ESBTs, particularly in regard to knowledge entry and construction of end user interfaces.

Knowledge engineers use iterative development to create a series of prototypes, if necessary. The system design obtained by prototyping provides the basis for full-scale development of the expert system. Like prototyping, large-scale development is also iterative and proceeds incrementally. This process continues until the expert system grows to maturity.

The description of expert systems development phases can be elaborated by dividing the process into distinct stages.

5.2 Stages of Development

As yet, there are no formal methodologies defined for expert systems development. A possible description of the stages of development in an expert system project is presented below.

(1) **Initial Problem Analysis.** Initially, a general problem area appropriate for expert systems must be selected. This involves assessing the needs of the organization, determining what tasks can be automated, and determining if expert system technology should be used. In some cases, alternative software methods may be preferable. If an expert system is to be developed, a domain expert must be selected. Knowledge engineer(s) and other participants must be identified and trained, if necessary.

(2) **Preliminary Knowledge Engineering.** In this phase, knowledge engineers meet with potential domain expert(s) and end users to acquire necessary background and understanding of the problem domain. Using the guidelines discussed in chapter 4, a specific task is identified as a target for development as an expert system. The task is analyzed and precisely defined.

(3) **Prototype Development and Requirements Analysis.** Detailed knowledge acquisition begins. Meetings with domain experts and end users take place. As described above, a series of prototypes are developed to obtain enough information about the problem to define requirements and develop an overall system design.

(4) **Large-Scale Development.** Large scale knowledge acquisition and knowledge entry begin. As described above, the process proceeds iteratively until the system performs in a satisfactory fashion. End user involvement becomes essential. During this phase, further work on the end user interface takes place with heavy input from end users. Screen layouts, questions, and system responses are reviewed to insure that they are understandable to end users.

(5) **System Delivery and Acceptance.** The system is deployed and organization personnel begin to use it. System documentation should be completed and the end product packaged. Packaging includes user training and user acceptance testing. Even in this stage, additional refinements may be necessary, particularly in the end user interface.

(6) **System Maintenance.** The expert system is permanently deployed and integrated into operation of the organization. Long-term maintenance and update of the knowledge base take place as needed.

Picking the right tool for building the expert system is an important step. Sometimes, more than one tool will be used during the course of a project. Typically, one tool is used for prototyping, but a different tool is chosen for large-scale development and delivery. This topic is addressed in chapter 12.

For general information on expert systems development methodology and life cycle, readers should consult [CUPE88], [HARM88], [MART88], [WATE83].

5.3 The Domain Expert as System Developer

For smaller expert systems, it is sometimes desirable for a domain expert to develop the knowledge base directly without a knowledge engineer as an intermediary. This makes the domain expert the system developer. In many cases, domain experts are willing to do this for several reasons.

- Familiarizing a knowledge engineer with certain domains such as chemistry or physics, can be very difficult. Often, knowledge engineers, who are trained as computer professionals, do not have the necessary background to comprehend key concepts associated with a domain. This may force the knowledge engineer to undergo a learning curve and make knowledge acquisition time-consuming and difficult. Under these circumstances, it may be easier to develop a correct knowledge base if domain experts can directly encode the knowledge base by themselves.

- Acquiring knowledge from an expert often consists of a prolonged series of intense interviews, sometimes extending over a period of many months. This process can become a bottleneck in expert systems development. Allowing the domain expert to do software development directly can save time.

- Many domain experts may simply wish to do new and interesting things, such as develop expert systems.

If the domain expert is to develop the expert system, choosing the right ESBT is essential. The ESBT must be easy to learn and to use, or the domain expert may not be able to do the job. In many cases, it is advisable to have a knowledge engineer or computer programmer develop other parts of the expert system program that require low level coding, such as the end user interface or any external interfaces that may be necessary. Large, complex expert system projects cannot normally be developed in this manner. Larger applications often require the efforts of one or more knowledge engineers.

5.4 General Categories of Software for Expert Systems Development

Several major categories of software for developing expert systems exist.

- **Conventional Programming Languages**

  This includes traditional languages such as Pascal, C, and Modula-2. While these languages were not designed for development of expert systems, it is possible to use them for this purpose [BUTL88], [HU89], [SAWY86], and [SCHI87].

- **Programming Languages With Development Environments**

  The term "environments," when applied to programming languages, refers to an integrated set of interactive software tools used to support programming activities. In addition to editors and compilers, environments can include facilities for file management, browsing program code, debugging and tracing program execution, and graphics programming.

- **Artificial Intelligence Programming Languages**

  This includes languages used for artificial intelligence applications, such as LISP [BETZ85], [CHAR87], [WILE84], [WINS89], and Smalltalk [GOLD83], [PARC89]. These languages have sophisticated capabilities for creating and manipulating information represented in symbols and structured types. Often, they also provide sophisticated programming environments.

- **Prolog and OPS5**

  Prolog and OPS5 are general-purpose AI programming systems that provide a means to represent knowledge and perform inference. Prolog is based on the concepts of logic programming [COLM82], [STER86]. OPS5 was developed for implementing production rule systems using the powerful Rete algorithm for inference [FORG77], [FORG82]. Both languages support development of a wide variety of artificial intelligence and knowledge-based systems, including expert systems [BRAT86], [BROW86], [MARC88], [MERR89]. Many ESBT products have incorporated features offered by these languages, as is discussed in chapter 9.

- **Expert System Building Tools**

  Like Prolog and OPS5, ESBTs provide a means to represent knowledge and carry out inference. Like AI languages, ESBTs provide a development environment. However, unlike Prolog, OPS5, and AI languages, ESBTs are more focused on supporting expert systems development and include features specifically for this purpose.

While each of these categories of software can be used to develop expert systems, there are important differences between each.

5.5 Differences Between ESBTs and Programming Languages

ESBTs specify how knowledge can be represented, what forms of inference are possible, and what types of interfaces can be created. Thus, to a great extent, ESBTs specify the structure of the applications. By doing so, an ESBT can restrict the kinds of applications that can be created. When selecting an ESBT to use, application characteristics must be matched to the capabilities offered by the ESBT (discussed in ch. 12). If there is a good match, low-level programming is minimized, simplifying and accelerating application development. This is particularly important for creating expert systems prototypes, since prototyping requires the ability to construct software quickly.

In contrast, when developing an expert system using an AI language, both the knowledge representation system and the inference engine must be programmed from scratch using low-level language primitives. Although this means more programming, most programming languages, especially AI languages, are more general than ESBTs and provide more flexibility in creating knowledge representation structures and inference procedures. This added generality and flexibility permits development of a wider range of applications. However, using a language sometimes requires a considerable amount of programming skill and effort.

Prolog and OPS5 can be thought of as being "in between" AI languages and ESBTs. Because they provide specific knowledge representation structures and inference procedures, Prolog and OPS5 impose more structure on application development than do languages. However, Prolog and OPS5 impose less structure than most ESBTs. Often, their powerful capabilities can be used to develop expert systems whose requirements cannot be met by ESBTs. Prolog and OPS5 are more general than ESBTs and can be used for other AI applications besides expert systems. They are also valuable as programming tools for research purposes. Figure 5.2 summarizes some of the differences between AI languages and ESBTs.

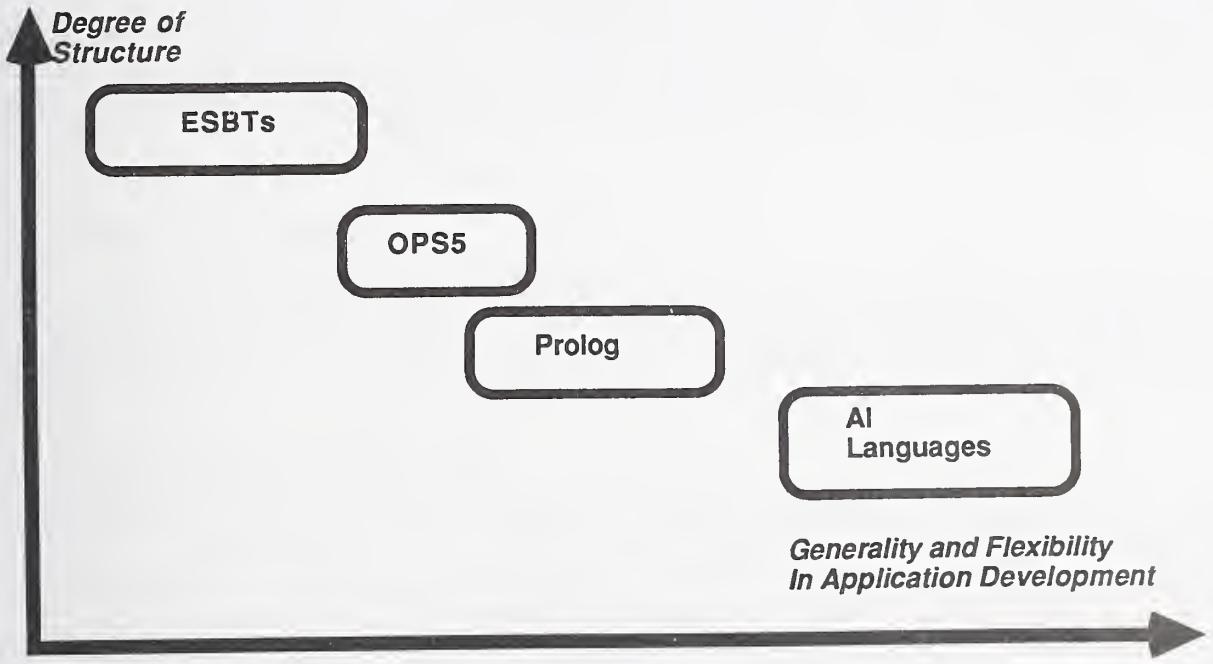

Figure 5.2. Classes of Software Tools.

The contrast between structure and flexibility presented above is a generalization useful in understanding ESBTs as well. As discussed in subsequent chapters, more sophisticated ESBTs provide the structure of rule-based systems as well as the generality and flexibility of programming languages.

6. ANALYZING MICROCOMPUTER-BASED EXPERT SYSTEMS BUILDING TOOLS

The purpose of this chapter is to define the ESBT architecture and provide a framework for detailed analysis of microcomputer tools.

This chapter begins by describing commonly accepted categories of ESBTs. These categories and the differences between them are discussed in this chapter, and microcomputer ESBTs are placed within this context. Relevant trends that are causing changes to these categorizations and affecting the capabilities of microcomputer-based tools are described. The architecture common to microcomputer-based ESBTs is then presented in detail, and its components are explained.

The discussion of the ESBT architecture and tool categories provides a background and framework for the detailed analysis of microcomputer-based tools. The last section of this chapter describes the criteria used in this analysis and sets the stage for the remainder of the report.

6.1 General Categories of ESBTs

ESBTs are sometimes grouped into general categories based on cost, the kinds of platforms they run on, and their level of sophistication. A representative categorization is described below.

- **Small Rule-Based ESBTs**

  This class of tools supports construction of expert systems of relatively modest size that consist primarily of rules. To date, most of the ESBTs available for the microcomputer environment fall into this category. These tools are, for the most part, relatively inexpensive, costing less than $1000. Publications describing examples of such tools include [BROD89], [COFF90], [GEVA87], [HARM88], [MART88], and [VEDD89].

- **Medium-Sized ESBTs**

  ESBTs in this category are generally intended to operate on medium-sized workstation platforms and mainframe computers. Besides rules, they support additional features for knowledge representation and reasoning (described in ch. 9) and can be generally used to implement larger expert systems. Relevant references that include examples of such tools include [GEVA87], [HARM88], and [MART88].

- **Large-Scale Hybrid Systems**

  In the past, this has been regarded as the most powerful category, providing the widest range of capabilities. Hybrid systems combine production rules, sophisticated knowledge representation capabilities, a complete programming language, and graphics. Hybrid systems were originally developed for powerful workstation environments such as those provided by LISP machines [KUNZ84], [MART88].

In addition, ESBTs that use a technique known as induction are sometimes recognized as another class. Induction is a procedure by which a decision tree (or set of rules) is generated from a series of examples of problem solutions. Each example consists of a complete description of a problem occurrence and its solution (see ch. 8). Currently, relatively few ESBTs rely solely on induction [BIEL88], [BROD89]. In some cases, existing ESBTs have been extended to incorporate inductive subsystems.

6.2 Contrasting and Comparing Categories of ESBTs

Section 5.4 contrasted general classes of expert systems development software. These classes were distinguished by (1) the degree of structure the software tool imposed in limiting the kinds of applications that could be developed and (2) by support for generality and flexibility. The major categories of ESBTs can also be distinguished using these criteria.

An ESBT permits generality and flexibility by supporting a variety of knowledge representation features, the ability to implement complex forms of inference, and procedural programming. To permit as much generality as possible, an ESBT may provide a complete programming language (in addition to inference) with rich data representation capabilities. Often, the language is object-oriented.[3] Using a programming language, the developer can implement representation structures and procedures that are not possible in purely production rule tools. Traditionally, only ESBTs running on more powerful computer platforms have combined the capabilities offered by programming languages with production rule processing. Figure 6.1 summarizes these differences among classes of ESBTs.

[3]Object-oriented programming is becoming more important in ESBTs and is described in more detail in chapter 9.

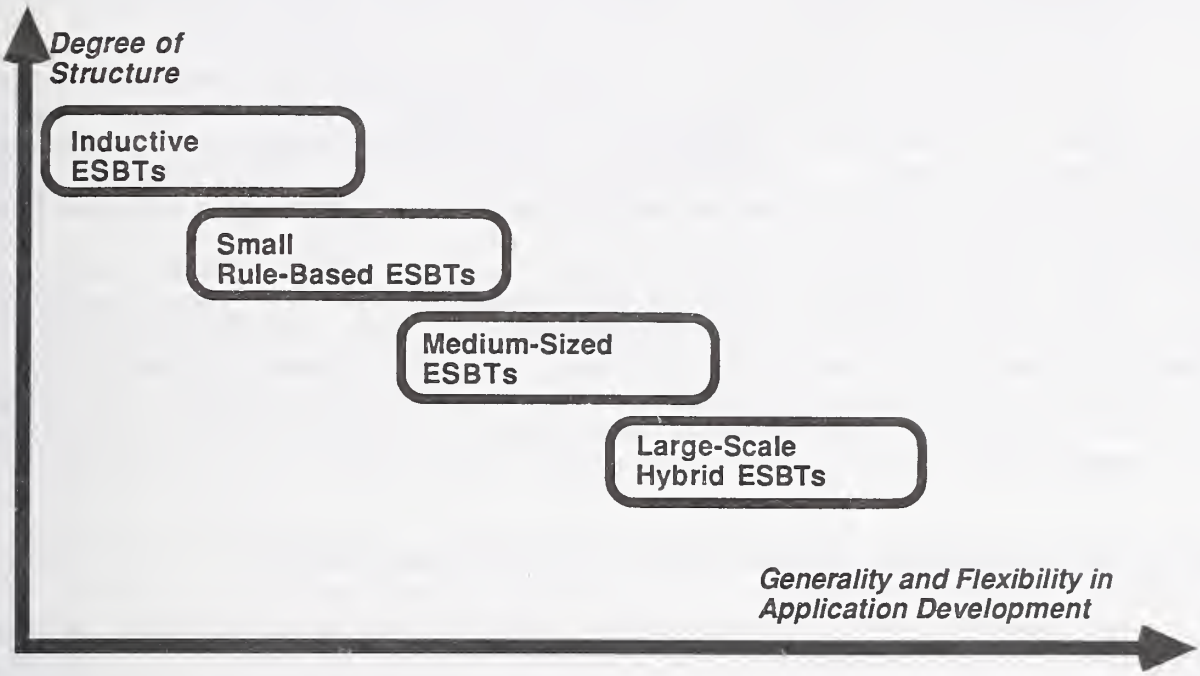

Figure 6.1. Comparison of ESBT Classes.

The categorization of ESBTs presented in this chapter is affected by two factors. The first factor is the progressive development of smaller, more powerful varieties of relatively low-cost computers and computer workstations. The second factor is the advancements in software technology that have led to more efficient implementations of ESBT software based on programming languages such as C.

6.3 The "Downscaling" of ESBT Capabilities

These advances in hardware and software technology are making it possible to add new features to ESBTs that operate on smaller hardware platforms. Features once available only for large, expensive computers are now being offered by ESBTs designed for smaller environments.

Features such as object-oriented programming, advanced forms of knowledge representation, and graphics were once provided only

by ESBTs that ran on LISP machines and other specialized computers. These features are now becoming available in ESBTs running on smaller, less-expensive platforms. Some vendors are adding features to small rule-based ESBTs that were once supported by medium-sized and large-scale tools. This is rapidly altering traditional tool category distinctions and "blurring" the lines between them.

The evolution of ESBTs, in general, has been greatly affected by this "downscaling" of capabilities from larger, more powerful platforms to smaller platforms. This has also been true of microcomputer ESBTs. Microcomputer-based tools have emerged that support many of the capabilities once found only in medium-sized and large hybrid ESBTs.

The "downscaling" of ESBT capabilities has also made it difficult to think of microcomputer tools as a category with a fixed set of tool features and capabilities. Instead, it is more appropriate to discuss state-of-the-art ESBT technology in terms of what features are becoming available as well as what features are currently available.

6.4 The ESBT Architecture and Its Major Features

The major use of the ESBT is to develop expert system programs. The architecture of the ESBT is designed to fulfill this function. The architecture presented below is shared by most, though not all, microcomputer-based ESBTs. This architecture consists of the several components.

- **The Development Environment**

  The ESBT provides a customized development environment for knowledge engineering. The developing expert system program is stored in one or more files that can be accessed and edited through the development environment. This environment includes:

  (1) Facilities for creating and updating the knowledge base and other parts of the program using file editors and other knowledge entry tools.

  (2) Facilities for testing and debugging including a variety of software "aids," such as explanation systems and possibly graphics utilities for representing and browsing the structure of the knowledge base.

  The development environment is analogous to the development environments of many sophisticated programming languages available for computer workstations. ESBT development environments are discussed in detail in chapter 7.

- **The Inference Engine**

  The ESBT provides an inference engine that can be used by the developing expert system. When application development is completed, the inference engine can be incorporated as a software component in the finished expert system program. ESBT features for supporting inference capabilities are discussed in chapters 8 and 9.

- **The Knowledge Representation System**

  The knowledge representation system is used to represent rules as well as other forms of knowledge. Typically, an ESBT provides a high-level, English-like language for encoding knowledge. The language can be used to declare rules and other representation structures that are stored in the knowledge base of the developing expert system. ESBT support for knowledge representation is discussed in chapter 9.

- **Software Facilities for Constructing End User Interfaces**

  This includes facilities for defining what the end user sees at the terminal while using the expert system, including display of questions generated by the expert system, display of solutions, and display of explanations. Typically, the ESBT provides a high-level programming language for developing end user interface software. ESBT support for development of end user interfaces is discussed in chapter 10.

- **Software Facilities for Creating External Interfaces**

  This includes facilities for defining interfaces between expert systems and other software systems, such as DBMSs. Typically, these facilities are programmed with a high-level language provided for this purpose. ESBT support for external interfaces is discussed in chapter 11.

- **Facilities for Application Delivery**

  To support delivery of completed expert system applications, most ESBTs provide facilities to create delivery versions of the completed expert system. The delivery version consists of those parts of the ESBT and expert system application that are needed for actual operation of the finished system. Delivery versions usually include compiled code for the knowledge base, inference engine, end user interface and other external interfaces. Delivery versions of expert systems often exclude the developer interface. This means that the end user cannot modify the expert system program. This is often desirable to prevent end users from making unwanted changes to the knowledge base.

The knowledge engineer uses the development environment to develop an expert system. This environment provides access to the knowledge representation system, inference engine, end user interface construction facilities, and external interface facilities. The knowledge engineer uses these components as software building blocks for constructing an expert system program. Use of these building blocks can reduce the amount of programming and enhance the ease and speed of development.

When development of the expert system is finished, the delivery facilities are used to create copies of the expert system program for distribution. Figure 6.2 below illustrates the process.

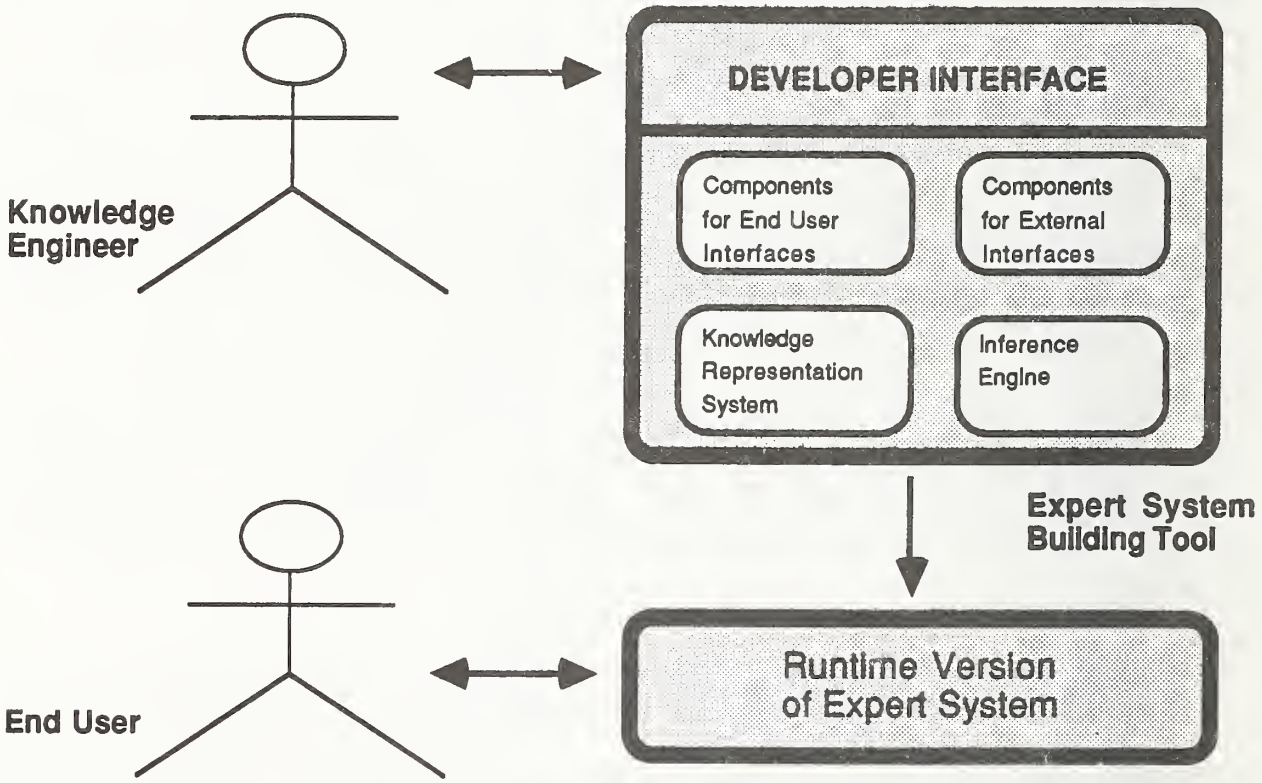

Figure 6.2. An ESBT Architecture.

A single ESBT may be used and reused to create many expert system applications. The ESBT can maintain files for many different expert system programs at the same time.

6.5 Criteria for Analyzing ESBT Features

The criteria for analyzing ESBT features must be discussed before proceeding with ESBT analysis. In chapter 1 (sec. 1.2), a list of questions was provided that could be used to analyze features and capabilities of microcomputer-based ESBTs. Below, this list is modified and extended to take into consideration additional factors introduced in the preceding chapters. These factors include: the "downscaling" trend, support for expert system functions, the type of expert system to be developed, and development activities such as prototyping.

- What features exist to implement characteristic aspects of expert systems such as explanation and end user interfaces?

- What capabilities does each feature support? Can it be found in some, most, or all products? To what extent is it a subject of ongoing research? Has the feature been recently introduced? Is the feature considered "state-of-the-art?"

- Which ESBT supported capabilities are necessary to implement the different classes of expert system applications (discussed in ch. 4)? Is a particular capability appropriate for selection expert systems? For construction expert systems?

- Does a feature support the capability to develop large applications? Or is it limited to smaller applications that are customary for microcomputer-based ESBTs?

- How do features of the development environment support specific aspects of expert systems development? What features support application prototyping? What features are important in other application development activities, such as debugging?

- To what extent are well-developed computer skills required to use a feature? Can the feature be used by a non-programmer?

- Was the feature "downscaled" from larger platforms? If so, how well does the feature work in a microcomputer environment? To what extent are microcomputer resources stretched; e.g., memory, processing power, etc.?

The actual criteria used may differ from chapter to chapter, as appropriate for the subject matter.

7. FEATURES OF THE DEVELOPER INTERFACE

The development environment provides a means for the knowledge engineer to develop individual expert system applications. The development environment consists of an integrated set of features to aid the knowledge engineer in creating the knowledge base, end user interface, and external interfaces. Development environments may also provide a means for managing different expert system programs under development.

Features of the development environment include facilities for supporting knowledge entry, knowledge base analysis, and other program development support facilities. Knowledge entry refers to the ability to access, augment, and change the knowledge base of an individual expert system under development. Knowledge base analysis facilities provide a means to analyze, test, and debug knowledge bases and other parts of the expert system program. The development environment provides a menu-based file system interface to permit display and access of individual expert system programs.

As a whole, the development environment can be viewed as an integrated software package whose individual features are accessible via a menu interface. In the first two sections of this chapter, the individual features of the development environment are discussed. Section 7.3 presents an example of how the development environment is used. Section 7.4 describes facilities for learning how to use ESBTs. Section 7.5 focuses on the important subject of application prototyping. In section 7.6, ESBT support for long-term maintenance of knowledge bases is discussed.

7.1 Knowledge Entry Facilities

Like conventional computer programs, expert system programs are stored in files maintained on permanent storage media. In the course of developing an expert system program, individual files must be accessed, edited, and updated. ESBT can provide several different ways to do this, three of which are listed below.

- **Word Processing Type Editors**

  Some ESBTs supply a text editor that allows production rules and other knowledge structures to be entered and edited in a buffer. This method of knowledge entry is the same as provided by a programming language editor or a text word processor.

  An ESBT may either provide its own text editor or permit use of an outside commercial word processing package. An important capability is use of multiple edit buffers. Separate buffers can contain different parts of a knowledge base, allowing the developer to work on larger applications.

- **Structured Knowledge Entry**

  This method permits production rules and other structures to be entered systematically through use of menus and prompts. Groups of rules can be displayed and accessed through a system of hierarchical menus. Parts of individual rules may be entered in separate operations. Individual rule conditions and rule conclusions can be entered in response to prompts or selected from menus.

  A structured entry system can also support type checking and enforcement of structural constraints. The effectiveness of structured entry systems is dependent on a well-designed sequence of menus and commands.

- **Graphics Interfaces**

  More advanced ESBTs use graphics facilities to display the structure of the knowledge base as a network of related rules. Within a group of interconnected rules, the complete text of each rule may be shown, or rules may be represented by graphics icons that can be expanded. An individual rule can be selected for editing by the developer. Once selected, the rule can appear in a text edit buffer or structured entry interface. Larger collections of rules cannot always be placed entirely on a single screen; therefore, pan and zoom capabilities are sometimes provided.

  Graphics interfaces presuppose the use of specialized graphics software that can support the construction of sophisticated window interfaces. Until recently, sufficiently powerful graphics software packages were available mostly for larger computer platforms.

Text editors are available in nearly all microcomputer-based ESBTs. Structured knowledge entry is becoming more commonly supported. With the improvements being made to graphics software available for microcomputers, graphics interfaces are also beginning to appear at the microcomputer level.

Small rule-based systems can be easily developed using only text editors. However, for larger, more complex expert systems, graphics interfaces provide advantages in managing substantial numbers of rules and knowledge structures.

Structured interfaces generally do not require extensive programming skills. A well-designed structured interface can often be used by a non-computer specialist to implement rule-based expert systems of modest size and complexity.

7.2 Features for Knowledge Base Analysis

Knowledge bases with even a modest number of rules can be quite complex and difficult to understand. This section describes facilities that the ESBT development environment may provide for analyzing and understanding the contents of the knowledge base.

Three categories of facilities are identified: (1) facilities to help analyze the syntactic correctness of the knowledge base, (2) facilities to analyze the structure of the knowledge base, and (3) facilities to analyze the execution of the developing expert system to determine correct performance. Individual ESBTs may contain some or all of these features.

7.2.1 Facilities for Syntactic Analysis During Program Compilation

Some ESBTs compile production rules directly into low-level executable images. ESBTs may also translate knowledge bases into the source code of a programming language, such as C or Pascal, and then compile it. Other components of the expert system program, including those that specify end user and external interfaces, must also be translated and compiled. During translation or compilation, syntactic errors should be diagnosed and displayed to the developer. Completeness and accuracy in providing compilation diagnostics is an important factor in being able to develop applications easily.

Many smaller microcomputer ESBTs require the compilation of an entire file, as takes place in the traditional program development. When developing larger expert systems, it is sometimes an advantage to be able to recompile small segments of the expert system program file individually. This process, known as incremental compilation, is useful when only a single rule in a large file must be changed. The rule can be located in the buffer, modified as appropriate, and recompiled separately without affecting the rest of the file.

An ESBT may provide a programming language interpreter (in addition to a regular compiler) that can be used in the manner of an incremental compiler.[4] Generally, incremental rule compilation or language interpretation facilities are available in more sophisticated microcomputer-based ESBTs.

---

[4]Interpreted programs generally execute more slowly than compiled programs. Usually compiled code is used for production of runtime copies of large expert system applications.

7.2.2 Facilities for Analyzing Structure of Knowledge Bases

During knowledge engineering, the contents of the knowledge base and its interrelationships need to be examined and analyzed. Some features designed for this purpose are listed below.

- **Graphics Rule Interfaces for Display of Rule Networks**

  As discussed in section 7.1, the structure of a knowledge base may be illustrated using graphics to display a rule network. In a network, links between rules can be represented using symbols such as arrows. These links can be followed using a mouse (mouse facilities will be discussed in ch. 10), thus allowing the developer to browse the contents of the knowledge base and examine interrelationships between its components.

- **Facilities for Querying the Knowledge Base**

  Querying a knowledge base is a desirable capability. For instance, it may be necessary to retrieve all rules that conclude a particular fact. Similarly, it may be desirable to retrieve rules that have a certain condition. Despite their utility, conditional query capabilities are not widely supported by microcomputer-based ESBTs. A limited query capability can be attained by using the text editor to search for text strings.

- **Facilities for Verifying Consistency and Completeness of Knowledge Bases**

  Verification of knowledge base consistency involves detecting problems such as contradictory rules, circular rules, redundant rules, and subsumed rules. Contradictory rules are rules that have identical conditions but contradicting conclusions. Circular rules are rules whose conditions and conclusions form a cycle. Elimination of redundant rules and rules that subsume other rules makes a knowledge base clearer and more maintainable. Verifying completeness means ensuring that there are no gaps in the knowledge base. That is, the knowledge base is complete if its knowledge covers all reasonable combinations of conditions that may occur in an actual problem. In a rule-based system, the knowledge engineer can then add rules to fill gaps. Verifying knowledge bases manually can be tedious and time-consuming. Automatic procedures for knowledge base verification are, for the most part, in the research stage [NGUY87], [STAC87]. However, some commercial products are beginning to appear that support verification capabilities.

It is possible that knowledge base query facilities and automatic verification of knowledge bases may attain greater support among microcomputer-based ESBTs in the future.

7.2.3 Facilities for Analyzing Execution of Expert Systems

In microcomputer-based ESBTs, facilities for analyzing execution of expert systems are used to debug and test developing applications. Some of these facilities are listed below.

- **Facilities to Trace the Execution of Expert Systems**

  The ability to display the sequence of rule invocations and actions during the execution of an expert system is valuable in debugging an expert system. Most ESBTs provide a trace facility to display the rules being tested and the facts being concluded. Well-designed trace facilities permit the developer to limit the amount of information displayed, to control the speed of execution, and to interrupt execution when desired. It is also useful to be able to write the contents of a trace to a file for later analysis.

- **On-Line Examination of Facts and Explanation of Conclusions**

  Many ESBTs permit an executing expert system to be halted to examine the contents of its context file and see what facts have been concluded. The explanation facility may then be used to generate explanations of particular conclusions. A context file can also be examined after the expert system has finished executing. This capability is especially helpful for testing a developing expert system.

- **Facilities for Breaking Execution**

  Some ESBTs permit the developer to specify that an executing program should halt under certain conditions--such as when a particular rule is being tested or a variable is set to a particular value. Once the predetermined "break" in execution occurs, the developer can examine the context file and other system variables. During the "break," the developer may also change values assigned to variables, add facts to the context file, or remove facts from the context file. The developer may then resume execution, ending the "break."

- **Generation of "What if" Queries**

  "What if" queries are used to examine how an expert system's answer changes when the set of problem facts is altered. To execute a "what if" query, the developer must first access the context file of an expert system that has finished executing. The developer can then change selected variable values and re-run the expert system without repeating other answers. "What if" query features permit rapid evaluation of system performance with many different combinations of facts, a useful capability in ensuring the expert system performs correctly.

Many inexpensive ESBTs support most or all of the execution analysis features presented in this section. If bit-mapped graphics displays of rule networks are supported, the execution of an expert system can be traced by highlighting sections of the rule network as they are tested by the inference engine.

7.3 A Typical Development Session Using an ESBT

The iterative style of expert systems development (discussed in ch. 5) involves gradual revision and augmentation of the application over a period of time. During development, the knowledge engineer will repeatedly need to edit, test, and debug the knowledge base and other parts of the program. A typical session using an ESBT is described below.

(1) The ESBT is accessed. Usually a command is typed at the computer terminal that invokes the ESBT program.

(2) Typically a top level menu of options is displayed. The menu may consist of a list of expert system program modules under development. The menu may provide options to edit, compile, or execute individual modules. The developer may also be permitted to create an entirely new module.

(3) If the developer chooses to edit a particular module, an edit buffer with the corresponding file is created. In a less expensive ESBT, the editor will be similar to an editor available in a commercial word processing package.

(4) During the edit session, individual rules, data structures, and other programming language statements may be viewed, modified, or deleted. Additional rules may be entered.

(5) After the edit session is complete, the file is saved and compiled. If syntax errors occur during compilation, appropriate messages are displayed, usually with references to line numbers. Some ESBTs automatically reenter the edit buffer and display the rule or code in error.

(6) Once the program compiles correctly, the developer may then execute the module to test the operation of the system. When the expert system is operating, execution trace facilities may be employed to display which rules are being tested and to show what information has currently been concluded. In many ESBTs, trace information may be stored in files for later review by the developer.

(7) After the module has finished executing, the top level menu is redisplayed. The developer may choose to edit the module again or to edit other modules. Alternatively, the developer may exit the ESBT.

7.4 Learning to Use an ESBT

An often overlooked aspect of ESBTs is the support they may provide (or fail to provide) for learning to use the product. This is particularly important to users with little or no previous computer experience. Support for learning to use ESBTs may be provided in several ways.

- **Manuals**

  All software products provide manuals that describe how they should be used. Manuals for ESBTs must be as clear and nontechnical as possible. A description of every feature should be provided together with references in an index. Examples of how individual features are used are also helpful.

- **Tutorials**

  A tutorial is an excellent way to learn how to use an ESBT. A thorough tutorial provides a sequence of steps that takes the developer through the process of using an ESBT (similar to the session described in sec. 7.3). A tutorial may assist the developer in constructing a sample expert system. ESBT manuals frequently have tutorial sections for developers to read and follow.

- **Sample Demonstration Programs**

  Most ESBTs provide a few examples of small expert system programs for demonstration purposes. If source code is also provided, the developer may use these small working programs as a basis for creating larger programs.

- **On-Line Help Facilities**

  During a development session, on-line help facilities are essential in explaining the use of various development environment features. Help facilities are often implemented using menus to display a list of features for which help can be provided. When the developer chooses a feature, an explanation of its function may be displayed.

- **Training**

  Many ESBT vendors provide training classes to assist developers in learning to use their product. Training classes can accelerate the learning process and are particularly useful for novice developers.

Not all ESBTs support these features to the same degree. For example, tutorials are sometimes limited or even omitted. Manuals and on-line help facilities may also be incomplete.

7.5 The Development Interface and Application Prototyping

As discussed in chapter 5, application prototyping is an important aspect of expert systems development. Therefore, it is critical for an ESBT to support prototyping. Some useful features for prototyping are listed below.

- **Support for Accessing the Knowledge Base**

  Rapid and convenient access to specific portions of a knowledge base under development allows the knowledge engineers to quickly comprehend the structure of a knowledge base and make modifications. Though not widely available in ESBTs designed for microcomputer environments, graphics interfaces and query facilities (discussed in sec. 7.2.2) can be used to support knowledge base access capabilities.

- **Effective Application Debugging Facilities**

  As with any programming environment, facilities for syntactic analysis of knowledge bases speed program development. Rapid program development is essential to the prototyping process. Facilities for syntactic analysis include a compiler with accurate diagnostics as well as utilities to trace and analyze the execution of an expert system (also in sec. 7.2.2).

- **Incremental Compilation**

  Incremental compilation (and language interpretation) support the expert systems development cycle (discussed in sec. 5.1) by making possible rapid alteration and testing of small portions of a knowledge base. This capability is especially important for development of larger knowledge bases.

- **Software Libraries for Procedural Components of Expert System Programs**

  To some extent, ESBTs can promote reusability of procedural components (such as end user interface software, external interface communication modules, etc.) by providing the ability to create software libraries for programming language modules (in addition to the menu-based file systems discussed at the beginning of this chapter). This allows knowledge engineers to easily incorporate previously developed procedural code into new applications.

- **Configuration Management and Control Facilities**

  Configuration management and control facilities enable different files associated with a particular expert system application to be more easily maintained. In particular,

configuration management facilities provide the ability to maintain consistent versions of knowledge bases which are made up of many files. At the present time, few ESBTs extensively support configuration management.

While the development interface can provide essential features for prototyping, other aspects of the ESBT are equally important for quickly developing expert system programs. Among the most important are the software components that the ESBT provides for creating the expert system program code itself. These components, introduced in chapter 6, are discussed in the following chapters.

7.6 ESBTs and Long-term Maintenance of Knowledge Bases

One of the major benefits attributed to expert system technology is long-term preservation of knowledge. For example, it is potentially desirable to be able to preserve the knowledge of domain experts for use after they retire from an organization. At the writing of this report, there has been less experience with long-term preservation and maintenance of knowledge than with other aspects of expert system development.[5] Hence, this area is not completely understood and remains a research topic.

However, it is known that even knowledge bases of moderate size can have many complex interrelationships between rules. It can therefore be stated with some confidence that, over a period of time, gradual evolution of problem-solving knowledge can make long-term maintenance of larger expert systems a difficult task. To support long-term maintenance, the following features are desirable:

- Configuration management and control,
- Facilities to assess the impact of changes to the knowledge base, such as "what if" facilities that can be used to determine the effect of updates to the knowledge base,
- Automatic verification facilities for analysis of changes to the knowledge base,
- Conditional query facilities similar to those supported by DBMSs that permit retrieval of subsets of the knowledge base, such as all rules that conclude a particular fact, and
- Advanced explanation generation facilities for extraction and display of parts of the knowledge base are useful in showing what solutions can be reached. For example, advanced

---

[5]One case study of the evolution of a knowledge base over time is [MCDE84].

explanation facilities may include the capability to display and browse rules that represent lines of reasoning used to reach particular solutions. At present, however, few of these facilities are provided by ESBTs.

7.7 Summary of Development Interface Features

In the following table, the major features of the development environment are summarized. Each feature or capability is characterized by:

- The level of computer skill required to use it effectively.
- Its frequency of occurrence in microcomputer-based ESBTs.
- Computer resource requirements.
- Whether or not the feature is especially useful for development of large applications.

Two of these categories require some additional explanation. The level of required computer skill is divided into three categories. The term "Computer Literate" refers to individuals who regularly use microcomputers but are not professional programmers. This includes users of word processing packages, computer spreadsheet programs, and other software analysis packages intended for non-programmers. The term "Computer Specialist" refers to someone who is proficient in designing and developing computer programs using a conventional programming language--such as COBOL, FORTRAN, C, Pascal, etc. A "Computer Specialist" may be a professional computer programmer or a systems analyst. The term "AI Specialist" refers to a computer professional whose area of specialization is expert systems or artificial intelligence.

Resource requirements are rated "Low" if the feature can be effectively used on a standard PC with 640K memory, "Medium" if 1-4 megabytes of memory are required, and "High" if a 386 processor and 4 megabytes or more of memory are required.

Table 7.1. Summary of Development Interface Features

Feature	Required Skill Level	Occurrence Of Feature	Computer Resource Require-ments	Supports Large Application Development?
Menu-Based File System	Computer Literate	Common	Low	No
Basic Word Processing Editor	Computer Literate	Common	Low	No
Structured Knowledge Entry Facility	Computer Literate	Becoming Common	Medium	No
Facilities For Type & Constraint Enforcement	Computer Specialist	Becoming Common	Medium	No
Graphics Interface	Computer Specialist	Less Common	High	Possibly
Knowledge Base Query Facilities	Computer Literate	Rare	High	Possibly
Knowledge Base Compilation Diagnostics	Computer Specialist	Common	Low	No
Incremental Compilation	Computer Specialist	Less Common	High	Yes
Verification	AI Specialist	Rare	Unknown	Possibly
Basic Execution Tracing Facilities	Computer Specialist	Common	Low	No
Control Of Speed & Content Of Tracing	Computer Specialist	Less Common	Low	No
On-Line Examination Of Facts & Conclusions	Computer Specialist	Common	Low	No
Configuration Management Facilities	Computer Literate	Rare	Unknown	Yes

8. SUPPORT FOR BASIC INFERENCE CAPABILITIES

In an expert system, inference is carried out by the inference engine, introduced in chapter 2. An inference engine is a software module that encodes specific procedures for carrying out inference in a systematic way, most often using production rules. Unlike the knowledge base whose contents can be easily changed, an inference engine is a more or less fixed program that cannot be modified substantially. It is possible that the same inference engine can be used in many expert systems, each of which has a different knowledge base. For this reason, one of the most important advantages of an ESBT is that it provides an inference engine program module for direct incorporation into a developing expert system.

This chapter focuses on features of inference engines commonly provided by microcomputer-based ESBTs and discusses the basic inference capabilities that are supported. ESBT support for major inference strategies, including both forward and backward chaining, are discussed in some detail. Other inference strategies are mentioned briefly. Support for reasoning about uncertainty, an important capability in some expert systems, is introduced and discussed. Similarly, inductive systems are also presented. The discussion of pattern matching, a feature that supports more powerful inference capabilities and involves complex forms of knowledge representation, is deferred to chapter 9.

8.1 Backward Chaining

Chapter 2 introduced the backward chaining strategy and provided an example. In practice, many expert systems that select solutions, including those that employ heuristic classification, can be implemented using backward chaining. (Selection type expert systems and heuristic classification were described in ch. 4.) This is because the backward chaining process permits consideration of alternative problem solutions and examination of supporting evidence in a systematic manner. For this reason, a number of smaller microcomputer-based ESBTs provide inference engines in which backward chaining is the predominant inference strategy. These tools can be used to implement small to moderate-sized expert systems that select solutions.

8.1.1 The Backward Chaining Process

The backward chaining process can be described as decomposing a problem into goals and subgoals that must be systematically fulfilled. A subgoal is created whenever an inference engine must determine a fact or other piece of information. The needed fact can be determined by: checking the context file (explained in sec. 2.3), using a production rule to make an inference, or by asking

the end user. When a rule is used, its conditions must be satisfied--therefore, finding facts that satisfy the rule's conditions become new subgoals that must be fulfilled first. This process may repeat many times, working "backwards" from a rule conclusion that matches a subgoal to the rule's conditions that become new subgoals.

Figure 8.1 illustrates the decomposition of the factory floor example. In this figure, rules and subgoals appear as nodes connected by arrows. The arrows show the direction of chaining.

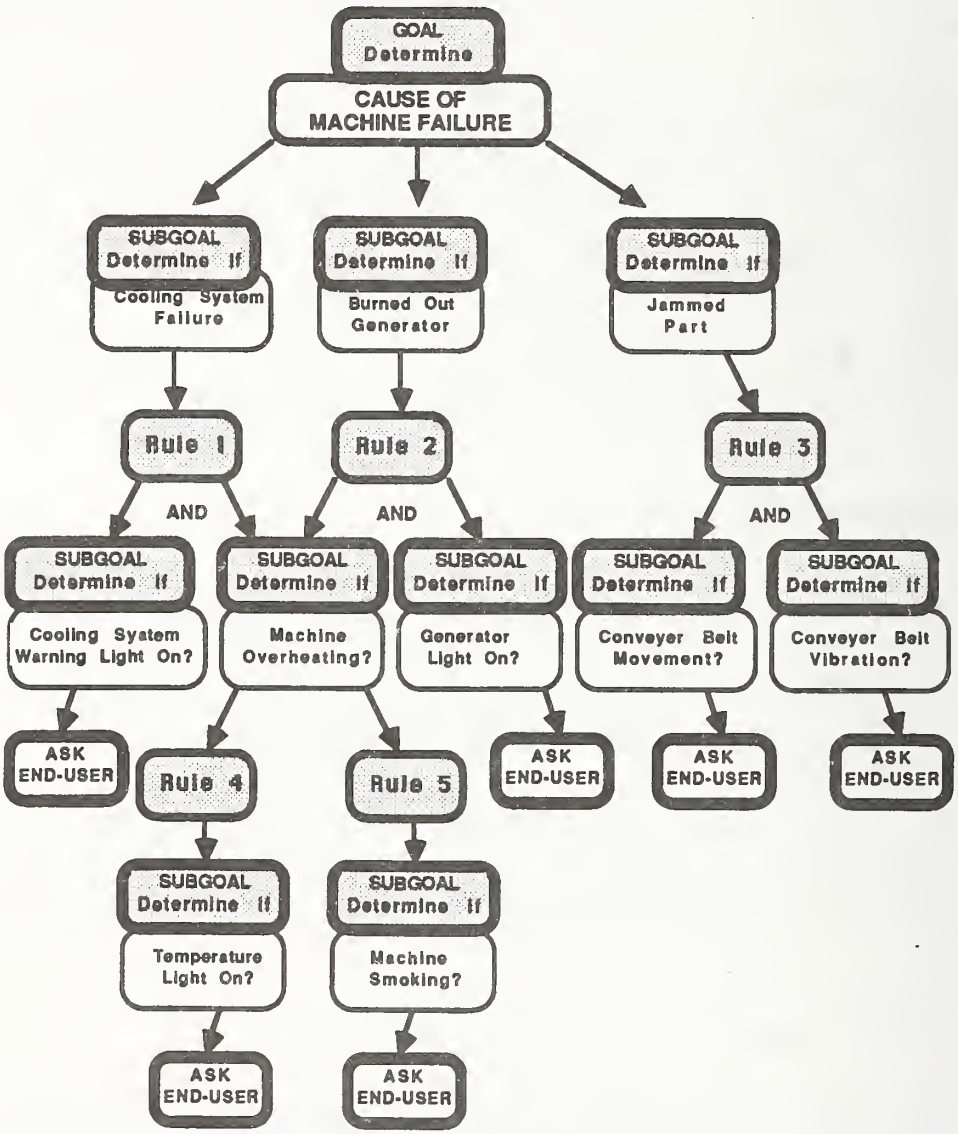

Figure 8.1. Model of Goal Directed Backward Chaining.

When the subgoals associated with a rule's IF part are fulfilled, the rule conclusions can be added to the context file. These conclusions can then be used to satisfy a previously established subgoal. In this way, the direction of the flow is reversed, leading back to the original goal.

Figure 8.1 depicts the structure of a knowledge base and the interconnections between rules. This structure forms a graph in which subgoals and rules are the nodes, while the edges are formed by the connections between rules and subgoals. Backward chaining attempts to find a solution by creating a path through this graph in which all the subgoals in the path are satisfied. This path then constitutes a complete solution. However, to find the solution path, many alternative paths may have to be tried. In a large backward chaining system, each path can be quite lengthy. Finding the right path (or paths if multiple solutions exist) requires search.

8.1.2 Supporting Search in Backward Chaining Systems

The concept of search in expert systems was introduced in section 3.4. It is possible to describe how search works in a backward chaining system in terms of subgoals and paths. In the graph depicted in figure 8.1, a subgoal may be fulfilled by more than one rule. Backward chaining on any one of these rules initiates a different path through the knowledge base. (For instance, in fig. 8.1, Rules 1 and 2 initiate alternative paths for fulfilling the initial goal.) If a particular subgoal along the path cannot be satisfied, the result is a dead end. (For instance, if the cooling system light is OFF, the path initiated by using Rule 1 fails.) When a dead end occurs, a new path must be tried.

A graph may be searched in more than one way. Three basic methods for doing search are listed below.

- **Depth-First Search**

  In depth-first search, the graph is traversed by successively following each path to its maximum length before exploring any other path. The order of the paths examined (which corresponds to the order in which rules are examined) is said to proceed from "left to right" as is shown in the figure 8.2. In this figure, the sequence of actions taken is indicated by the labeled numbering. The graph is searched until all possible paths are exhausted or until a solution is found.

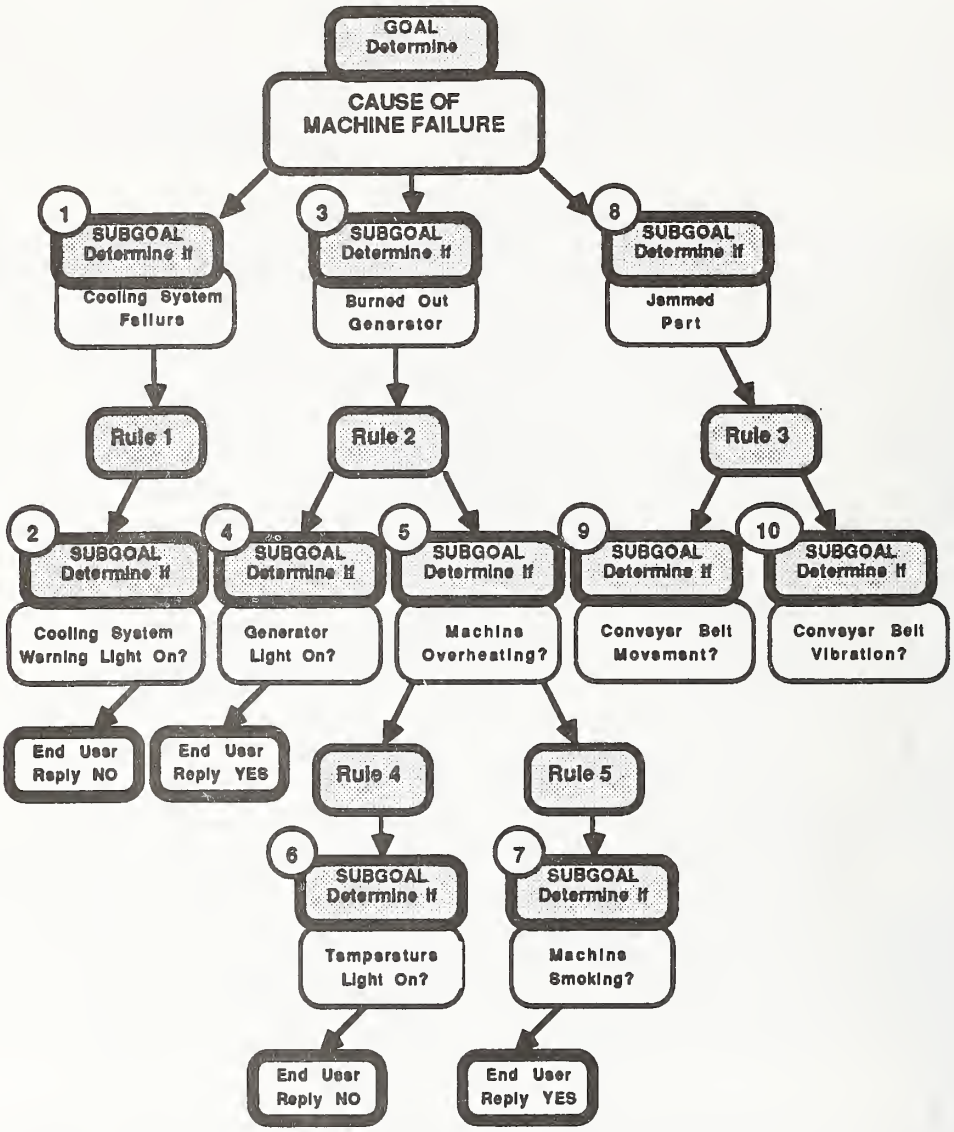

Figure 8.2. Depth-First Search.

- **Breadth-First Search**

Breadth-first search, shown in figure 8.3, is the opposite of depth-first. In breadth-first, all paths are explored at the same level before proceeding to the next level. That is, the node, or rule, at the same level in each path is explored before proceeding to the next level. The direction of the search is left to right, as the labeled numbering indicates. Breadth-first search has the effect of evenly generating a broad search pattern across the graph. As with depth-first search, breadth-first search exhaustively examines all possible paths.

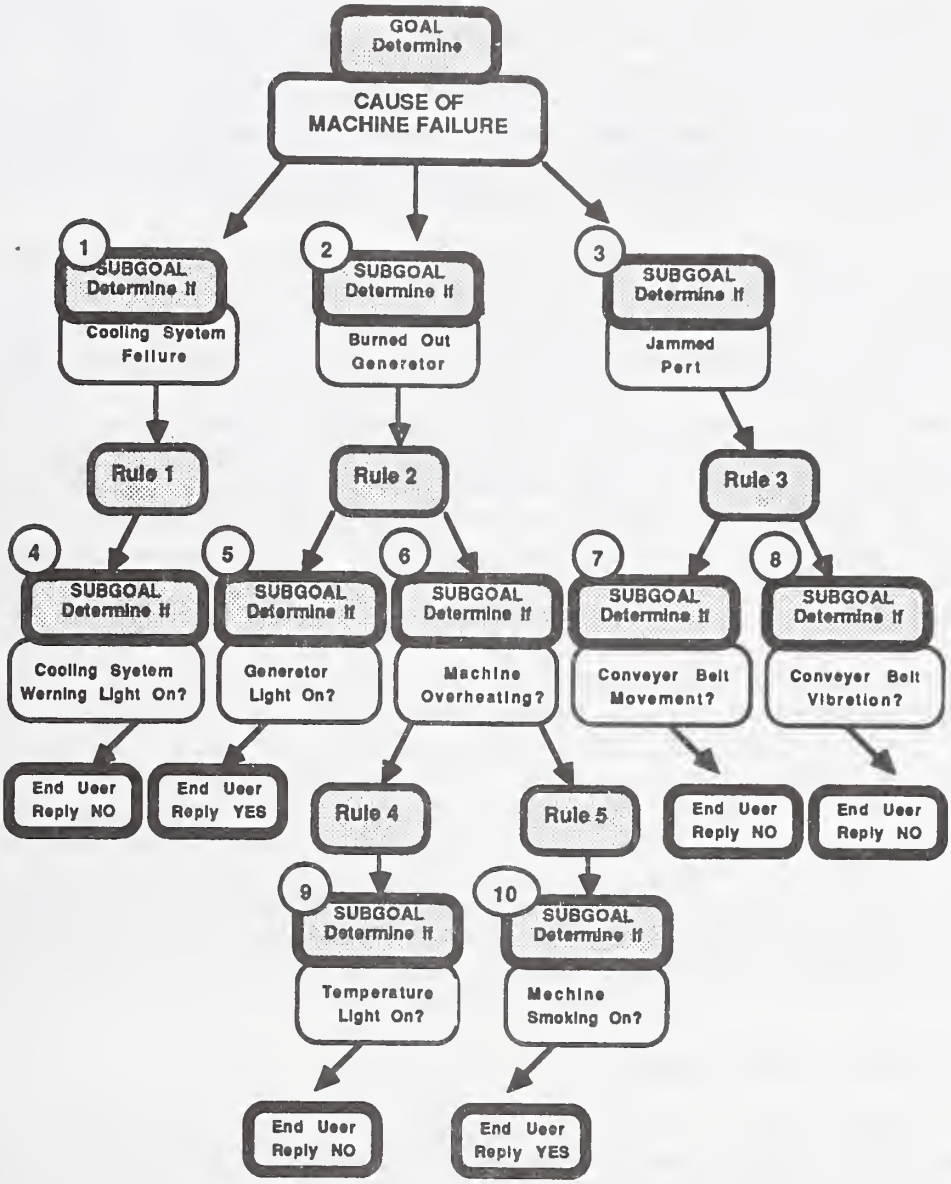

Figure 8.3. Breadth-First Search.

o **Using Heuristics to Reduce Search**

The depth-first and breadth-first procedures are designed to search an entire graph exhaustively. To reduce search effort, heuristic knowledge may be used to try to pick the "best" path leading to a solution. That is, the rule representing the path judged to have the best chance of resulting in a solution is selected first. For instance, if conveyer belt jamming is known to be the most frequent cause of machine failure, the

path initiated by Rule 3 might be tried before Rules 1 and 2. The same decision may be repeated at other subgoal nodes in the graph (such as for machine overheating with Rules 4 and 5). If more than one solution is possible (as it frequently is in diagnostic problems), less promising paths may be tried later. In addition to reducing search effort, use of heuristic knowledge has the effect of more closely emulating an expert's behavior in solving a problem.

However, backward chaining does not always follow one of these search procedures exactly. Sometimes, the end user may wish to restart the expert system program and remove or change certain facts that have been provided. This may be necessary if incorrect or contradictory information has been entered in response to a question. To meet this need, ESBTs often provide a "system restart" feature. Using this feature, search can be interrupted by the end user and selected facts can be removed or replaced in the expert system's context file. When the system is "restarted," the inference engine resumes backward chaining at an appropriate subgoal. (Sec. 10.1 discusses other end user options in making responses.)

Many rule-based ESBTs that support backward chaining use depth-first search. In simple tools that permit only depth-first search, heuristics can be used to place the rules in the knowledge base in a particular order. (Since the inference engine will examine rules in the order that they were stored and subsequently compiled, it is advantageous to place the rules most likely to succeed first.) However, for larger expert systems with many rules, a more sophisticated use of heuristics is required to reduce search. This is achieved by providing explicit control knowledge to the inference engine.

8.1.3 Control Knowledge

In a backward chaining system, control knowledge represents the heuristics used to select the rule most likely to lead to the solution; i.e., the rule that represents the most promising path. As such, control knowledge is sometimes considered to be of a "higher level" than the rest of the knowledge base. Therefore, it is sometimes referred to as meta-knowledge. ESBT support for control knowledge (or meta-knowledge) is particularly useful in large knowledge bases with many rules. Control knowledge can take several different forms. Some of these are described below.

- **Rule Prioritization**

  ESBTs may permit the knowledge engineer to assign numeric priorities to rules. When two or more rules have their conditions satisfied, priorities may be used to determine which rule should be used.

- **Built-In Control Strategies**

  ESBTs may provide "built-in" criteria for selecting a rule based on rule characteristics, such as the number of individual conditions that must be satisfied. Several alternative methods may be provided by the ESBT, one or more of which can be chosen by the developer.

- **Meta-Rules**

  A few ESBTs allow explicit specification of control knowledge in rules. Such rules are called meta-rules (after meta-knowledge). Meta-rules are maintained separately from the rest of the knowledge base. During the operation of an expert system, meta-rules may specify or change priorities among the "regular" rules in the knowledge base. Meta-rules may also state conditions under which other rules are used.

At the present time, relatively few microcomputer-based ESBTs support features for explicit representation of control knowledge. However, even without control knowledge capabilities, it is sometimes possible for a knowledge engineer to modularize and partition the knowledge base to control the way in which search is done.

8.2 Forward Chaining

In contrast to goal-directed backward chaining, forward chaining proceeds in the opposite direction--from facts to rules. Among microcomputer-based ESBTs, forward chaining inference is somewhat less prevalent than backward chaining.

8.2.1 The Forward Chaining Process

Forward chaining begins with a given set of facts describing the problem and a set of production rules. A goal condition can be specified which, when achieved, causes inference to terminate. Forward chaining can be described in the following steps:

(1) The conditions in the IF part of each rule are compared against known facts to determine which rules have their conditions satisfied. In a forward chaining system, rules whose conditions are entirely satisfied are said to have been triggered. If no rule can be satisfied, inference terminates.

(2) Otherwise, the rule(s) whose conditions are satisfied are collected and ordered in a conflict set.

(3) If more than one rule is triggered, the inference engine must decide which rule(s) in the conflict set should have its conclusions added to the expert system's context file. In a forward chaining system, such rules are said to be "in conflict." The process of determining which rule to use is known as conflict resolution. In forward chaining systems, the control knowledge for selecting the rule to use is called a conflict resolution strategy. Different conflict resolution strategies may be employed.

(4) The rule selected through conflict resolution is said to fire. The facts concluded by the selected, or fired, rule are added to the context file.[6] If the conflict set contains other rules, their conditions now may no longer be satisfied after the selected rule changes the facts in context file. Such rules may be removed from the conflict set. (For instance, the conclusion of a fired rule may result in changing a variable value that previously satisfied a condition of another rule in the conflict set.)

(5) If a goal condition has been met by the newly concluded facts, inference can stop. Otherwise, the cycle repeats beginning with step 1.

Because the conclusion of facts causes examination of rules, forward chaining is sometimes described as being data driven.

8.2.2 An Example of Forward Chaining

This section illustrates the forward chaining process using a very simple example of a construction type expert system. (Construction type expert systems were described in sec. 4.1.2.) Figure 8.4 shows a simple knowledge base of 10 rules to select compatible components and assemble a factory machine. The rules select a platform and a conveyer belt (Rules 1 and 2), a motor (Rules 3, 4, 5, and 6), and a connector between the motor and the conveyer belt (Rules 7, 8 and 9). Rule 10 specifies a goal condition.

Please note that Rule 7 contains "OR" conditions in its IF part. For this rule to be satisfied, the value of the MOTOR variable can be equal to either "Motor_X2," "Motor_X3," or "Motor_V23," while the CONNECTOR_TYPE must be "Unknown." "OR" conditions are discussed in section 9.1.

---

[6]In a forward chaining system, rule conclusions may also specify retraction of facts. A retracted fact is removed from the context file.

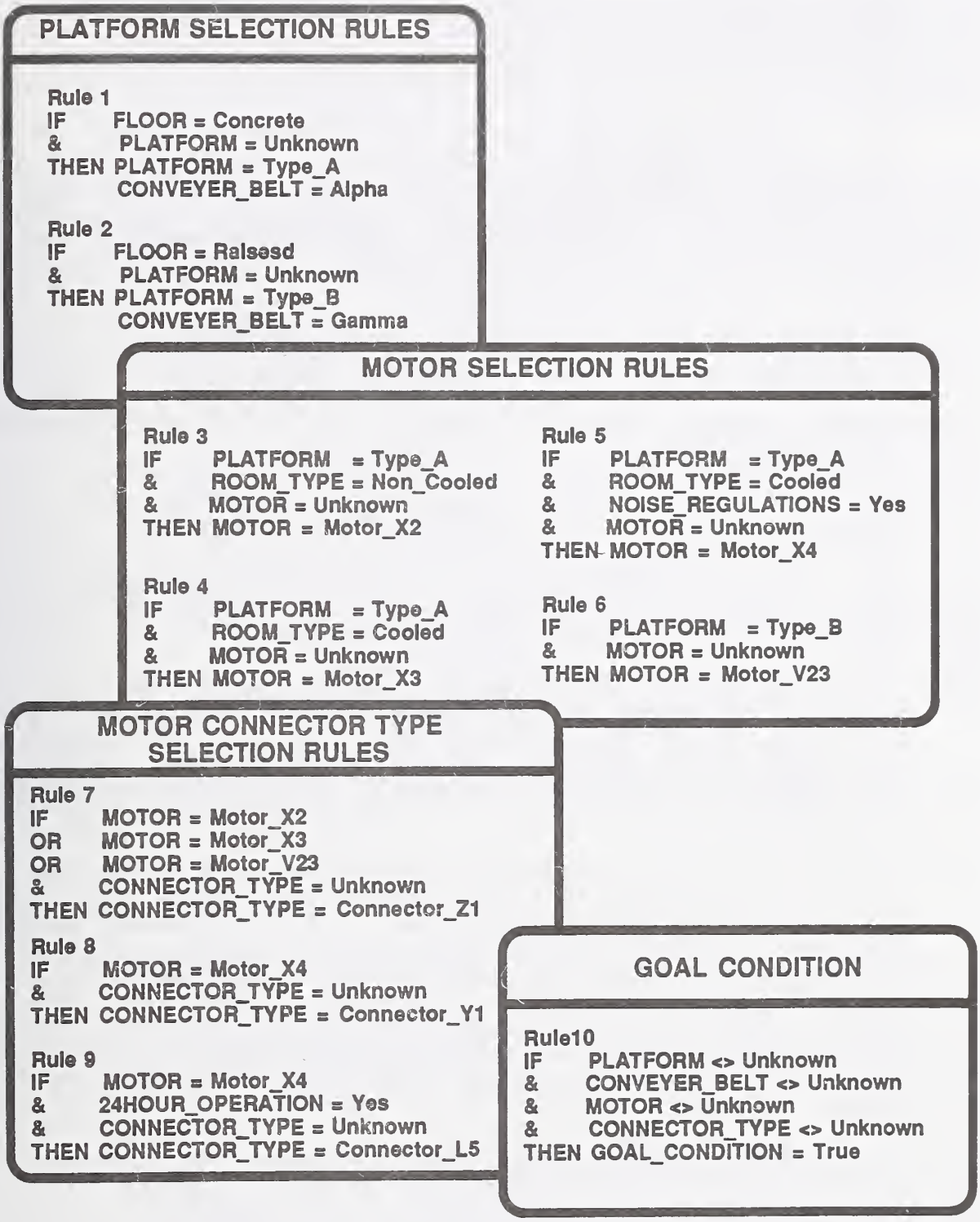

Figure 8.4. Rules for Assembling Machine Components.

The inference engine of this expert system utilizes the method of forward chaining described above. The conflict resolution strategy determines that if two rules are in conflict, the more specific rule--or the rule with more conditions in its IF part--is selected.

Let's assume the expert system is asked to select components for assembling a machine in a cooled factory room with a concrete floor. Furthermore, the factory has noise restrictions in effect and must operate on a 24-hour basis. Thus, the following facts are placed in the context file of the expert system when it is first activated:

```
FLOOR = Concrete          ROOM_TYPE = Cooled
NOISE_REGULATIONS = Yes   24HOUR_OPERATION = Yes
```

To perform its task, the expert system must determine the value of the following variables:

```
PLATFORM = Unknown        CONVEYER_BELT = Unknown
MOTOR = Unknown           CONNECTOR_TYPE = Unknown
```

The steps in the expert system's solution to the problem, in which the forward chaining process outlined in section 8.1.1 is repeated four times, can be described as follows:

(1) Initially, only the conditions of Rule 1 are satisfied by the value of the FLOOR fact, and this rule is triggered. Since it is the only rule in the conflict set, it fires, concluding that the PLATFORM should be "Type_A" and that the CONVEYER_BELT should be an "Alpha." These facts are added to the context file.

(2) Rules 4 and 5 are triggered by the value of PLATFORM and the values of ROOM_TYPE and NOISE_REGULATIONS facts. These rules are placed in the conflict set. Since Rule 5 is more specific, it fires, concluding that the MOTOR should be of type Motor_X4. This fact is added to the context file.

(3) Rules 8 and 9 are triggered by the values of MOTOR and 24HOUR_OPERATION and put in the conflict set. Rule 9 is more specific and therefore fires, concluding that CONNECTOR_TYPE should be Connector_L5.

(4) This satisfies the goal condition (Rule 10). The expert system terminates execution.

The expert system has constructed a machine assembly consisting of a "Type_A" platform and an "Alpha" conveyer belt. Motor_X4 was selected for the Motor slot. Connector_L5 was selected as the connector between the motor and conveyer belt.

This is a simple example. ESBTs that support forward chaining as a primary strategy often support representation of information using more complex data structures and carry out inference through pattern matching. These features are discussed in chapter 9.

8.2.3 Supporting Search in Forward Chaining Systems

In a forward chaining system, interconnections between rules can be complex. As with backward chaining, these interconnections form a graph (not shown) that represents the structure of a knowledge base and the interconnections between rules and facts.

As the example above shows, the conclusion of a new fact can result in the triggering of several rules, each of which represents the beginning of a new path through the graph. When a rule is selected through conflict resolution, a new path is initiated in the graph.

In principle, forward chaining systems may search the graph using the depth-first or breadth-first procedure. However, conflict resolution strategies usually rely on some form of heuristic information to select a path to follow. Conflict resolution may be based on one of the kinds of control knowledge discussed in section 8.1.3. In the machine assembly example, the conflict resolution strategy required selection of the more specific rule; i.e., the rule with more conditions. The more specific rule was selected because it was more likely to be applicable to a situation than a more general rule.

ESBTs that support forward chaining may provide several alternative conflict resolution strategies. The developer may choose one strategy for a particular application or, in some cases, even define a new strategy.

8.2.4 Daemons

Daemon forward chaining refers to a restricted form of rule chaining. Daemons or demons are forward chaining rules that are triggered by an event that occurs while an expert system is executing. Daemons may be used to display messages or give warnings. In some ESBTs, daemons can trigger other daemons only under restricted conditions, thus limiting the amount of chaining that can be done.

Unfortunately, the term "forward chaining" is sometimes used loosely. Forward chaining usually refers to the inference procedure described in section 8.2.2. This is the most commonly accepted meaning of forward chaining. However, "forward chaining" is used by some to refer exclusively to daemon capabilities. In still other cases, forward chaining may be emulated using methods

that are different than those described in section 8.2.2. Thus, different usages of the term "forward chaining" can cause confusion.

8.3 Other Inference Strategies Supported by ESBTs

Two other less common inference strategies may be supported by ESBTs. These are described below.

- **Mixed Forward and Backward Chaining**

  A mixed forward and backward chaining ESBT can support both strategies equally. The developer may create an expert system that uses either strategy exclusively. Or an expert system can be developed that utilizes both strategies together--hence, the term "mixed inference." In a "mixed inference" system, the two inference strategies can be used cooperatively. For example, the expert system can begin solving a problem by backward chaining until a particular event happens, such as the conclusion of a new fact. This event then triggers forward chaining inference. After a sequence of forward inferences terminates, backward chaining can resume. A forward chaining rule can also initiate backward chaining by establishing a new goal.

- **Blackboard Systems**

  Blackboard systems use a multiplicity of inferencing structures to solve problems. In a blackboard system, these inferencing structures act as autonomous agents that interact through a common memory area, called a blackboard. Typically, each agent is specialized to solve part of the problem and has its own knowledge base and inference strategy. As a particular problem is being worked on, the blackboard contains information about the problem and the current state of the solution being developed. Each agent monitors the blackboard and may use its expertise to propose actions that are relevant to finding a solution. A controlling agent manages the solution of the problem by determining which agent's action should be executed. For interested readers, appropriate references are [ENGE88], [HAYE83], [NII86].

Smaller ESBTs available for microcomputers generally do not support mixed inference strategies, though more sophisticated microcomputer-based ESBTs are beginning to appear that do. The potential user of ESBTs should examine this capability carefully to understand the method of operation.

Microcomputer-based ESBTs that directly support blackboard strategies are also less common. Both of these strategies normally require considerable software skills to use.

8.4 Support for Reasoning Under Uncertainty

In some expert system domains, problem solving based on heuristic knowledge is inherently uncertain. For instance, in factory machine diagnosis, heuristics are used to link observed symptoms to possible causes. Since in most cases the actual state of the machine is unknown, the cause of failure cannot be determined with absolute certainty. More than one cause may appear to be possible. Under these circumstances, it is appropriate to conclude that each possible cause of malfunction has an element of uncertainty. It is also appropriate to estimate what the level of uncertainty is for each cause. Hence, some ESBTs provide the capability to represent uncertainty.

In recent years several coherent systems for representing and reasoning under uncertainty have been proposed. The most common method employed by ESBTs is based on certainty factors, sometimes called confidence factors. Certainty factors are provided by many ESBT products, including those that are microcomputer-based, and have the advantage of being relatively easy to learn and use.

8.4.1 Certainty Factors

A certainty factor is a numeric coefficient that describes the strength of belief assigned to a fact. That is, a certainty factor specifies the level of confidence with which a fact is believed. The expert system may use rules to assign certainty factors to individual facts.

Typically, certainty factors are specified within a range, such as -1.0 to +1.0. A certainty of -1.0 corresponds to total disbelief that a fact is true. A certainty of +1.0 corresponds to absolute certainty that the fact is true.

Figure 8.5 shows an example of how a certainty factor (CF) may be assigned by a rule. The rule states that if its conditions are satisfied, the fact "Generator_Burned_Out" is concluded with a certainty factor of 0.7. In a range of -1.0 to +1.0, 0.7 may be interpreted as signifying a high level of confidence.

```
IF      GENERATOR_LIGHT = On
 &      MACHINE_OVERHEATING = True
THEN    PROBLEM = Generator_Burn_Out  CF = 0.7
```

Figure 8.5. A Rule With a Certainty Factor.

The combination of inference and certainty factors is known as evidential reasoning. A simple example of evidential reasoning occurs when two different causes of machine failure are concluded, each having a different level of certainty. First suppose one rule is used to conclude generator failure with a certainty of 0.70. Then another rule is used to infer cooling system failure with a lower factor of 0.40. The expert system can rank these conclusions on the basis of confidence level and present the results to the end user. The rankings provide additional guidance about the possible cause of the problem.

Certainty factors were originally derived from Carnap's theory of confirmation [CARN50]. The MYCIN system adapted Carnap's theory for expert systems [SHOR85] to produce a system of evidential reasoning. MYCIN included formulas for combining certainty factors of two or more rules and formulas for propagating certainty factors during rule chaining. The formulas used to calculate certainty factors in MYCIN have since provided the basis for calculation of certainty factors in some commercial ESBTs.

8.4.2 Other Methods for Reasoning About Uncertainty

In addition to certainty factors, other methods for reasoning about uncertainty have been developed. These methods are more complex and less commonly supported by microcomputer-based ESBTs. Three methods are briefly mentioned below.

- **Fuzzy Logic**

  Fuzzy set theory, developed by Zadeh [ZADE65], provides a mathematical method for representing ambiguous or vague concepts such as "coolness" or "warmth." In fuzzy set theory, such "fuzzy" concepts are quantified using fuzzy sets--a graded set membership function expresses the degree of ambiguity or "fuzziness" of a proposition such as "The machine is overheating at 130°." Fuzzy set theory also provides set

operations that can be applied to fuzzy sets. Fuzzy logic extends fuzzy set theory to permit inexact or approximate inference with fuzzy concepts. The utility of fuzzy logic for expert systems has been described in [YAGE84]. Fuzzy logic is supported by some microcomputer-based ESBTs [SCHW91].

- **The Dempster-Shaffer Theory**

  The Dempster-Shaffer theory of uncertainty was developed by A. P. Dempster [DEMP67] and extended by Glen Shaffer [SHAF76]. Formally based on probability theory, the Dempster-Shaffer theory attempts to describe uncertainty using ranges of probabilities instead of one number. At present, methods of uncertain reasoning based on the Dempster-Shaffer theory are supported by very few ESBTs.

- **Bayesian Methods**

  Bayes theorem, which has given rise to extensive research in Bayesian inference and Bayesian decision making, is based on the use of conditional probability; that is, the probability of one event occurring given prior knowledge of the probability of one or more other events occurring. Bayesian methods have been applied to expert systems, including the Prospector research expert system [DUDA79]. However, the application of Bayesian inference is based on exact knowledge of the probabilities of events. In some practical problems, the number of events can be numerous, and the data necessary to establish reliable probabilities is often inadequate.

A comparative analysis of these methods and certainty factors is found in [HENK88]. Of the three methods described in this section, fuzzy logic is perhaps finding the most real-world application. Fuzzy logic has also been applied to problems outside expert system technology such as industrial process control [MAIE85].

8.5 Inductive Systems

Inductive ESBTs were briefly mentioned in chapter 6. Induction is a procedure by which rules are automatically generated from a set of examples [QUIN79]. Each example contains a problem situation and an associated solution. Typically, each example has a set of attributes, each of which has a value. The inductive process is based on an algorithm that analyzes the examples and determines the degree to which each attribute is used in the decision-making process. The attributes are ordered by the extent to which they influence the decision making. This ordering then serves as the basis for the generation of a decision tree.

Table 8.1 displays a set of examples associated with a small induction problem, while figure 8.6 shows the structure of the resulting decision tree.

Table 8.1. Set of Examples for an Inductive System

No.	Conveyer Belt Problem?	Temp. Light On?	Machine Smoking?	Cooling System Light On?	Generator Light On?	Malfunction
1)	No	No	Yes	No	Yes	Generator Burn Out
2)	No	No	Yes	Yes	No	Cooling System Failure
3)	No	Yes	No	No	Yes	Generator Burn Out
4)	No	Yes	No	Yes	No	Cooling System Failure
5)	No	??	??	No	No	Unknown
6)	No	No	No	??	??	Unknown
7)	Yes	??	??	??	??	Jammed Part

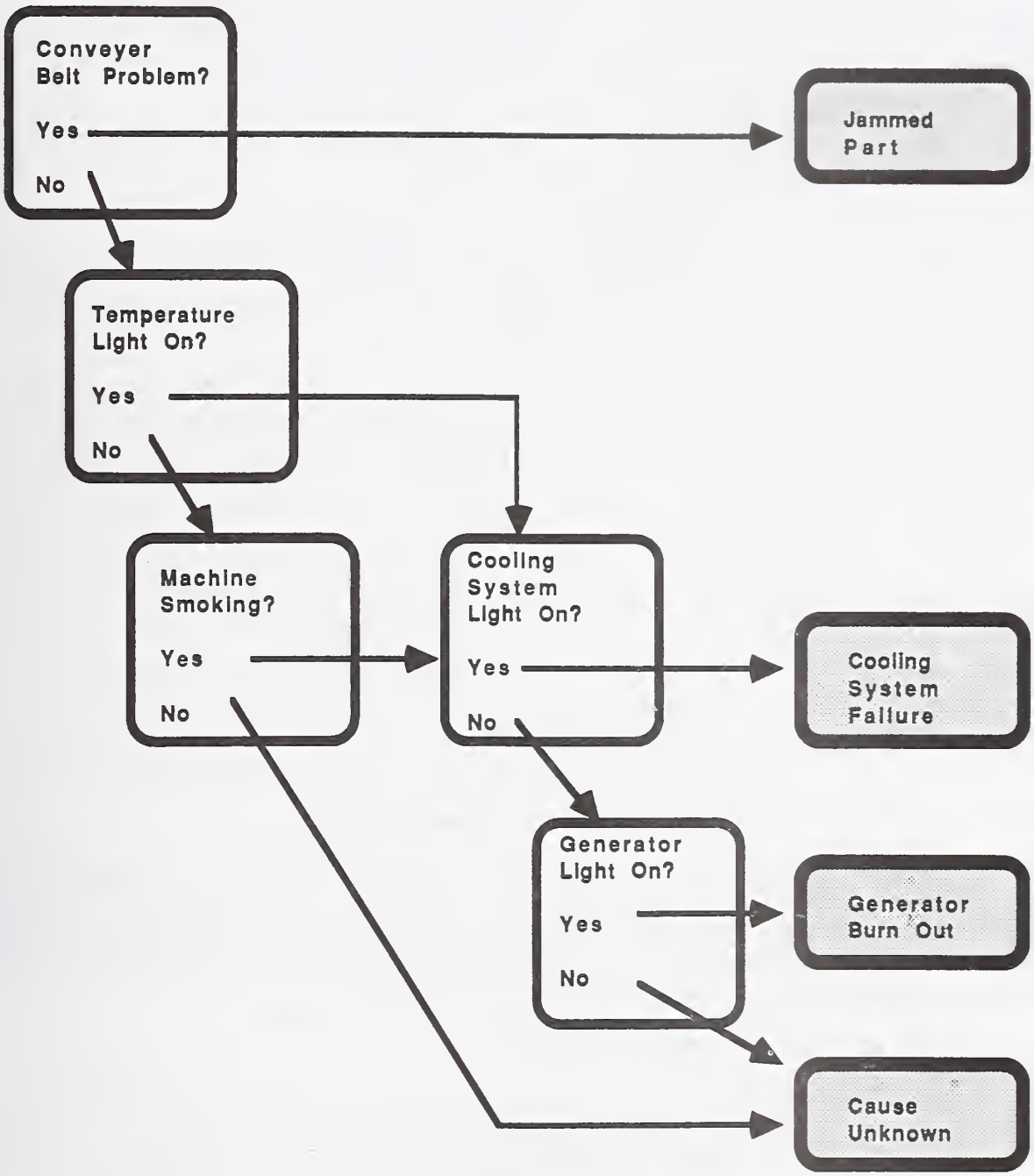

Figure 8.6. The Structure of the Resulting Decision Tree.

As the example shows, the structure of the induced decision tree is essentially hierarchical. Solutions are reached by taking paths through the tree. In some tools, an equivalent set of production rules can be generated. Once induced, the decision tree or equivalent set of rules can be applied to find solutions for individual instances of a problem. From the end user's point of view, the inductive expert system operates in a manner similar to a rule-based system.

ESBTs that use induction as a primary method are commercially available [BIEL88], [MART88]. These systems support the ability to create multiple decision trees that can be linked together. Some rule-based commercial ESBTs have also added an inductive component for generating production rules.

Induction is a convenient method for developing expert systems in which problem-solving knowledge can be described in a series of examples. For induction to be effective, the examples should cover all or most of the possible attribute and value combinations and provide a solution for each. However, if the problem-solving knowledge cannot be expressed in examples, or if the examples cover a small portion of the range of attribute value possibilities, induction systems are less useful. Under these circumstances, other forms of knowledge representation may be more appropriate.

ESBTs that utilize induction are useful to developers that have little or no programming skill or to domain experts who wish to develop expert systems themselves.

8.6 Analysis of ESBT Inference Capabilities

In this chapter, alternative inference capabilities offered by ESBTs have been examined in detail. Interested readers, including those tasked with selecting ESBTs and implementing expert systems, may wish to know when each strategy is appropriate. The following provides some brief guidelines.

- **When to Use Backward Chaining**

  Backward chaining is most useful for problem-solving tasks that have relatively few solutions and a moderate to large amount of information about the problem to examine. These conditions often occur in problems where solutions are selected and where heuristic classification is employed (discussed in ch. 4). Hence, many expert systems that select solutions employ backward chaining. For problems with few solutions and a small amount of problem information, forward chaining may also be used.

- **When to Use Forward Chaining**

  In contrast, forward chaining is most often used when the number of potential solutions is as large or larger than the amount of information about the problem. Forward chaining is particularly desirable for larger construction type expert systems that must consider components in alternative configurations. Such systems may require representation of, and reasoning about large amounts of structured information having complex patterns and interrelationships. These systems are examined more closely in the next chapter.

- **When Is Control Knowledge Useful**

  Control knowledge is useful for large expert systems in which a great deal of search is required. Small expert systems which have traditionally been implemented on microcomputers generally do not require ESBTs that support control knowledge.

- **When Is Induction Useful**

  Induction is useful for problem domains that can be described in a series of examples in which problem characteristics can be mapped directly to solutions. The examples should cover nearly all possible cases. Induction is particularly appropriate when the implementor has little or no programming skill.

For selection type expert systems with relatively few rules, an ESBT that supports backward chaining is used most often.

8.7 Summary of Features Supporting Inference Capabilities

Table 8.2 summarizes the features used to support basic inference capabilities. Each feature is characterized by:

- The level of computer skill required to use it effectively.
- Frequency of occurrence of feature among microcomputer products.
- The major category of expert system application that the feature can be used to support.
- Whether or not the feature is especially useful for development of large applications.

The categories used for level of skill were described in section 7.7. Resource requirements are rated "Low," "Medium," and "High," also as described in section 7.7.

Table 8.2. Summary of Features Supporting Inference Capabilities

Feature	Required Skill Level	Occurrence Of Feature	Suitable Application Type	Supports Large Application Development?
Support For Backward Chaining Depth First	Computer Specialist	Common	Classification	Possibly
Support For Backward Chaining w/Heuristics	Computer Specialist	Less Common	Classification	Yes
System Restart Facility	Computer Specialist	Less Common	Classification	No
Daemons	Computer Specialist	Common	Classification	No
Certainty Factors	Computer Specialist	Common	Classification & Construction	No
Fuzzy Logic	AI Specialist	Rare	Classification & Construction	No
Induction	Computer Literate	Becoming Common	Classification	No
Support For Forward Chaining	Computer Specialist	Less Common	Classification & Construction	Possibly
Support For Mixed Inference	Computer Specialist	Less Common	Classification & Construction	Possibly

9. SUPPORT FOR KNOWLEDGE REPRESENTATION

This chapter focuses on the different forms of knowledge representation found in microcomputer-based ESBTs.

The most basic form of knowledge representation is the production rule, introduced and discussed in earlier chapters. In the production rules discussed so far, facts have been represented by individual variables. Rule conditions have been represented by variables that are compared to specific values. (The comparison checks if a fact is known.) Rule conclusions have been represented by variables that are assigned values. (A new fact is concluded.)

Other forms of knowledge representation represent factual information by grouping related pieces of information into a single data structure that can be viewed and processed as a unit. These structures are complex and may consist of many variables. An example might be a data structure that holds information about a machine component, such as its name, weight, and position. Such structures provide a basis for more complex systems of knowledge representation that are useful in many domains, including manufacturing, medicine, and CAD.

Production rules that reason about information contained in such structures are often large and elaborate. Correspondingly, the algorithms necessary to support inference with these rules can be complex. For large applications with many structures, the inference process can consume considerable computer resources.

In the early 1980s, ESBTs that supported these capabilities operated mostly on larger platforms and were implemented in languages such as LISP. The few microcomputer-based ESBTs that offered similar capabilities were hindered by poor performance in larger applications. In recent years, more powerful microcomputer processors have been introduced. At the same time, microcomputer-based ESBT software has become more efficient, often implemented in programming languages designed to enhance performance. As a result, microcomputer-based ESBTs have expanded the knowledge representation and inference capabilities they support.

This chapter first reviews representation of knowledge in production rules. Then, data structures for representing complex information are discussed. Use of complex structures in production rules is presented. Use of pattern matching techniques to support inferencing is explained. ESBT support for pattern matching in connection with both backward and forward chaining is discussed. Finally, frame systems and ESBT support for object-oriented programming are discussed.

Perhaps an important consideration regarding knowledge representation in ESBTs is the lack of a standard terminology. This sometimes makes understanding of features and capabilities of

different ESBTs harder. Although it is beyond the scope of this publication to present a standard terminology, it is hoped that this chapter will help promote common understanding of important concepts and terms. (Standardization in expert systems is discussed in ch. 13.)

9.1 Production Rules That Use Simple Variables

This section describes production rules that represent facts using individual variables. Production rules have already been introduced to the reader in previous examples in this report. The variables used in these rules are similar in appearance and function to programming language variables. Two basic types of rule variables may be identified.

- **Boolean Variables**

  Boolean variables can take on only TRUE or FALSE values. Almost all ESBTs support boolean variables in some form.

- **Attribute Value Variables**

  Also known as attribute/value pairs, attribute value variables can take on many different values. A simple example of an attribute value variable is Color=Blue, where Color is the attribute, while "Blue" is one of the many different values Color can have. Some ESBTs support storage of attribute value variables in arrays and permit array elements to be referenced and manipulated in rules.

Figure 9.1 shows a production rule which uses attribute value pairs and boolean variables. On the following page, table 9.1 contains some possible values that may be assigned to the rule's variables.

```
IF    GENERATOR_LIGHT = On
      MACHINE_OVERHEATING = True
THEN  PROBLEM =Generator_Burn_Out
```

Figure 9.1. A Production Rule With Variables.

Table 9.1. Some Possible Values for Production Rule Variables

Variable Name	Possible Values	Variable Type
GENERATOR_LIGHT	On, Off	Attribute Value
MACHINE_OVERHEATING	True, False	Boolean
PROBLEM	Generator_Burn_Out Cooling_System_Failure Jammed_Part	Attribute Value

To recapitulate: the IF part of the production rule can consist of conjunctive conditions, also known as AND conditions. Each condition tests the value of an individual variable. All conditions linked by an AND must be satisfied for the rule to be used. Conditions can also be disjunctive conditions, also called OR conditions. When conditions are linked by an OR, only one condition must be satisfied for the rule to be used.

In addition, individual conditions can be negative or "NOT" conditions. For instance, the rule condition - "GENERATOR LIGHT <> On" or "NOT GENERATOR LIGHT is On," is true when the variable GENERATOR LIGHT is found to be equal to a value other than "On." The use of negation sometimes varies among ESBTs and should be examined carefully when evaluating a tool.

The knowledge representation requirements of many expert system applications can be met by production rules that use individual variables to represent facts. This is partially the reason why many microcomputer-based ESBTs rely primarily on this form of knowledge representation.

9.2 Representing Complex Information

As stated at the beginning of this chapter, factual information that is naturally grouped together may be represented in data structures with multiple attributes. For instance, information about a machine component can be represented in a data structure with attributes for the component's name, the machine the component is part of, the component's weight, and its position. Such structures allow more efficient representation of complex information than is possible using only attribute value variables.

Use of structures is important for another reason--when used in conjunction with sophisticated pattern matching techniques, complex structures support more powerful inference capabilities. Pattern matching will be discussed in section 9.3.

9.2.1 Basic Structures Supported by ESBTs

Two basic kinds of structures for representing complex information may be identified.

- **Object-Attribute-Value Triplets**

  Object-attribute-value[7] (O-A-V) triplets consist of:

  (1) The object that the attribute belongs to.

  (2) The name of the attribute.

  (3) The value of the attribute.

  Several O-A-V triplets may be used to represent information about the same fact or object. A simple example of the use of O-A-V triplets might be:

  ```
  MACHINE_COMPONENT Component_Name = Generator,
  MACHINE_COMPONENT Machine_Name = Grinder_1,
  MACHINE_COMPONENT Weight = 100.
  ```

  In these triplets, the object is MACHINE_COMPONENT, its attributes are Component_Name, Machine_Name, and Weight, and the attribute values are "Generator_27," "Grinder_1," and "100" respectively.

- **Multiple Attribute Expressions**

  Instead of O-A-V triplets, structured information can be grouped in a record-like format, as illustrated below.

  ```
  {MACHINE_COMPONENT  Component_Name = Generator_27
                      Machine_Name   = Grinder_1
                      Weight         = 100
                      Position       = position-y}
  ```

  The structure describes a generator component for the machine "Grinder_1" that has a weight of 100 and is located at position y.

---

[7]The term "object" used in this context is not the same as the term "object" used in object-oriented programming. Object-oriented programming is discussed later in this chapter.

Among microcomputer-based ESBTs that support representation of complex information, multiple attribute data structures are somewhat more prevalent than O-A-V triplets.

9.2.2 Defining Generic Types of Structures

An important feature associated with the use of multiple attribute data structures is the ability to define generic types or kinds of data structures. The generic type serves as a template from which many individual instances of the type may be created. Thus, a generic type for MACHINE_COMPONENT might be defined as shown in figure 9.2 below. This type describes the set of attributes that a machine component must have. The type serves as a basis for the creation of many individual instances of a MACHINE_COMPONENT, each with individual values for different attributes. The role of a generic type or template is analogous to a database record definition where individual records that fit the definition are stored in a file.

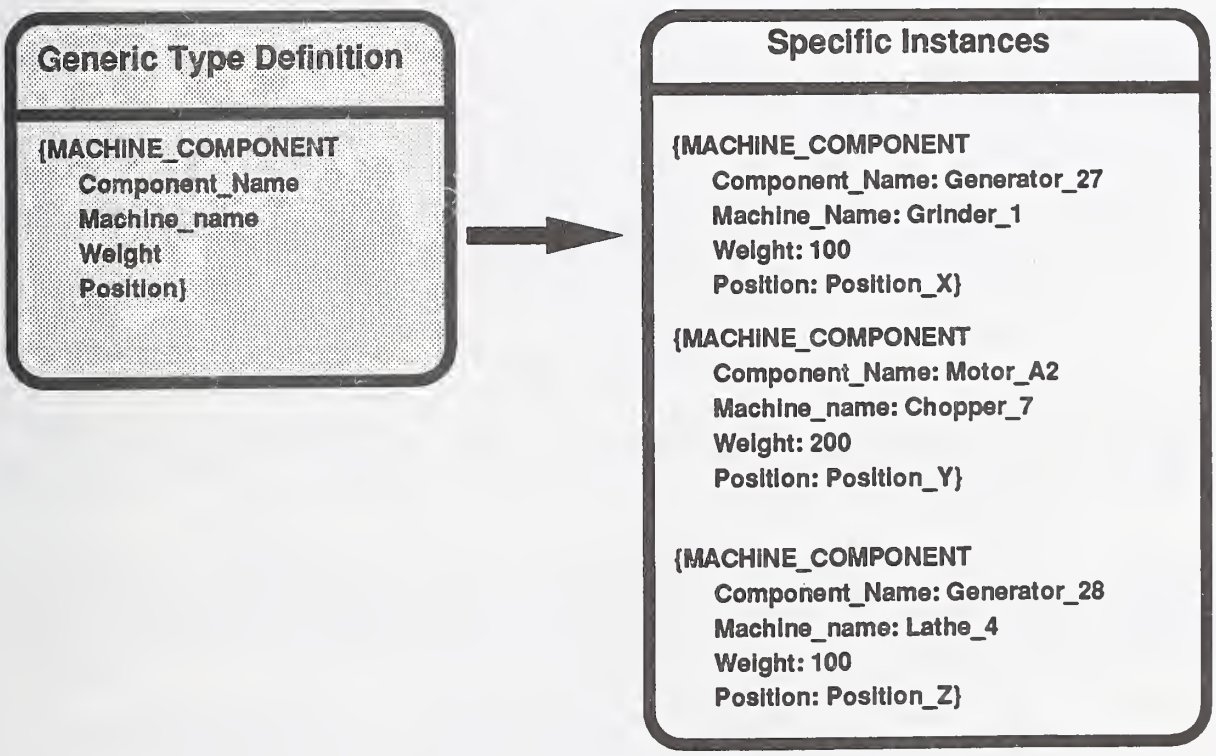

Figure 9.2. Generic Types and Instances.

The ability to represent factual information in complex expressions requires a corresponding ability to execute inference operations with these data structures. This topic is addressed in the next section.

9.3 Production Rules and Complex Information

Rules that reason about information represented in multiple attribute expressions can specify complex comparisons. For example, the IF part of a rule may contain several multiple attribute expressions and compare the values of several attributes.

In addition to direct comparisons, ESBTs may support even more powerful capabilities for doing comparisons with rule variables. These capabilities center around the ability to represent patterns and match them to facts represented in multiple attribute expressions.

9.3.1 Patterns

A pattern is a generalized data structure that can be used to match many specific instances of the same thing. Patterns can contain constants which must match specific values. Patterns can also contain variables, called free variables for the purposes of this discussion. In contrast to variables that are compared to specific values, a free variable can match many different values. The simple pattern shown below is intended to match multiple attribute expressions of the type MACHINE_COMPONENT. It has two free variables, denoted by symbols beginning with "?":

```
{MACHINE_COMPONENT  Machine_Name   = ?machine_name,
                    Component_Name = ?component_name}
```

If the attributes Weight and Position[8] are disregarded, this simple pattern can match all three multiple attribute expressions in figure 9.2 above.

In a rule, the IF and THEN parts are considered to have two separate patterns. The pattern in the IF part can be composed of several individual conditions, just as a simple rule is. For instance, the rule below has two conditions:

```
IF   {MACHINE_COMPONENT  Machine_Name   = ?machine_name,
                         Component_Name = ?component_name}
&    {OVERHEATING Component_Name = ?component_name}
THEN {SHUTDOWN Machine_Name ?machine_name}
```

---

[8]These attributes will be discarded from now on to simplify the discussion.

Each condition is intended to match multiple attribute expressions that represent specific facts. A free variable can appear in more than one place, and can thus be used to link the individual conditions in a meaningful way. For example, in this rule, the free variable ?component_name links the two conditions in the IF part by specifying that both conditions must apply to the same component. Likewise, the variable ?machine_name, which appears in the patterns for both the IF and THEN parts of the rule, is used in the same way. The entire meaning of the rule is:

> If the component "?component_name" belongs to the machine "?machine_name," and the component "?component_name" is overheating, then shut down the machine "?machine_name."

9.3.2 Pattern Matching in Production Rules

Pattern matching is the process of matching the pattern in the IF part of a rule to specific multiple attribute expressions in an expert system's context file. During pattern matching, free variables are assigned specific values. For the IF part of a rule to be satisfied, its pattern must be successfully matched. If the pattern in the IF part matches, the variable assignments--called variable bindings--can be substituted into the pattern in the rule's THEN part. This results in the creation of a new multiple attribute expression that represents the inference of a new fact. For instance, suppose the context file contains the following:

```
{MACHINE_COMPONENT  Machine_Name   = Grinder_1,
                    Component_Name = Generator_27}
{OVERHEATING Component_Name Generator_27}
```

The pattern in the IF part of the rule shown in section 9.3.1 can be matched as follows:

(1) The free variable ?machine_name is bound to "Grinder_1."

(2) The free variable ?component_name is bound to "Generator_27."

The variable binding for ?machine_name can now be substituted into the pattern in the THEN part. The result is the creation of a new multiple attribute expression:

```
{SHUTDOWN Machine_Name Grinder_1}
```

The expression is then added to the expert system's context file.

In addition, patterns can incorporate functions, specify constraints on values that can be matched, and specify that free variables match lists of elements of variable sizes. Pattern matching, when applied on a wide scale, can have a powerful effect.

9.3.3 The Power of Pattern Matching

A rule with free variables can match different sets of facts. Each set of facts that satisfies the condition portion of a rule is said to instantiate the rule. As figure 9.3 shows, a rule can be instantiated to produce more than one conclusion.

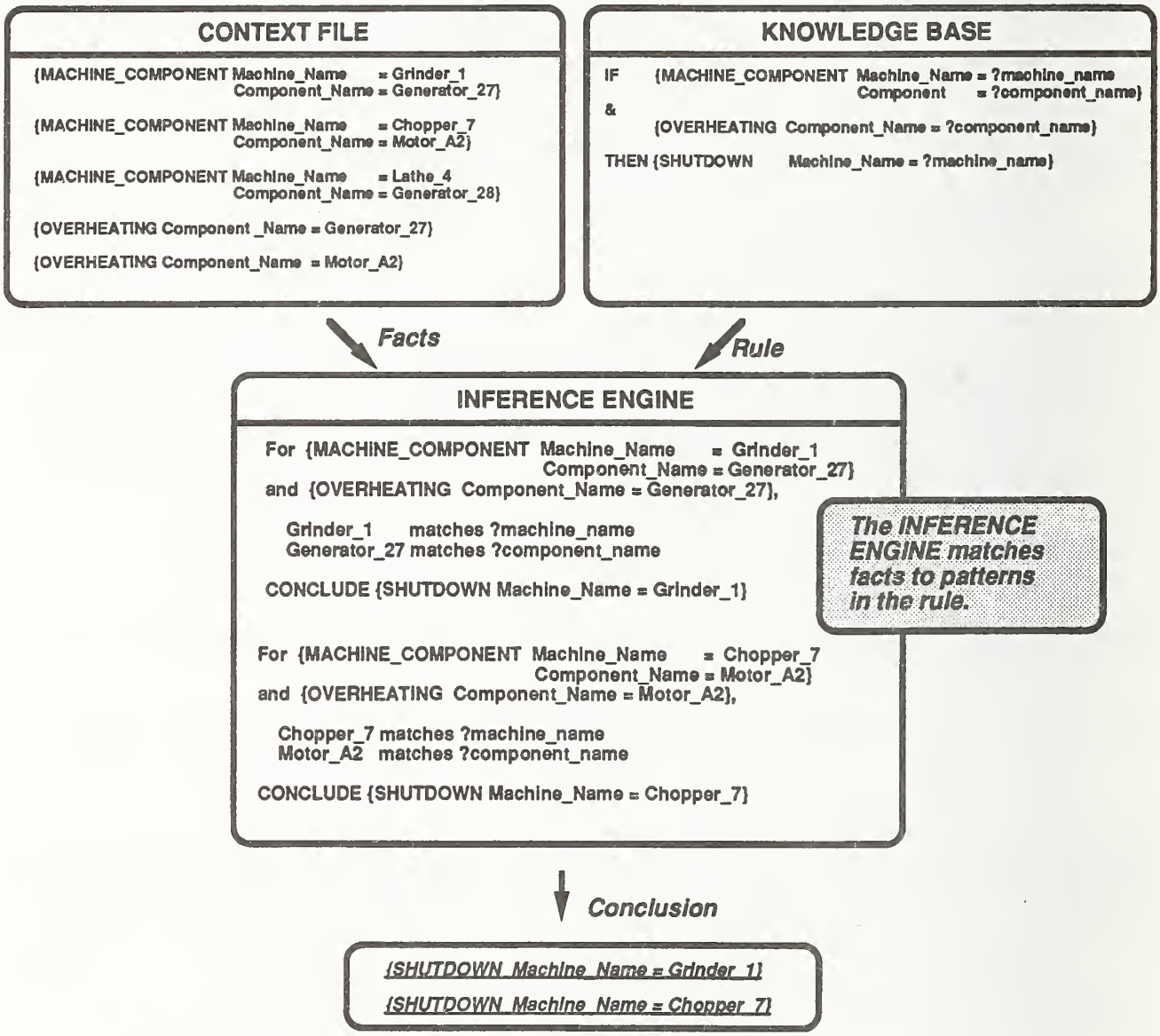

Figure 9.3. Example of Pattern Matching.

Pattern matching provides some important advantages for developing expert system programs.

- **Definition of Generalized Rules**

  Instead of having specific rules to handle each instance of a problem or situation, pattern matching permits specification of generalized rules that can be applied to many instances. For some applications, this permits the knowledge engineer to specify a smaller number of general rules instead of many specific rules.

- **Representation of Complex Relationships**

  As the example above illustrates, generalized rules can be used to match many repetitive sets of facts. Such rules can be used to reason about recurring relationships in complex physical systems, a capability necessary for many design and configuration applications.

- **Links to Databases**

  An expert system that supports multiple attribute expressions and free variable pattern matching can represent and reason about information in a record-like format. Multiple attribute expressions can be mapped into database structures such as tables. This is an advantage if an expert system must use large amounts of information stored in a DBMS. ESBT support for interfaces to DBMSs is discussed in chapter 11.

The power provided by pattern matching permits the development of many expert system applications that would be difficult to develop with only production rules that represented facts in individual variables.

9.3.4 Pattern Matching and Forward Chaining

An increasing number of forward chaining ESBTs designed for the microcomputer environment support pattern matching of multiple attribute expressions. These tools utilize the basic forward chaining procedure, or a variation thereof, discussed in section 8.3. The tools are intended for larger, more complex expert system applications. Some characteristics of these tools are summarized below.

- **Types of Applications That Are Appropriate**

  ESBTs that support forward chaining and pattern matching are particularly useful for implementing applications that must manipulate significant amounts of data, including expert

systems that perform construction type tasks. This includes expert systems that design computer systems [MCDE82], [MCDE84] or that perform planning and scheduling functions.

- **Efficiency and Use of the Rete Algorithm**

  The Rete Algorithm [FORG77], [FORG81] was originally developed in connection with the OPS5 language. This algorithm provides an efficient means of matching production rule patterns to multiple attribute expressions. Many ESBTs that support forward chaining and representation of factual information in multiple attribute expressions use the Rete algorithm or a close derivative. A detailed description of this algorithm is beyond the scope of this report.

- **Greater Computer Resource Requirements**

  Expert system applications developed with ESBTs employing a variant of the Rete algorithm generally require more processor power and more memory than applications developed with simple rule-based ESBTs. Because of these extensive resource requirements, large applications can be efficiently implemented only on the largest microcomputer platforms.

- **Control Knowledge in Large Forward Chaining ESBTs**

  Control knowledge and its role in managing search in large applications was discussed in chapter 8. Control knowledge is necessary because of the correspondingly greater resource requirements of large forward chaining expert system applications. Typical control strategies supported by ESBTs include prioritizing rules and providing built-in conflict resolution strategies (discussed in secs. 8.1.3 and 8.2).

ESBTs that support forward chaining and pattern matching provide a capability to develop large, complex applications. However, because of their greater resource requirements, applications that use this capability may sometimes have to be implemented on larger platforms.

9.3.5 Pattern Matching and Backward Chaining

Some ESBTs support pattern matching in backward chaining systems. As with forward chaining systems discussed in the previous section, backward chaining systems that support pattern matching are also intended for larger, more complex applications and have correspondingly greater computer resource requirements.

These systems utilize the basic backward chaining algorithm given in section 8.1. Instead of the Rete algorithm, pattern matching in backward chaining systems has traditionally been

implemented using the Unification Algorithm [ROBI65], or variations thereof. The unification algorithm underlies inference procedures in the Prolog language and other programming systems that support logic programming. The control strategies discussed in section 8.1.3 are applicable to backward chaining systems that support variable pattern matching as well.

In some cases, ESBTs support variable pattern matching capabilities and both forward and backward chaining.

9.4 Representing Knowledge in Frames

Frames and frame systems are partly based on representation of knowledge in multiple attribute expressions, presented in section 9.2. Like multiple attribute expressions, frames are data structures that are also "record like." The attributes of a frame are often called slots. Slots can be assigned specific values. Figure 9.4 shows an example.

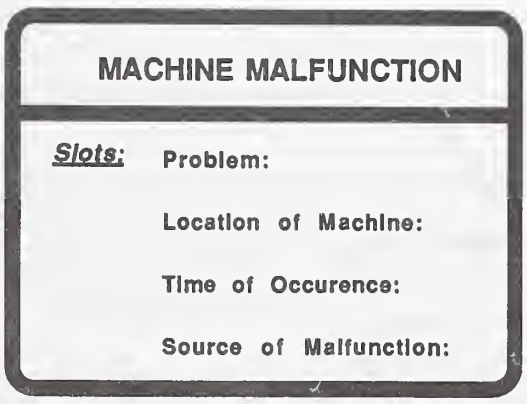

Figure 9.4. An Example of a Frame.

In addition, frame systems generally provide other capabilities for representing and manipulating information.

9.4.1 Capabilities Supported by Frame Systems

Frame systems offer the ability to define frame types and to specify links, or relationships, between them. These systems can be designed to reflect the organization of real-world phenomena.

- **Frames Types Can Be Created to Serve as "Templates"**

  As with generic types of structures (discussed in sec. 9.2.2), frame systems provide the ability to define frame "types" and to create specific instances of particular frame types. Each instance has the same structure as its type. However, the specific values that are assigned to the frame slots can differ. These assignments can be made when the frame instance is created or at any time after.

- **Frames Can Be Linked**

  Frame types can be defined to have links to other frame types. Links may describe existing real-world relationships that must be represented in a knowledge base. Thus, individual links can be interpreted to have specific meanings, such as:

  (1) Natural relationships between real-world objects such as between machines and the factories they are located in.

  (2) Pointers from frame slots to other frames that elaborate the slot description. (For instance, a frame for a machine may have a slot for a generator that points to a frame that describes the generator component.)

- **Daemons Procedures for Frame Slots**

  Frames can have Daemon procedures defined that are activated when individual slots are accessed or updated. These procedures may be used to compute values or to update other slots. Slot daemons are analogous to daemons in forward chaining rules.

Readers familiar with the database technology will recognize that frame systems bear similarity to some database data models implemented in commercial DBMS products. For instance, the ability to define frame types and to specify relationships between frame types bears similarity to the capabilities provided in the entity-relationship model [CHEN76]. It is therefore not surprising that some experimental and applied expert systems store frames in conventional DBMSs.

9.4.2 Generalization/Specialization Hierarchies

Frame types can be defined that are specializations of other frame types. For instance, it is possible to define a frame type that describes a general malfunction of a machine generator component. In a real-world system, this description can be specialized to identify particular kinds of generator malfunctions, such as failure of specific generator subcomponents. In a frame

system, it is possible to define a different frame type for each more specific malfunction. A frame for a more specific malfunction is automatically given the slots of the general malfunction frame. However, the more specific frame may also define additional slots for information about its particular malfunction.

The mechanism by which a more specific frame acquires properties from the more general frame is called inheritance. Inheritance can be used to create extensive hierarchies of frames. Figure 9.5 shows a simple example of a two element frame hierarchy.

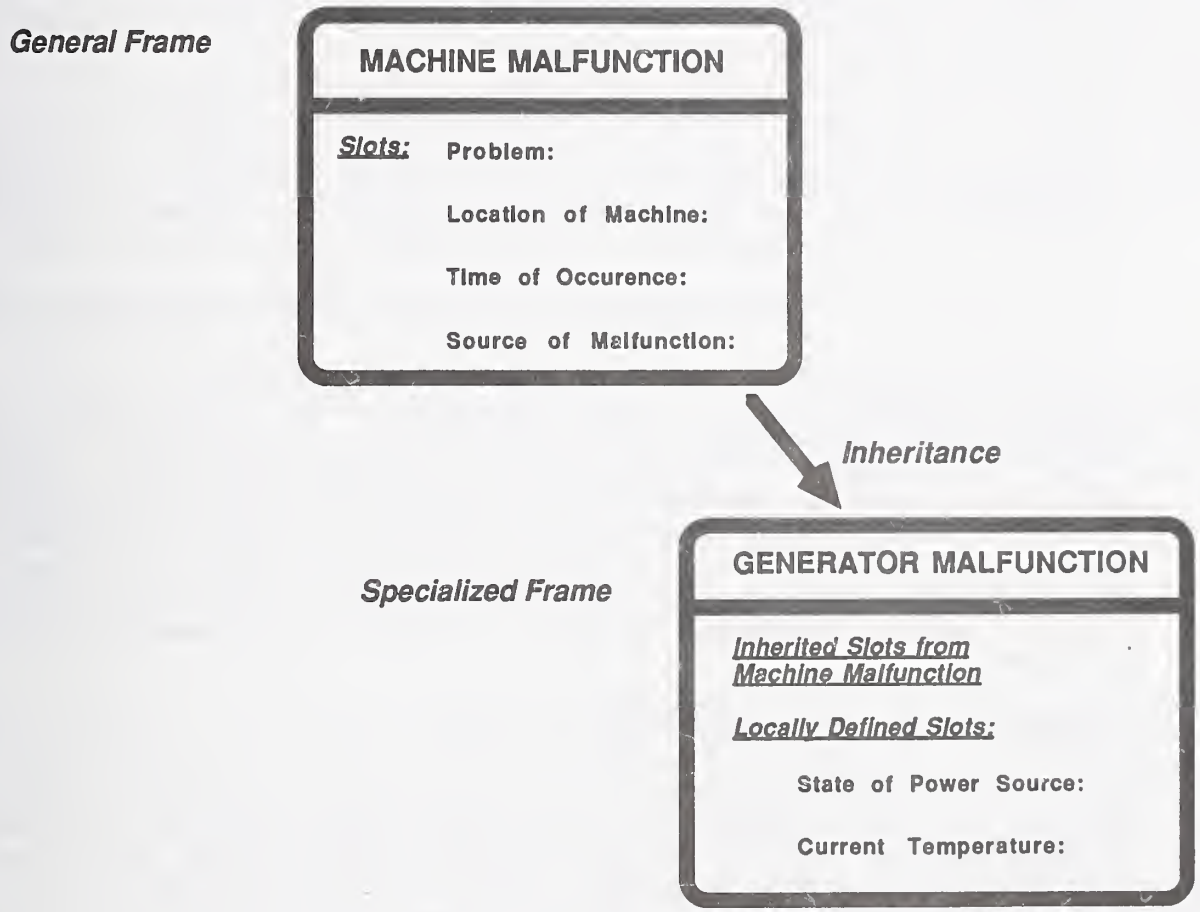

Figure 9.5. Example of a Simple Frame Hierarchy.

Most microcomputer-based ESBTs that support frame systems also support inheritance of frame types, though exceptions may exist.

Like other features of frame systems, inheritance is found in data models that underlie other programming systems. In particular, inheritance is an important feature of object-oriented programming systems and object-oriented DBMSs. Object-oriented systems and their relationship to frame systems are discussed below in section 9.5.

9.4.3 Integrating Rules and Frames

In contrast to DBMSs, frame systems interact with rules. This interaction can take place in different ways.

- **Rules Can Be Associated With Individual Frame Types**

  Frame types can be associated with specific rules or groups of rules. Rules can by used to give specific values to frame slots, to create instances of frame types, and to link frames.

- **Inference Can Be Guided by the Structure of a Frame Hierarchy**

  A hierarchical organization of frames can reflect the problem-solving method of a human expert. For instance, a generalization/specialization hierarchy of frames might be used to represent a taxonomy of machine malfunctions. This hierarchy could be used to narrow down the cause of specific cases of malfunction problems. To identify a malfunction, rules associated with a more general frame would determine a more specific frame to examine. The process could be repeated until a path is created through the hierarchy leading to a solution.

- **Frames Can Be Linked by Rule Chaining**

  Frames can be linked by rules during rule chaining. For instance, during backward chaining, conditions in a rule associated with one frame may match information stored in an instance of a different frame. In this way, backward chaining can traverse a series of frames. The traversals may follow defined real-world relationships between frames (discussed in sec. 9.4.1) instead of a generalization/specialization relationship.

ESBTs in which production rules can reference information in frames may support pattern matching. However, frame systems are not always accompanied by pattern matching capabilities as described in section 9.3.

9.4.4 Analysis of Frame Systems and ESBTs

Some frame systems supported by ESBTs provide a rich set of representational capabilities including the ability to define generalized frame types, specify relationships between frames, and create generalization/specialization hierarchies. However, these capabilities are not necessary or advantageous in every application.

Frames are useful for organizing larger knowledge bases. Associating groups of rules with frames permit modularization of expert system programs. By representing knowledge in frame hierarchies instead of rules, the overall number of production rules that must be maintained by the system can sometimes be reduced.

However, there are tradeoffs associated with use of frame systems. More complex frame languages require added system software overhead, resulting in slower execution. Large frame systems with many frame types and hundreds of rules may require larger, more powerful microcomputer platforms.

9.5 ESBTs and Object-Oriented Programming

A complete discussion of object-oriented programming is beyond the scope of this report. However, object-oriented programming systems have been incorporated in a number of microcomputer-based ESBTs and have proven to be powerful tools for developing sophisticated expert system applications. Therefore, a brief treatment of the subject is in order.

9.5.1 Summary of Features of Object Systems

Object systems are named after their most essential feature: the "object." An object can be thought of as a piece of structured information, like a multiple attribute expression or frame. However in contrast to most frame systems, individual objects have specific behavioral aspects. The behavior is carried out by procedures that perform specific operations. Taken together, the object's structure and behavior can be defined to model real-world entities. For example, a machine can be represented by an object. The object can have attributes to describe the machine's essential properties and have procedures to simulate the machine's function (and possibly malfunction).

In addition to the ability to create objects, object programming systems also have the following capabilities, many of which are similar to frame systems.

- **Definition of Object Classes**

  Object types can be defined, often called classes. Like frame types, classes can represent abstract specifications that are intended to capture the essential characteristics of real-world objects. Classes can be used to declare generic types of multiple attribute expressions (discussed in sec. 9.2). In contrast to many frame systems, a class can also define a behavioral component described by a set of procedures.

- **Creation of Individual Objects From Class Definitions**

  Class definitions serve as "templates" from which objects are created. Each object has the structure (represented by its attributes) and behavior (described by its procedures) defined for its class.

- **Computation Through Message Passing**

  Message passing is a computational mechanism that allows individual objects to communicate with each other and yet preserve their independence. Message passing is closely associated with the object's behavioral aspect. A message is directed to an object resulting in invocation of a procedure, called a method. The method performs a predefined operation that exhibits some aspect of the object's behavior.

- **Inheritance and Class Hierarchies**

  Class hierarchies can be created using the mechanism of inheritance (described in sec. 9.4.2). Like frame systems, object system hierarchies permit definition of classes that are specializations of other classes. Class hierarchies are very similar to generalization/specialization hierarchies in which both attributes and behavior are inherited.

Object programming systems possess other unique features. For a complete treatment of object programming systems, the reader may consult [DABR90], [GOLD83], [PARS89].

9.5.2 Analysis of Object Systems

ESBTs that support object-oriented programming provide both greater flexibility and generality in defining representation structures as well as the advantages of production rules.

- **The Advantages of Computationally Complete Programming Languages**

  In principle, a programming language is computationally complete, whereas a production rule system alone is not. For practical purposes, this means that there are some algorithms that can be implemented in an object-oriented programming language but cannot be implemented using rules and frame systems. Additionally, an ESBT that provides a built-in programming language eliminates the necessity of developing external software modules that perform conventional programming tasks. This allows an entire application, including conventional components, to be developed using a single self-contained software package.

- **Object Systems and Frame Systems**

  Object systems have many properties in common with frame systems. In fact, some authors consider object systems and frame systems to be essentially equivalent. However, in contrast to most frame systems, object-oriented programming systems have complete programming languages that provide powerful capabilities for implementing a wide variety of software applications. Object systems also possess features which some frame systems do not, such as message passing. The generality of object-oriented programming systems can be used to implement a wider variety of real-world systems than is possible with most frame systems. Object-oriented programming systems can be used to implement frame systems.

Production rules, frame systems, and object-oriented programming were first combined in large hybrid ESBTs that appeared on LISP machines [KUNZ84].

9.6 Summary of Features for Knowledge Representation

It should be emphasized that the capabilities discussed in this chapter are not uniformly supported by all ESBTs. For instance, some tools may support use of multiple attribute expressions (discussed in sec. 9.2), but provide incomplete support for pattern matching (sec. 9.3). Similarly, specific features of frame systems and object-oriented programming systems may vary among different ESBTS.

Table 9.2 below summarizes key knowledge representation features. Each feature is characterized by:

- The level of computer skill required to use it effectively.
- Computer resource requirements.

- The major category of expert system application that the feature can be used to implement.

- Whether or not the feature is especially useful for development of large applications.

Most of the features summarized below are currently offered in only a few, more expensive microcomputer-based ESBTs. The categories used to describe level of skill were described in section 7.7. Resource requirements are rated "Low," "Medium," and "High," also as described in section 7.7.

Table 9.2. Summary of Selected Knowledge Representation Features

Feature	Required Skill Level	Computer Resource Require-ments	Suitable Application Type	Supports Large Application Development?
Simple Rules	Computer Specialist	Low	Classification	No
Rules With Multiple Attribute Expressions	AI Specialist	Medium	Classification	No
Support For Pattern Matching	AI Specialist	High	Classification & Construction	Possibly
Support For Forward Chaining & Rete Algorithm	AI Specialist	High	Classification & Construction	Possibly
Support For Backward Chaining & Pat. Matching	AI Specialist	High	Classification & Construction	Possibly
Support For Definition Of Simple Frames	AI Specialist	Medium	Classification & Construction	Possibly
Support For Frame Hierarchies	AI Specialist	High	Classification & Construction	Possibly
Object-Oriented Programming	Computer Specialist	Medium To High	Classification & Construction	Possibly

Support for representation of complex information, pattern matching, and object-oriented programming substantially increases the complexity of expert systems that can be developed using an ESBT. Accordingly, it is generally not possible for the non-programmer to develop expert systems that rely on the more sophisticated forms of knowledge representation discussed in this chapter. Generally, a team of developers is required, consisting of one or more knowledge engineers and skilled programmers.

10. FEATURES FOR CONSTRUCTING END USER INTERFACES

This chapter is about end user interface construction facilities provided by microcomputer-based ESBTs. Perhaps the most important single factor in successfully deploying an expert system is acceptance of the end user interface by those who will use the system on a regular basis. Therefore, development of the end user interface can be a critical and time-consuming part of an expert system project. An ESBT that provides good end user interface construction facilities is important to a successful effort.

ESBTs can generally support development of three types of end user interfaces:

- The most basic are text-based end user interfaces, available in inexpensive ESBTs.
- More sophisticated end user interfaces support creation of elaborate menu and window systems. Bit-mapped computer graphics provide the ability to create pictures, charts, and drawings for display to the end user.
- Hypertext systems provide the ability to access and display large amounts of relevant information that cannot be provided easily using text-based interfaces.

In addition to text-based end user interfaces, microcomputer-based ESBTs may support either sophisticated end user interface construction facilities or hypertext or both.

In this chapter, each of these kinds of interfaces is examined in turn. Features for implementing explanation, another useful aspect of end user interfaces, are also discussed.

10.1 Features of Simple End User Interfaces

Display of character text is a basic form of communication between any computer system and an end user at a terminal. Display of text is also the basis for end user interface construction facilities supported by many microcomputer-based ESBTs.

The example in section 3.3 can be extended to show the use of character text. For instance, suppose the inference engine must determine the status of the cooling system warning light (Step 1). Since this fact is not known and is not inferred by any rule, the inference engine displays the following question:

```
Is Cooling System Warning Light On?

     YES
     NO
```

The end user may respond to the question by selecting YES or NO using the cursor key.

Most ESBTs also provide additional features for constructing end user interfaces using character text. These features are usually implemented with the high-level programming language provided by the ESBT. Some of these features are listed below.

- **Displaying Multiple-Choice Questions**

  To determine the value of an attribute value variable with possible several values (discussed in sec. 9.1), an ESBT may display a multiple-choice question, similar to what would appear on a paper form. During backward chaining, an ESBT displays the multiple-choice question when a variable with an as yet undetermined value is first encountered. Once the answer is selected, the variable receives the corresponding value. The value is retained, and the question need not be asked again.

- **Text Substitution**

  The substitution of more extensive text for rule variables and for values is important for phrasing questions in terms that are understandable to end users. Therefore, ESBTs support the ability to declare text substitutions in an expert system program. When the expert system is working on a problem, text substitutions are displayed if the associated variables and values must be shown to the end user. Some ESBTs may limit the length of the text that can be substituted or impose other restrictions on this capability.

- **Text Placement**

  The ability to specify the position where text appears on the terminal screen is important for displaying information in a clear, understandable way. Some ESBTs also permit drawing borders around blocks of text using special characters. This provides a more finished, professional look.

- **Explanation of Terms and Concepts**

  Sometimes, the expert system may display a question that contains a term or phrase the end user does not understand. To provide further information, some ESBTs support display of additional explanatory text that can be made to appear by depressing a function key. After the additional text has been read, the end user may depress another function key to redisplay the question and provide an answer. (Sec. 3.3 contains an example of explanation capabilities.)

- **Permitting End Users to Answer "Unknown"**

  Sometimes, the expert system may ask a question for which the end user does not know the answer. To handle this situation, the ESBT may provide a "built-in" feature that permits the end users to answer "unknown." The ESBT inference engine must also provide a method to account for "unknown" facts during inference. Possible ways of handling "unknown" responses by the end user are:

  (1) Causing rules with "unknown" conditions in its IF part to fail.

  (2) Ignoring the "unknown" conditions and allowing the rule to be evaluated on the basis of remaining conditions.

  Some ESBTs support only one of these options. Others permit developers to select whether or not "unknown" responses are to be permitted and, if so, to specify an alternative method for handling them. A problem may arise if the application has questions that cannot be answered "unknown." In this case, the ESBT must allow the "unknown" response option to be "turned off."

- **Retraction of Answers by the End User**

  Another helpful feature supported by many ESBTs is the ability to "undo" or retract an answer given by the end user. Sometimes, this feature is implemented using a function key. When the key is depressed, the user is given an opportunity to select the answer to be changed and then provide an alternative response.

Simple end user interfaces are adequate for many microcomputer expert system applications and are widely supported.

10.2 Constructing Advanced End User Interfaces

Advanced end user interface facilities can be used to create extensive systems of windows and menus. To support these capabilities, ESBTs sometimes use bit-mapped graphics software. In such cases, the ESBT may utilize a commercially available graphics package as the basis for a graphics programming subsystem.

The end user interface construction features described below permit creation of more sophisticated interfaces than are possible with simple text-based systems. With these features, interfaces can be constructed that help an end user to use the expert system in a systematic fashion. The objective is to make the expert system interface as clear and easy to use as possible.

10.2.1 Advanced Features Currently Available

Several features that may be supported by ESBTs are discussed below.

- **Window Design Facilities**

  A window is a rectangular structure that appears on a computer terminal screen. Windows can be of any size and have elaborate borders and title lines. Windows can contain text or graphics inside them.

- **Window Configuration**

  An important capability associated with using windows is the ability to subdivide a window into separate areas containing different information. For instance, a window may be divided into three parts: a top left-hand part that contains text describing the diagnosis of a malfunctioning factory machine, a top right-hand part consisting of a menu of actions the user can select, and a bottom part showing a diagram of the machine with the location of the problem.

- **Supporting End User Navigation Between Windows**

  In an expert system, it is often necessary to be able to erase one window and display another. For instance, the end user may make a selection from a menu in one window that represents a request for information about a particular topic. To satisfy the request, the menu and the window it is in are erased. A different window is then displayed that contains the desired information.

- **Definition of Menus**

  Menus are a common device for effecting communication between the expert system and the end user. The ability to define customized menus that suit specific needs is important in end user interface design.

- **Support for Use of a Mouse**

  Today, an increasing number of microcomputer systems are incorporating use of a mouse into their interfaces. Mouse facilities permit direct selection of items from menus. They can also be used to navigate between different windows.

- **Additional Facilities for End User Responses**

  This includes an assortment of specialized, menu-like facilities that permit the expert system to display questions and end users to select responses. For example, ESBTs may

display a question with a list of possible responses, each with an adjacent box next to it. The user may select one or more responses by typing a character in the adjacent box and thus "checking" the box. Boxes can also be "checked" using a mouse. The terminology describing these facilities varies.

- **Computer Fonts and Icons**

  Advanced user interface facilities may include the ability to select different font styles and sizes. This is useful in customizing interfaces to suit a particular style and to distinguish important pieces of text. Icons are graphic symbols provided by the system that can be used to represent important objects or concepts.

- **Bit-Mapped Graphics for Creating and Displaying Diagrams**

  Bit-mapped graphics provide the ability to create pictures, drawings, and graphs that can appear in windows. This is important for illustrating complex diagrams, flow charts, and processes to end users.

Not all the features described above are supported to the same extent. For instance, some ESBTs support creation of windows but may not support navigation between different windows. In other cases, complete window configuration facilities may not be provided. Also, the term "graphics" may be used to refer to different capabilities in different tools.

10.2.2 Using Advanced User Interface Construction Facilities

In general, sophisticated end user interfaces, especially those with extensive graphics, require more programming effort to implement than simple text interfaces. Often, the graphics programming system provided by the ESBT consists of a complete programming language that is complex and takes time to learn.

Some packages provide high-level programming facilities for window design that help reduce programming effort. Such facilities are sometimes referred to as "window painting" tools. A developer may use a "window painting" tool to draw--or "paint"--a window or menu directly on the terminal screen. The tool then internalizes the "painted" design and generates a graphics program that can later reproduce the window or menu. "Window painting" facilities, however, are not supported by all ESBTs.

In expert system projects that require the development of more extensive end user interfaces, it is sometimes advisable to assign an additional programmer to develop graphics interfaces. Effective graphics programming usually requires a significant amount of skill and experience.

10.3 Hypertext

Hypertext systems store large text documents and provide the ability to access, display, and navigate text documents in a manner that is flexible and natural. Hypertext systems permit the end user to retrieve a text document, display an interesting part, and read it in linear fashion, much as one would read a book. The user is also allowed to link to related parts of the document and read them, similar to the way a manual or encyclopedia might be read.

In a hypertext database, a text document is divided into sections of text with each section being assigned to a database node. Sections of text can be related topically and corresponding nodes provided with links. When viewing a hypertext document at a terminal, these links are indicated by highlighting terms that refer to the related topic. A mouse can be used to follow the links by selecting on appropriate highlighted terms. Typically, a hypertext system permits several pieces of text to be displayed simultaneously at a terminal using different windows (as described in sec. 10.2.1).

Hypertext can be very useful in an expert system for providing explanations. A hypertext interface allows the end user to select a term of interest and to link to the relevant portion of a text document where the term is described. If display of further information is desired, appropriate links can be followed to other parts of the document.

Increasingly, ESBTs are providing hypertext subsystems that can be integrated into expert system applications. The hypertext subsystems in some microcomputer-based ESBTs are not as sophisticated nor as extensive as those offered by many stand-alone commercial hypertext products. Interfaces to external hypertext systems can often be implemented via a CALL statement used to invoke external program modules. For further information on hypertext, see [CONK87].

10.4 Explanation Generation Facilities

A useful service provided by an expert system program is the ability to explain its actions. An example of the use of explanation capabilities was provided in section 3.4. For some applications, the ability to design and customize explanation facilities is important. This section summarizes different methods for generating explanations and presenting them to the end user.

- **Basic Explanation Generation Facilities**

  Explanation facilities that "play back" the sequence of inferences are provided by most backward chaining ESBTs and can easily be incorporated into the end user interface.

- **Use of Graphics Facilities**

  Graphics packages can be used to provide a powerful explanation capability. Often, the function of a real-world system can best be explained to the end user by displaying a window containing a diagram or drawing.

- **Use of Hypertext in Explanation**

  Perhaps one of the most important benefits provided by hypertext to expert system applications is explanation. If the user requires further explanation of a term, a hypertext window can be displayed that contains detailed information. The document containing the information about the term can then be navigated as described in section 10.3.

- **Facilities for Querying the Contents of the Knowledge Base**

  In some cases, an end user may want to know about the contents of an expert system's knowledge base. That is, the end user may wish to know what rules the expert system has about a particular topic. To obtain an answer, it is desirable to be able to query the knowledge base using a query facility designed for this purpose. At present, this type of interface is not widely supported by ESBTs but in some cases can be created by the developer. As expert systems become increasingly integrated into the organization's information resources, query interfaces may become more desirable.

ESBTs differ significantly in the explanation system facilities they provide and the ease with which these facilities can be implemented. Development of highly customized explanation capabilities often requires significant programming effort and use of a programming language or external software package.

10.5 Summary of End User Interface Construction Features

In table 10.1, characteristics of selected features relating to end user interface construction are summarized. Each feature is characterized by:

- The level of computer skill required to use the feature.
- The frequency of occurrence of the feature in microcomputer-based ESBTs.
- The computer resource requirements of the feature.

Categories for level of computer skill were described in section 7.7. Resource requirements are rated "Low," "Medium," and "High," also as described earlier in section 7.7.

Table 10.1. Summary of End User Interface Construction Features

Feature	Required Skill Level	Occurrence Of Feature	Computer Resource Requirements
Display Of Multiple Choice Questions	Computer Literate	Common	Low
Text Substitution And Placement	Computer Literate	Common	Low
Support For "Unknown" Answers	Computer Literate	Common	Low
Support For Retraction Of Answers	Computer Literate	Less Common	Low To Medium
Support For Text Explanation Screens	Computer Literate	Common	Low
Support For Explanation Facilities	Computer Literate	Common	Low To Medium
Support For Custom Menus	Computer Specialist	Recently Introduced	Low To Medium
Support For Window Design & Configuration	Computer Specialist	Recently Introduced	Medium
Support For Graphics Programming	Computer Specialist	Recently Introduced	High
Support For Mouse	Computer Specialist	Recently Introduced	Medium
Hypertext	Computer Specialist	Recently Introduced	Medium To High

11. SUPPORT FOR CONSTRUCTING INTERFACES TO OTHER SOFTWARE SYSTEMS

Chapter 3 contained a discussion of the different types of software interfaces that an expert system may have in the microcomputer environment. The focus of this chapter is on facilities provided by microcomputer-based ESBTs for creating interfaces to other software systems. ESBT support for application portability is also discussed.

Figure 11.1 shows some interfaces between an expert system and other software that may be developed using an ESBT.

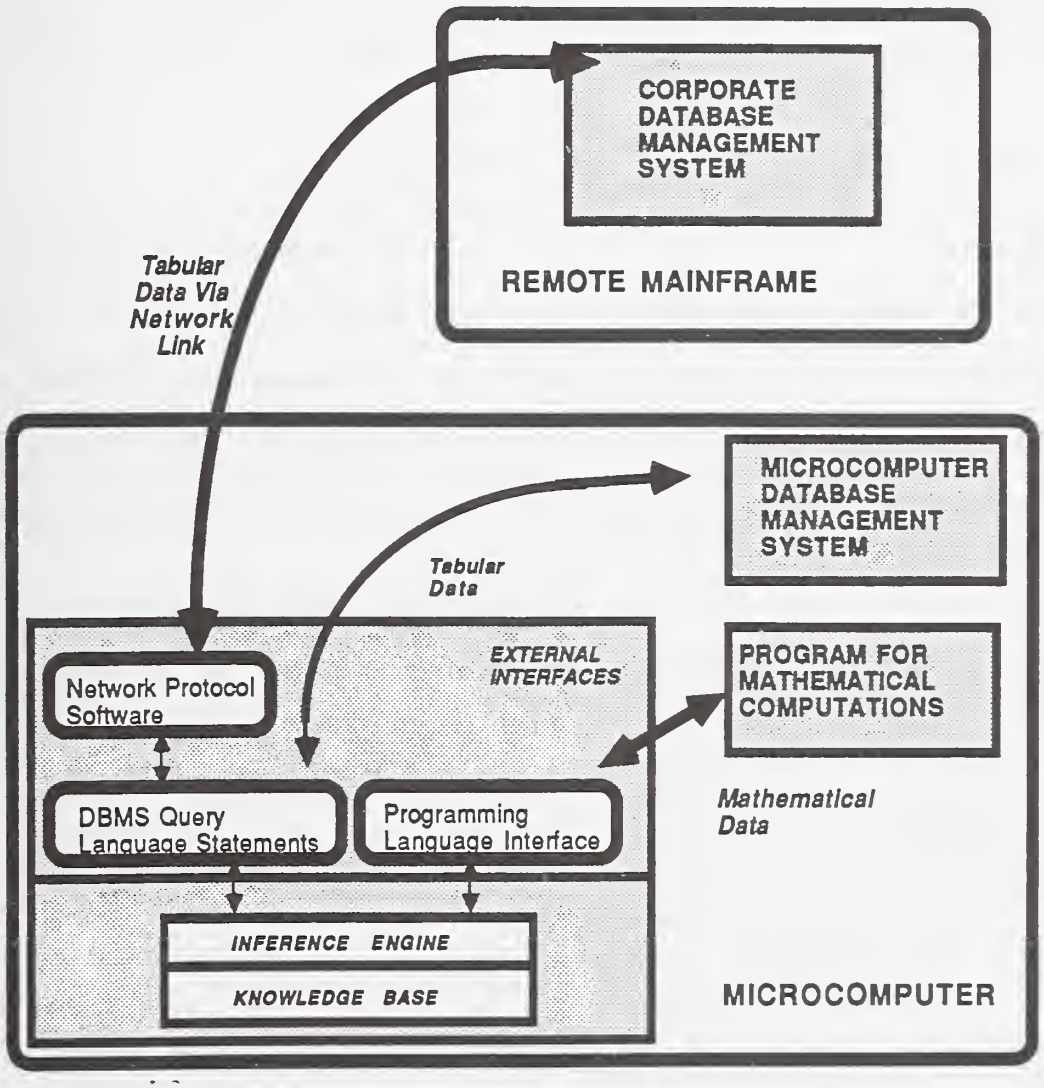

Figure 11.1. ESBT Bridges to Other Software Systems.

In general, microcomputer expert systems are somewhat more self-contained than expert systems running on larger platforms. The number of commercial DBMSs and other software products that an expert system can interface with in the microcomputer environment is somewhat limited. Capabilities for interfacing to software systems on other platforms are also more limited. However, as local area networks (LANs) become more prevalent in organizational environments, this will change.

11.1 Interfaces to Microcomputer DBMSs

Almost all microcomputer-based ESBTs support construction of interfaces to commercial microcomputer-based DBMSs. Sometimes, the ESBT provides its own DBMS. Transfer of data from a DBMS to an executing expert system depends on supporting several capabilities. These are discussed below.

- **Initiating Calls to an External DBMS**

  ESBTs may support the ability to issue ad hoc queries to a database. These statements can be embedded in the IF part or the THEN part of a rule. The query language used can be the SQL standard [ANSI89] or a proprietary language developed by the DBMS vendor. Increasingly, commercial DBMS vendors are providing SQL query languages. Correspondingly, ESBTs are being enhanced to support SQL interfaces. However, in the microcomputer environment, the SQL that is used is sometimes a subset of the complete language. (NIST provides an SQL test service for determining conformance of SQL implementations to the ANSI standard.)

- **Operations Supported by the ESBT's Query Language**

  An important factor in evaluating the interface is the operations that can be performed in the query language. At a minimum, most interfaces support query statements with selection criteria. Such queries are usually directed against a single table, dataset, or file. For example, a simplified SQL query to determine the availability and location of a replacement part for a burned out generator might be:

```
IF   PROBLEM = generator_burn_out
THEN ACCESS INVENTORY_DBMS
     QUERY = SELECT Part_Availability, Location
             FROM Part_Table
             WHERE Part_Name = generator
     DISPLAY Part_Availability, Location
```

  Many query language interfaces are limited in the number of selection conditions that can be specified. Also, not all

languages permit queries involving two or more tables; that is, they do not permit database "join" operations. Similarly, other operations normally supported in commercial DBMS products may not be provided by the ESBT.

- **Mapping Retrieved Data Into Expert System Data Structures**

  Once database values are retrieved, they must often be stored in the expert system program for reuse. This means that the database values must be transferred to program variables or other structures. DBMSs generally store information in records. If the ESBT supports representation of complex information and definition of generic structures (discussed in secs. 9.2 and 9.3), then database records can be mapped to expert system data structures. However, if the ESBT supports only attribute value variables (sec. 9.1), then transfer of large amounts of data might be somewhat more complicated.

- **Support for Updating the DBMS**

  The ESBT may also support update of database records including insertion, deletion, and modification of records. However, capabilities for updating external DBMSs may be limited.

- **Importation of Object Class Definitions From External Sources**

  A few ESBTs that support object-oriented programming (discussed in ch. 9) also support the ability to import class definitions stored in an external source, such as a DBMS. For expert systems that must use record or schema definitions created in an external application (or one being developed), this is a useful capability. Automatic importation of class definitions saves the developer the trouble of retyping definitions that already exist elsewhere.

While a direct interface from an expert system to a DBMS is desirable, it is not always possible. Though there are many DBMS products available, microcomputer-based ESBTs typically support bridges to only one or sometimes two DBMSs.

Instead of a direct interface, it is also possible for the expert system to call a programming language module which communicates with a DBMS. The module sends the query to the DBMS, receives the data, and forwards the data to the expert system program. To create this kind of interface, the ESBT must support a programming language interface.

11.2 Interfaces to Procedural Programming Languages

Most ESBTs support interfaces to external program modules implemented in conventional programming languages. The ESBT

provides the ability to call an external program module, pass values to the module, and receive values.

- **Initiation of Calls to Procedural Programs**

  Calls to external program modules are generally made in the same manner as in DBMS interfaces. The ESBT provides the ability to place a "call" statement in the THEN part of a rule. Similarly, statements for transferring values between the expert system program and the programming language module are also provided by the ESBT.

- **Calls to Different Programming Languages**

  The most common interfaces are to C, Pascal, or LISP. However, other programming language interfaces are also supported. As discussed in chapter 9, some sophisticated ESBTs provide a complete programming language that permits specification of any operation.

- **Limitations of Smaller Microcomputer Environments**

  A call to an external program module requires that the executable version of that module be placed in computer memory. Since the amount of memory in a smaller microcomputer can be limited, the program module itself must often be limited in size. Consequently, an application involving an expert system that communicates with a large procedural program (or programs) is better implemented on a microcomputer platform that has (or can be extended to have) more memory and a sufficiently powerful processor.

In addition to supporting interfaces to external program modules, ESBTs may also provide a programming language that allows program modules to be directly incorporated into the expert system application. (This capability was discussed in ch. 9.)

11.3 Interfaces to Other Microcomputer-Based Software

Microcomputer ESBTs may also provide facilities for constructing interfaces to other special-purpose software packages. These include spreadsheet packages, external graphics packages, and hypertext packages.

- **Spreadsheet Packages**

  Interfaces to spreadsheet packages are commonly supported among microcomputer-based ESBTs. These interfaces are particularly useful for expert system applications in accounting or other areas of business.

- **External Graphics and Hypertext Packages**

  Sometimes, expert systems applications have highly specific requirements that cannot be met by graphics or hypertext systems provided directly by the ESBT. For such applications, interfaces to external systems are necessary. Sometimes these interfaces must be made through a programming language module.

- **Special Statistical Graphics Packages**

  For some applications, it may be desirable to provide an interface to a statistical graphics package for specialized analysis. Such a package may provide the ability to graphically display statistical information about data.

Other special-purpose interfaces may be provided by ESBTs for development of expert systems in a specific domain. For instance, ESBTs for engineering expert systems may have interfaces to software packages that perform specialized analysis functions.

11.4 Accessing Remote Software Systems

At the writing of this report, network communications are not widely supported by microcomputer-based ESBTs. With the rising popularity of LANs and other distributed computing systems, this is changing.

- **Remote Access to Commercial DBMSs**

  Software components for remote access to commercial mainframe database servers are available with some ESBTs. In some cases, these interface components are proprietary. In other cases, third-party software products are used for remote access to minicomputer and mainframe DBMSs. Commercial ESBTs vary with respect to which commercial DBMS products they can access. Currently, a Remote Database Access (RDA) standard is under development by the International Organization for Standardization [ISO89]. In its initial phase, the standard specifies a remote database access protocol based on SQL. This development can be expected to influence trends in commercial ESBTs.

- **General Network Access Capabilities**

  Open Systems Architectures refer to computer systems consisting of modular, flexibly interchangeable software and hardware components that communicate across networks using standard protocols. At present, computer systems are being planned and implemented that are based on the Open System Architecture concept. In keeping with this trend, some ESBTs are being enhanced with standardized interfaces and support

for remote communications. These developments can be expected to impact the external interfaces that will be provided by microcomputer-based ESBTs in the near future.

- **Distributed Knowledge Bases and Distributed Expert Systems**

  In a distributed expert system, an individual expert system may have its knowledge base located on several computers. In other cases, several expert systems may communicate to solve a problem, each specializing in a particular task. Distributed expert systems may be important factors in future corporate computer environments. At present, development of distributed expert systems is not generally supported by commercial ESBTs and is still largely a research topic.

As support for remote communications capabilities increases among commercial ESBTs, it is expected that microcomputer-based expert systems will be developed that communicate with a variety of software systems on remote computer platforms. However, at present these capabilities are limited and sometimes difficult to use.

11.5 Support for Development of Embedded Expert Systems

Some ESBTs support the development of expert systems that are embedded in other software systems (discussed in sec. 3.6). Embedding an expert system means that an expert system program is integrated into a larger software system from which it can be invoked, receive arguments, and return values.

ESBTs that support a procedural programming language (discussed in sec. 9.5.2) provide one way of embedding expert system applications. By implementing the larger software system in the language provided by the ESBT, calls to the embedded expert system can be made in a straightforward manner. This solution is straightforward if the entire application can be implemented in the ESBT's language. Otherwise, the programming language provided by the ESBT must interface to the language in which the larger system is implemented.

Some ESBTs support development of compiled expert system programs which can be directly invoked from external software modules written in conventional programming languages such as C. In this case, the exact method of linkage may be specific to the ESBT being used.

Selecting an ESBT for an embedded expert system requires detailed understanding of each particular tool's method of support for developing embedded applications.

11.6 ESBTs and Application Portability

A number of ESBTs operate in multiple computer systems and support application portability across different hardware platforms and operating system environments. However, sometimes only part of an application can be ported. For instance, the inference engine and knowledge base may be portable, while other parts of the expert system program, such as the end user interface, are not.

- **Portability Across Hardware Platforms**

  The ESBT can be used to develop an expert system application on a mainframe computer that can later be deployed on a microcomputer, and visa versa.

- **Portability Across Operating Systems**

  ESBTs are also being developed that can be used in two or more commercial operating system environments.

- **Knowledge Base Interchange Formats**

  Currently a topic of research, knowledge base interchange formats are intended to facilitate transfer of knowledge bases between different knowledge representation systems. Thus, knowledge from one expert system could first be translated into the interchange format and later translated from the format into another expert system. In principle, a standard knowledge interchange format could allow transfer of knowledge bases between expert systems developed using different ESBTs.

As expert system technology becomes more widespread, application portability will correspondingly become more necessary and desirable. This will give an advantage to ESBT products that support portability across different hardware platforms and across different operating systems. In the future, ESBT support for translation of knowledge bases into standard knowledge interchange formats will also be desirable.

11.7 Summary of External Interface Construction Features

In table 11.1, selected features for supporting construction of external interfaces are summarized. Each feature is characterized by:

- The frequency of occurrence of the feature.
- The computer resource requirements of the feature.
- Whether or not the feature is especially useful for development of large applications.

Resource requirements are rated "Low," "Medium," and "High," as described earlier in section 7.7.

Table 11.1. Summary of External Interface Construction Features

Feature	Occurrence Of Feature	Computer Resource Requirements	Supports Large Application Development?
Interfaces To Commercial Microcomputer DBMS	Common	Low	Possibly
SQL Interface	Less Common	Low To Medium	Yes
Support For Microcomputer DBMS Update	Common	Low	Possibly
Interface To External Modules Written In Conventional Programming Language	Common For C Language. Less Common For Others	Low	Yes
Interface to Spreadsheet Package	Common	Low	No
Interface To External Hypertext System	Less Common	Low To Medium	Yes
Interface To External Graphics Package	Less Common	Low To Medium	Yes
Support For Access To Remote Computers	Rare	Low To Medium	Yes
Support For Application Portability	Portability Between Micro And Mainframe Becoming More Common	Low	Possibly

12. SELECTING ESBTs FOR USE IN AN ORGANIZATION

Previous chapters of this report covered applications of expert systems in the microcomputer environment (discussed in chs. 2-4) and expert systems development (ch. 5). A detailed description of the features of microcomputer-based ESBTs was provided in chapters 6-11. This chapter is intended to provide guidance on how to select ESBTs for development of applications.

12.1 Factors in Selecting an ESBT

Perhaps the most important consideration in selecting an ESBT is the characteristics of the application (or applications) the tool will be used to develop. Also important are the capabilities and skills of the developers who will use the tool. During the selection process, the characteristics of both application and developer are compared to capabilities of different tools in an effort to find an appropriate match.

Today, many commercial ESBTs are available for the microcomputer environment. These tools are complex, and as is indicated by the previous five chapters, have many features. It is therefore helpful to impose some structure on the process of matching characteristics of applications and developers to those of candidate tools. One way to structure the analysis is to divide it on the basis of the major areas of the ESBT architecture outlined in chapter 6 and discussed in chapters 7-11. These areas are discussed below.

- **Representing Knowledge About the Problem**

  It must be possible for the ESBT to represent knowledge about the problem and the way it is solved. In some cases, only production rules are needed. For complex applications, pattern matching, frame systems, or object-oriented programming may also be necessary. Knowledge representation in microcomputer-based ESBTs was discussed in chapter 9.

- **Selecting an Appropriate Inference Strategy**

  Determining whether backward chaining, forward chaining, or mixed inference is required is equally important. Inference strategies supported by ESBTs were described in chapter 8. Selecting a strategy to use was discussed in section 8.6.

- **Determining Characteristics of End Users**

  For expert system applications with human end users, the requirements (and preferences) of end users are important factors. The developer must determine what kinds of computer interfaces are needed and select an ESBT that can provide

them. For example, if end users are accustomed to seeing diagrams, it may be appropriate to select an ESBT that supports graphics capabilities. End user interface construction facilities were discussed in chapter 10.

- **Determining Requirements for External Interfaces**

  If the application must interface with other software systems, the ESBT must provide appropriate interfaces. External interfaces were discussed in chapter 11.

- **Selecting the Developer Interface**

  The development interface supported by the ESBT must be appropriate for the needs and skills of the developer. Important considerations include ease of use and ease of learning. The ability to support application prototyping (discussed in ch. 5) is particularly important. Features of developer interfaces were covered in chapter 7.

This organization is suggested as a starting point. The actual analysis may vary in specific situations, depending on the characteristics and requirements of applications, developers, and the organizations they belong to.

In addition, the analysis must take into consideration other factors. A particularly important consideration is the microcomputer platform the application will be developed on and the platform the application will be delivered on. (They may be the same or different.) The ESBT must be able to run on the platform intended for development and must also support delivery of applications on the intended platform.

The analysis must also consider the level of performance provided by the ESBT. Performance depends on both the efficiency with which the ESBT was implemented and the power of the microcomputer platform upon which the application will run.

12.2 Selection Steps

The actual process of selection might be described in a sequence of steps, summarized below.

- Determine the characteristics of the expert system application or applications. Characteristics of applications suitable for implementation using microcomputer-based ESBTs were discussed in chapter 4. The organization suggested in section 12.1 may serve as a basis for this analysis.

- Identify the characteristics of the developers, particularly in regard to level of software development skill.

- Assess the hardware platform (or platforms) that will be used for application development and for delivery of the completed system. This analysis may be limited to hardware already available in the organization or involve purchase of new computers. Relevant issues include performance, ease of use, product reliability, and copyright issues. A particularly important consideration is the operating system (or systems) that run on the computer. The ESBT and the operating system must be compatible.[9]

- Create a list of selection criteria based on knowledge of application characteristics, developer characteristics, and hardware characteristics. The organization suggested in section 12.1 may be used.

- Specify the list to identify mandatory features and optional features that are desirable (as well as features that may sometimes be supported but are not be needed or wanted).

- Research individual products. Information about products can be obtained by:

  (1) Surveying available literature.

  (2) Attending computer conferences in which ESBT vendors display products.

  (3) Attending product seminars presented by vendors.

  (4) Contacting current users of ESBTs.

  (5) Direct examination and evaluation of commercial ESBTs.

  Direct examination and evaluation of ESBTs require access to and use of individual products for a period of time. If a sample of the target application has been developed, it can be used for comparative benchmarking.

- The final selection is based on an overall comparative rating of features.

Once the ESBT is selected, further information about the tool will be acquired after development begins. Often, prototyping brings to light previously unknown characteristics of the application, sometimes making it necessary to change tools. In some cases, it may be desirable to prototype using one ESBT and use a different

---

[9]Sometimes, it is better to select the hardware after selecting the ESBT (although this is not always possible for practical reasons).

tool for large-scale development and to deliver the completed system (discussed in secs. 5.1 and 5.2).

12.3 Level of Effort in the Selection

The level of effort involved in the selection process will vary depending on the size of the application and the budget of the particular project. It would not be cost-effective to allocate hundreds of staff hours to select an ESBT that costs less than $500. Similarly, it would not be cost-effective to make a hasty decision on a software package costing thousands of dollars.

If the requirements of the application can be satisfied by inexpensive tools, it may be cost-effective to purchase several tools and experiment with them.

12.4 Other Considerations in Selection

Besides the technical feature-by-feature evaluation, there are other considerations relevant to the selection process. These areas of concern are discussed briefly below.

- **Portability**

  Though no standards have been developed specifically for ESBTs, portability across platforms is an effective means of maintaining long-term, stable usage of the selected ESBT.

- **Vendor Support**

  Evaluation of vendor support should consider such factors as comprehensiveness of manuals, availability of training, "hot-line" support, and warranties for the software product, etc.

- **Reliability and Product Bugs**

  It is advisable to procure an ESBT that has been available for a period of time and has had many users. This helps ensure that as many bugs as possible will have been discovered and fixed. New versions of the ESBT should be downward compatible. This helps assure that the introduction of a new version of an ESBT product does not require extensive revision to operational applications.

- **Licensing Considerations**

  Questions about the cost of site licensing (the cost of obtaining a copy of the ESBT for a particular site) and runtime licensing (the cost of each runtime copy of a developed expert system application) may also be important.

During the selection process, there are many trade-offs. The desirability of different features must ultimately be weighed against the cost of the tool that supports them. Frequently no single ESBT supports all desired features. For instance, a particular tool may support a sophisticated development environment, but it may lack the inference strategy appropriate for the intended application. In such cases, a sacrifice may be necessary, or an alternate means of implementing a capability must be sought.

13. CONCLUSIONS AND FUTURE TRENDS

This chapter presents conclusions about the state of the art in microcomputer-based ESBTs. Issues relevant to the future development of microcomputer-based ESBTs are also discussed.

13.1 The State of the Art in Microcomputer ESBTs

The summary of the state of the art in microcomputer-based ESBTs focuses on the following areas: features and the capabilities supported, kinds of applications that can be developed, and skills needed to use tools.

- **Features of Microcomputer-Based ESBTs**

  As this report has stressed, it is difficult to characterize microcomputer-based ESBTs in terms of a stable set of features. It is true that a number of highly useful commercial microcomputer-based ESBTs are based on a core set of features: goal-directed backward chaining, production rules, and text-oriented end user interfaces with limited graphics capabilities. However, as discussed in chapter 9, more powerful features for reasoning and knowledge representation are being incorporated into microcomputer-based ESBTs. Advanced end user interface facilities and sophisticated developer environments are becoming available. Microcomputer-based ESBTs are increasing their support for interfaces to external software systems, especially DBMSs.

- **Developing Applications With Microcomputer-Based ESBTs**

  Whether or not a particular ESBT can be used to develop a specific expert system application depends on matching tool capabilities to the characteristics of the application (discussed in chs. 5, 6, and 12). For relatively small, simple applications, a number of different microcomputer ESBTs might be adequate. For more complex applications, it may be difficult to find a tool that fully satisfies all application requirements. The developer may be forced to use a more sophisticated ESBT that provides additional knowledge representation capabilities and object-oriented programming. In some cases, the developer may have to use more powerful (and expensive) microcomputer platforms with extended memory or switch to a larger computer environment.

- **Appropriate Applications for Expert Systems**

  In chapter 4, two broad kinds of expert system problems were presented and discussed: problems for which solutions are selected and problems for which solutions are constructed. Most often, expert systems developed with microcomputer-based

ESBTs are selection type systems of small to moderate size. Many of these systems can be implemented on smaller microcomputer platforms. Larger applications, particularly those of the construction variety, are reserved for more powerful (and expensive) platforms.

- **Skills Required for Using ESBTs**

  In general, development of all but small, simple applications requires some programming skill. Using an induction tool, it is possible for individuals with little or no programming background to develop an expert system. Most ESBTs, however, require at least some ability to program and debug. Development of end user interfaces and external interfaces often requires considerable programming skills.

In the future, it can be expected that ESBT support for development of more complex applications in the microcomputer environment will continue to be enhanced. The number of interfaces to external software systems will increase. ESBTs will also become easier to use as development environments become increasingly sophisticated.

13.2 Near-Term Evolution of Microcomputer-Based ESBTs

How will microcomputer-based ESBTs evolve in the next 5 years? This section provides a summary of trends that are likely to effect the development of microcomputer-based ESBTs in the short term.

- **Continued Migration of Features From Larger Platform to Microcomputers**

  As the processing power of microcomputers increases, features are being added that were previously associated only with medium-sized or large-scale tools running on larger platforms. These features are making microcomputer-based ESBTs more powerful development tools.

- **Addition of Features**

  As microcomputer-based ESBTs continue to evolve, new features will continue to be added. Over the near term, the following features will become more commonplace:

  (1) Forward chaining and mixed inference as primary inference strategies (in addition to support for backward chaining).

  (2) Pattern matching.

  (3) Induction subsystems.

(4) Use of meta-knowledge to control inference.

(5) Use of frame systems for knowledge representation.

(6) Object-oriented programming facilities.

(7) Advanced end user construction facilities (described in sec. 10.2).

(8) External interface construction facilities for interfacing with mainframe DBMSs and other software systems on remote computers.

(9) Hypertext subsystems.

Microcomputer-based ESBT development environments can be expected to provide:

(1) Increasingly sophisticated environments that support structured knowledge entry.

(2) Graphics facilities to represent the structure of the knowledge base and permit access to its components.

(3) Support for integration of graphics with other facilities of the development environment such as rule tracing.

Other features, such as fuzzy logic, may not be supported by as many commercially available ESBTs.

o **The Emergence of Problem Specific ESBTs**

Some ESBTs are being specialized to construct expert systems that solve particular kinds of problems. For instance, problem specific tools may contain knowledge base components and other facilities for developing specific kinds of selection type expert systems. Examples include tools for developing expert systems that process claims or perform "help desk" functions [FERR90]. Eventually, problem specific ESBTs may be developed for many of the types of expert systems discussed in chapter 4.

It can be expected that, by the mid-1990s, support for the capabilities discussed in this section will be fairly prevalent.

13.3 Other Possible Near-Term Additions

There are some features that, while generally not supported at the present time, would be very useful for microcomputer-based ESBTs. Four of these are listed below.

- **Configuration Management**

  Support for configuration management and control was discussed in section 7.5. Configuration management has been seen as being valuable in other software development tools, such as computer-aided software engineering (CASE) tools, and can be expected to have a beneficial impact on ESBTs as well.

- **Support for Remote Access**

  External interface facilities for systems operating on remote computers and support for integration into LANs were discussed in section 11.4. As distributed computing systems and LANs become more popular and as access to remote mainframe DBMSs becomes more necessary, these facilities will become necessary to support development of expert systems that can access an organization's information resources.

- **Knowledge Base Verification**

  Knowledge base verification, introduced in section 7.2, is a research topic with strong potential for future commercialization. If implemented, this feature would assist in ensuring the consistency and completeness of knowledge bases under development.

- **Comprehensive Knowledge Management**

  A desirable addition to future ESBTs will be comprehensive facilities to support long-term knowledge management and maintenance (discussed in sec. 7.6).

Another important consideration in the development of commercial ESBTs and expert system technology is standardization.

13.4 The Need for Standardization

Standardization is important in many areas of software technology. In expert systems, standardization will be necessary to support interoperability between individual expert systems, interfaces between expert systems and other software systems, and exchange of knowledge. At the writing of this report, standards are being discussed in the following areas.

- **Knowledge Base Interchange Formats**

  Standardized interchange formats, discussed in section 11.6, would facilitate application portability and permit transfer of knowledge bases between expert systems developed with different ESBTs.

- **Standards for External Interfaces**

  Specification of standard external interfaces would permit expert systems to more easily interoperate with external software systems.

- **Standardization of Terminology**

  Development of a standard terminology would serve to promote common understanding of various aspects of expert systems and ESBTs. Lack of such a terminology can make it difficult for potential developers to compare and evaluate features of different products. Standardization might help ease these difficulties.

The development of standards in these areas would aid in the continued acceptance of expert system technology in traditional data processing environments. ESBTs that support standards could be used to develop expert systems that are portable, interoperable, and more easily integrated into existing software and hardware environments.

13.5 Final Remarks

Expert system technology is beginning to stabilize. The growing acceptance of this technology in industrial data processing departments bears evidence of this. The process of stabilization is also reflected among ESBTs in the trend toward support for a recognized set of capabilities (discussed in sec. 13.2).

At the same time, more and more expert systems are being implemented using microcomputer-based ESBTs. This trend has been aided by the continually increasing computational power of microcomputer platforms. Despite these developments, microcomputer-based ESBTs still lack potentially useful capabilities. Developing these capabilities will require research in the coming years.

14. REFERENCES

[ALUR90] Aluri, R., and Riggs, D. E. Expert Systems in Libraries, Ablex Publishing Corporation, 1990.

[ANSI89] ANSI "American National Standard--Database Language SQL with Integrity Enhancement," No. X3.135-1989, American National Standards Institute, 1989.

[BARR81] Barr, A., and Fiegenbaum, E. A., eds. A Handbook of Artificial Intelligence, Vol. 1, William Kaufman, 1981.

[BECK90] Beckman, T. J. "Methods for Selecting Promising Expert System Applications," The Fifth Annual AI Systems in Government Conference, pp. 14-21, Bethesda, MD, May 1990.

[BETZ85] Betz, D. "An Xlisp Tutorial," BYTE, Vol. 10, No. 3, pp. 221-236, March 1985.

[BIEL88] Bielawski, L., and Lewand, R. Expert Systems Development, Building PC-Based Applications, QED Information Sciences, 1988.

[BOLE90] Boley, H. "Expert System Shells: Very-High-Level Languages for Artificial Intelligence," Expert Systems, Vol. 7, No. 1, pp. 2-8, February 1990.

[BRAT86] Bratko, I. PROLOG Programming for Artificial Intelligence, Addison-Wesley, 1986.

[BROD89] Brody, A. "Product Comparison, the Experts," Infoworld, June 19, 1989, pp. 59-75.

[BROW86] Brownston, L. et al. Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming, Addison-Wesley, 1986.

[BUTL88] Butler, C. W., Hodil, E. D., and Richardson, G. L. "Building Knowledge-Based Systems with Procedural Languages," IEEE Expert, Vol. 3, No. 2, pp. 47-58, Summer, 1988.

[CARN50] Carnap, R. Logical Foundations of Probability, University of Chicago Press, 1950.

[CHAR87] Charniak, E., Riesback, C. K., and McDermott, D. V. Artificial Intelligence Programming, Lawrence Erlbaum Associates, 1987.

[CHEN76] Chen, P. "The Entity-Relationship Model--Toward a Unified View of Data," ACM Transactions on Database Systems, Vol. 1, No. 1, pp. 9-36, March 1976.

[CLAN86] Clancy, W. J. "Heuristic Classification," in Knowledge-Based Problem Solving, J. Kowalik, ed. Prentice-Hall, 1986.

[COFF90] Coffee, P. "AI Tools for PCs Tackle Application Development," PC Week, February 26, 1990, pp. 86-89.

[CONK87] Conklin, J. "Hypertext: An Introduction and Survey," Computer, Vol. 20, No. 9, pp. 17-41, September 1987.

[COLM82] Colmerauer, A. "Prolog and Infinite Trees," in Logic Programming, K. L. Clark and S. A. Tarnlund, eds. Academic Press, 1982.

[COYN90] Coyne, R. D. et al. Knowledge-Based Design Systems, Addison-Wesley, 1990.

[CUPE88] Cupello, J. M., and Mishelevich, D. J. "Managing Prototype Knowledge/Expert System Projects," Communications of the ACM, Vol. 31, No. 5, pp. 534-541, May 1988.

[DABR88] Dabrowski, C. E., and Jefferson, D. K. A Knowledge-Based System for Physical Database Design, NBS Special Publication 500-151, National Bureau of Standards, Gaithersburg, MD, February 1988.

[DABR90] Dabrowski, C. E., Fong, E. N., and Yang, D. Object Database Management Systems: Concepts and Features, NIST Special Publication 500-179, National Institute of Standards and Technology, Gaithersburg, MD, April 1990.

[DEMP67] Dempster, A. P. "Upper and Lower Probabilities Induced by a Multivalued Mapping," Annals of Mathematical Statistics, Vol. 38, pp. 325-339, 1967.

[DUDA79] Duda, R. O., Gaschnig, H., and Hart, P. "Model Design in the Prospector Consultant System for Mineral Exploration" in Expert Systems in the Micro-Electronic Age, edited by D. Michie, Edinburgh University Press, pp. 153-167, 1979.

[ENGE88] Engelmore, R., and Moran, T. Blackboard Systems, Addison-Wesley, 1988.

[FERR90] Ferranti, M. "Software Assists in Automation of Help Desks (AION Corporation Announces Path Builder Expert System Shell)," PC Week, Vol. 7, No. 40, pp. 33-35, October 8, 1990.

[FONG88] Fong, E. N., and Dabrowski, C. E. "An Expert System to Select Data Sources From Chemical Information Databases," ASME Computers in Engineering Conference, San Francisco, CA, August 1988.

[FORG77] Forgy, C. L., and McDermott, J. "OPS: A Domain-Independent Production System Language," Proceedings of the Fifth International Conference on Artificial Intelligence, Cambridge, MA, 1977.

[FORG82] Forgy, C. L. "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence, Vol. 19, No. 1, pp. 17-32, 1982.

[FOX90] Fox, M. S. "AI and Expert System Myths, Legends, and Facts," IEEE Expert, Vol. 5, No. 1, pp. 8-20, July 1990.

[GEVA87] Gevarter, W. B. "The Nature and Evaluation of Commercial Expert System Building Tools," Computer, Vol. 20, No. 5, pp. 24-41, 1987.

[GOLD83] Goldberg, A., and Robson, D. Smalltalk-80: The Language and Its Implementation, Addison-Wesley, 1983.

[HARM85] Harmon, P., and King, D. Expert Systems, John Wiley & Sons, 1985.

[HARM88] Harmon, P. et al. Expert Systems: Tools & Applications, John Wiley & Sons, 1988.

[HAYE83] Hayes-Roth, F., Waterman, D., and Lenat, D. eds. Building Expert Systems, Addison-Wesley, 1983.

[HENK88] Henkind, S. J., and Harrison, M. C. "An Analysis of Four Uncertainty Calculi," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 18, No. 5, pp. 700-714, 1988.

[HU89] Hu, D. C/C++ for Expert Systems, Management Information Source, 1989.

[ISO89] ISO "Information Processing Systems--Open Systems Interconnection--Remote Database Access--SQL Specialization," ISO Working Draft, Document ISO/JTC1/SC21/WG3 N844, June 2, 1989.

[KUNZ84] Kunz, J. C. et al. "Applications Development Using a Hybrid AI Development System," AI Magazine, Vol. 5, No. 3, 1984.

[LAUF90] Laufmann, S. C., DeVaney, D. M., and Whiting, M. A. "A Methodology for Evaluating Potential KBS Applications," IEEE Expert, Vol. 5, No. 6, December 1990.

[LIEB90] Liebowitz, J. "Expert Configuration Systems: A Survey and Lessons Learned," Expert Systems With Applications, Vol. 1, No. 2, pp. 183-187.

[MAIE85] Maiers, J., and Sherif, Y. S. "Applications of Fuzzy Set Theory," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-15, No. 1, pp. 175-189.

[MARC88] Marcellus, D. H. Expert Systems Programming in Prolog, Prentice-Hall, 1988.

[MART88] Martin, J. and Oxman, S. Building Expert Systems, A Tutorial, Prentice-Hall, 1988.

[MCDE82] McDermott, J. "R1: A Rule-based Configurer of Computer Systems," Artificial Intelligence, Vol. 19, No. 1, 1982.

[MCDE84] McDermott, J.; and Bachant, J. "R1 Revisited: Four Years in the Trenches," AI Magazine, Vol. 5, No. 3, pp. 21-32, Fall 1984.

[MERR89] Merritt, D. Building Expert Systems in Prolog, Springer-Verlag, 1989.

[MOCK90] Mockler, R. J. "Using knowledge-based systems for estimating risks inherent in a proposed KBS project," Expert Systems, Vol. 7, No. 1, pp. 27-41, February 1990.

[MURD90] Murdoch, H. "Choosing a problem -- when is Artificial Intelligence appropriate for the retail industry?," Expert Systems, Vol. 7, No. 1, pp. 42-49, February 1990.

[NII86] Nii, H. P. "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," AI Magazine, Vol. 7, No. 2, Summer 1986.

[NGUY87] Nguyen, T. A. et al. "Knowledge Base Verification," AI Magazine, Vol. 8, No. 2, Summer 1987.

[PARC89] ParcPlace Systems, Inc. The Objectworks for Smalltalk-80 Manual, 1989.

[PARS89] Parsaye, K., Chignell, M., Khoshafian, S., and Wong, H. Intelligent Databases: Object-Oriented and Deductive Hypermedia Technologies, John Wiley & Sons, 1989.

[POTT90] Potter, W. D. et al. "SLING: A Knowledge-Based Product Selector," Expert Systems With Applications, Vol. 1, No. 2, pp. 161-169.

[PRER89] Prerau, D. S. "Choosing an Expert System Domain," in Topics in Expert System Design: Methodologies and Tools, G. Guida and C. Tasso, eds. North-Holland-Amsterdam, 1989.

[QUIN79] Quinlan, J. R. "Discovering Rules by Induction From Large Collections of Examples," in Expert Systems in the Microelectronic Age, D. Michie, ed. University of Edinburgh Press, 1979.

[RADA90] Rada, R. "An Expert System for Journal Selection," IEEE Expert, Vol. 5, No. 2, April 1990.

[RICH91] Rich, E. and Knight, K. Artificial Intelligence (Second Edition), McGraw-Hill, 1991.

[ROBI65] Robinson, J. A. "A Machine-Oriented Logic Based on the Resolution Principle," Journal of the ACM, Vol. 12, pp. 23-41, January 1965.

[ROBI85] Robinson, V. B., and Jackson, M. "Expert Systems for Map Design," Proceedings of AutoCarto-7: Digital Representations of Spatial Knowledge, pp. 430-439, Washington, DC, March 1985.

[ROTH90] Roth, J. "NATIONAL DISPATCHER ROUTER: A Multi-Paradigm Based Scheduling Advisor," Second Annual Conference on Innovative Applications of Artificial Intelligence, Washington, DC, May 1990.

[SAWY86] Sawyer, B., and Foster, D. Programming Expert Systems in Modula-2, John Wiley & Sons, 1986.

[SCHI87] Schildt, H. Artificial Intelligence Using C, Osborne/McGraw-Hill, 1987.

[SCHW91] Schwartz, T. J. "Fuzzy Tools for Expert Systems," AI Expert, Vol. 6, No. 2, pp. 34-41, February, 1991.

[SHAF76] Shafer, G. A Mathematical Theory of Evidence, Princeton University Press, 1976.

[SHOR76] Shortliffe, E. H. MYCIN: Computer-based Medical Consultations, Elsevier, New York, 1976.

[SHOR85] Shortliffe, E. H., and Buchanon, B. G. "A Model of Inexact Reasoning in Medicine," in Rule-based Expert Systems, Addison-Wesley, pp. 233-262, 1985.

[STAC87] Stackhowitz, R. A. et al. "Validation of Knowledge-Based Systems," Proceedings of the Second AIAA/NASA/USAF Symposium on Automation, Robotics, and Advanced Computing for the National Space Program, Arlington, VA, March 9-11, 1987.

[STER86] Sterling, L., and Shapiro, E. The Art of Prolog: Advanced Programming Techniques, The MIT Press, 1986.

[TSUD90] Tsudik, G., and Summers, R. "AudES--an Expert System for Security Auditing," Second Annual Conference on Innovative Applications of Artificial Intelligence, Washington, DC, May 1990.

[VEDD89] Vedder, R. G. "PC-Based Expert Systems Shells: Some Desirable and Less Desirable Characteristics," Expert Systems, Vol. 6, No. 1, pp. 28-42, February 1989.

[WATE83] Waterman, D. A. A Guide to Expert Systems, Addison-Wesley, 1983.

[WEIN90] Weinman, E. "PC-based AI Environments," Systems AI, May 15, 1990, pp. 1-3.

[WILE84] Wilensky, R. LISPcraft, W. W. Norton, 1984.

[WINS84] Winston, P. H. Artificial Intelligence, Addison-Wesley, 1984.

[WINS89] Winston, P. H., and Horn, B. H. LISP (third edition), Addison-Wesley, 1989.

[YAGE84] Yager, R. R. "Approximate reasoning as a basis for rule-based systems," IEEE Transactions on Systems, Man, and Cybernetics, SMC-14, pp. 636-643, July/August 1984.

[ZADE65] Zadeh, L. "Fuzzy Sets," Information and Control, Vol. 8, pp. 338-353, 1965.

INDEX

☆ U.S. GOVERNMENT PRINTING OFFICE: 1991 — 281-557 40650

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NIST/SP-500/188

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
July 1991

4. TITLE AND SUBTITLE

Guide to Expert System Building Tools for Microcomputers

5. AUTHOR(S)

Christopher E. Dabrowski Elizabeth N. Fong

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED
Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

Same as item #6

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

Microcomputer-based expert system building tools (microcomputer-based ESBTs), sometimes known as expert system shells, are software packages for development of expert systems that run on microcomputers. This report provides system managers, planners, and potential expert system developers with a readable description of ESBTs for microcomputers including a detailed description of specific tool features and the capabilities they support. The technical content of this report is based on analysis of commercially available ESBTs. However, individual commercial products are not described, compared, or ranked.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

artificial intelligence; expert system; expert system building tool; expert system shell; inference; knowledge base; knowledge-based system; knowledge engineering; knowledge representation; microcomputer; object-oriented programming; production rule

13. AVAILABILITY

X	UNLIMITED
	FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
X	ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
X	ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
147

15. PRICE

ELECTRONIC FORM

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents
Government Printing Office
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NIST Technical Publications

Periodical

Journal of Research of the National Institute of Standards and Technology—Reports NIST research and development in those disciplines of the physical and engineering sciences in which the Institute is active. These include physics, chemistry, engineering, mathematics, and computer sciences.
Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Institute's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Institute's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NIST, NIST annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NIST under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published bi-monthly for NIST by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW., Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Institute on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NIST under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NIST administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NIST research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.
Order the **above** *NIST publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*
Order the **following** *NIST publications—FIPS and NISTIRs—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NIST pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NIST Interagency Reports (NISTIR)—A special series of interim or final reports on work performed by NIST for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Institute of Standards and Technology
Gaithersburg, MD 20899