Embedding expert systems in semi-formal domains: Examining the boundaries of the knowledge base

Edgar Albrecht Whitley
London School of Economics and Political Science

Submitted in fulfilment of the requirements for the award of the degree of Doctor of Philosophy of the University of London.

April 1990.

ABSTRACT

This thesis examines the use of expert systems in semi-formal domains. The research identifies the main problems with semi-formal domains and proposes and evaluates a number of different solutions to them. The thesis considers the traditional approach to developing expert systems, which sees domains as being formal, and notes that it continuously faces problems that result from informal features of the problem domain. To circumvent these difficulties experience or other subjective qualities are often used but they are not supported by the traditional approach to design.

The thesis examines the formal approach and compares it with a semi-formal approach to designing expert systems which is heavily influenced by the socio-technical view of information systems. From this basis it examines a number of problems that limit the construction and use of knowledge bases in semi-formal domains. These limitations arise from the nature of the problem being tackled, in particular problems of natural language communication and tacit knowledge and also from the character of computer technology and the role it plays. The thesis explores the possible mismatch between a human user and the machine and models the various types of confusion that arise.

The thesis describes a number of practical solutions to overcome the problems identified. These solutions are implemented in an expert system shell (PESYS), developed as part of the research.

The resulting solutions, based on non-linear documents and other software tools that open up the reasoning of the system, support users of expert systems in examining the boundaries of the knowledge base to help them avoid and overcome any confusion that has arisen. In this way users are encouraged to use their own skills and experiences in conjunction with an expert system to successfully exploit this technology in semi-formal domains.

Fy annwyl rhieni

My dear parents

"*Gewiß ist es zu bedauern, daß unter diesen Umständen manche häufige und bemerkenswerte Art aus Mangel an Platz nicht genannt werden konnte und vieles, was sicherlich eine eingehendere Besprechung verdient hätte, nur mit knappen verallgemeinernden Worten sich andeuten ließ*".

"*It is certainly regrettable, that under the circumstances many frequent and notable species cannot be referred to due to lack of space and a great number of them, which would definitely have been worthy of further attention, can only be mentioned in general*".

Richard Heymons. Berlin, Oktober 1915.

Vorwort "Brehms Tierleben - Die Vielfüßler, Insekten und Spinnenkerfe" - Neubearbeitet von Richard Heymons unter Mitarbeit von Helene Heymons.

Δοξα τῶ fax το επιουσιον

ACKNOWLEDGEMENTS

This thesis would not have been possible without the encouragement and friendship of my supervisor, Dr. Georgios Doukidis. Georgios has given me so many opportunities in the years that I have known him that I cannot hope to ever repay him for what he has done for me. **In some small way, Georgios, I hope that this thesis will be able to say thank you for everything that you have done for me.** Special thanks are also due Georgios' fiancee Lina Vantziou, who acted as messenger between Georgios and myself.

I am also indebted to Dr. Tony Cornford who became my `local supervisor' when Georgios was required to go to Greece. Without his support and encouragement this thesis would still be no more than hazy ideas rather than a properly presented argument. I have also benefitted greatly from his views on my work and his friendship.

To my parents, who have given me so much love, I am dedicating my thesis. They, and the rest of my family, have taught me the most important lessons about life, love and friendship. My parents have made many sacrifices to enable me to do the work I enjoy so much. **Thank you.**

The Information Systems department at the London School of Economics and Political Science has been a wonderful place to work in - both as a student and, more recently, as a member of staff. The friendship, encouragement and discussions I have enjoyed with the lecturers, secretaries and students have shaped many of the ideas expressed in this thesis to such an extent that I could never mention all the individuals concerned.

Last, but by no means least, I am grateful for the love and companionship of all my friends who kept me happy and `sane' while I was researching my thesis. Their patience was particularly appreciated when I became `anti-social' in the final stages of preparing the thesis. Special thanks go to my mentor and friend, Scott Lucas who was the first PhD student I met at university and who has been a dear, close friend ever since.

The first two years of this research were supported by an SERC research studentship no. 87307777.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

Scholarly inquiry is concerned with understanding and explaining the world. The physical sciences such as chemistry, physics and biology attempt to explain the structure of the world, whilst the social sciences such as economics, sociology and anthropology attempt to explain social phenomena of the world. The field of inquiry that this thesis is concerned with relates to understanding the human mind, as research for its own sake but more importantly as a method of designing computer systems that perform useful tasks in a manner similar to a human being. These tasks may be performed in situations where no human expert is available or in areas where the environment is not amenable to human action. This thesis will examine the limitations on this form of technology and will try to devise and evaluate solutions that minimise the effects of these limitations so that the benefits of such programs can be made more widely available.

a. ARTIFICIAL INTELLIGENCE

Metaphors have been used extensively throughout the history of science to help understand and explain phenomena that have been discovered. For example, in trying to understand the human mind, metaphors have often been used that were based on the current technology of the time. Thus at the time when clockwork mechanisms were the dominant technology the mind was thought to be an intricate clockwork mechanism, carefully regulated using cogs and springs (Bolter 1984). Attempts were made to explain the behaviour of the mind in terms of this metaphor; forgetting, for example, could possibly be `explained' in terms of a spring loosing its tension. When the telephone became the dominant new technology it too was used as a metaphor by which the human mind could be understood (Searle 1984). Forgetting, in the telephone metaphor, could be described as a temporary connection between two points (two pieces of data) being dropped. Inevitably, therefore, when the digital computer was developed it too was used as a metaphor to understand the human mind. Turkle (1984) gives an interesting insight into the way that young children can be seen to alter their beliefs about artifacts being alive and having `intentions' in the light of their experiences with computer games.

The field of **Artificial Intelligence** arose to examine this particular area of study and can be broadly classified into two sections: the first is **strong artificial intelligence** which

believes that an "appropriately programmed computer is a mind" (Searle 1980, p. 417) whilst the second view, **weak artificial intelligence**, believes that the computer can be successfully used to help understand aspects of the mind and that there are many practical benefits to designing machines that act in an `intelligent' manner. Both approaches to artificial intelligence research benefit from the nature of computers since any theories that are devised can be tested using the computers themselves.

The thought that a computer might, in some way, be `intelligent' provokes strong reactions in many people. To some `intelligence' is the very thing that distinguishes us from `lower' creatures and therefore *cannot* exist in a machine. Thus the use of the word `artificial' as part of the title for the research area could be seen to imply forgery or deception. Others hold that there is no fundamental difference between the human brain and a digital computer and that with sufficient research this similarity can be successfully demonstrated.

In his seminal paper "Computing machinery and intelligence" (1950) the mathematician Alan Turing considered how we might recognise a machine that demonstrated `intelligence'. The question of what intelligence exactly is has perplexed philosophers for centuries and Turing proposed a practical test which could be used to determine if a computer had `intelligence'. If a person sitting at a teletype (which does not offer the advantage of any visual clues) could not distinguish between statements made by a computer and those made by a human then the computer would be deemed to have passed the test.

A number of authors have criticised the predictive ability of the **Turing Test** (Gunderson 1985, Weizenbaum 1983) since it is a rather open-ended mechanism for examining the claimed intelligence of a machine. Consider the case of a computer system which, it is claimed, can understand natural language. If the person sitting at the teletype enters "What is the time?" and the computer responds correctly but then fails to give a sensible answer to the question "Could you tell me the correct time please?" then the system would have been deemed to fail the Turing Test since it's responses could be distinguished from those made by a human. The problems with the Turing test arise when the topics of conversation become more advanced. How would a computer system which was asked questions about thermodynamics be judged? These questions only differ from the previous questions in the topic that they refer to. If the computer cannot respond appropriately to these questions should it be judged to be intelligent?

i. Tasks that require intelligence

Figure 1.1 shows the main areas of research in artificial intelligence. Of these areas, game playing and natural language understanding probably best demonstrate the factors that led to the development of expert systems and they will be described in more detail below. Other areas of artificial intelligence research, such as problem solving, vision systems and robotics, had less of an influence on the origins of expert systems although they made important contributions to artificial intelligence in general. They are described in more detail by Boden (1987) and McCorduck (1979).

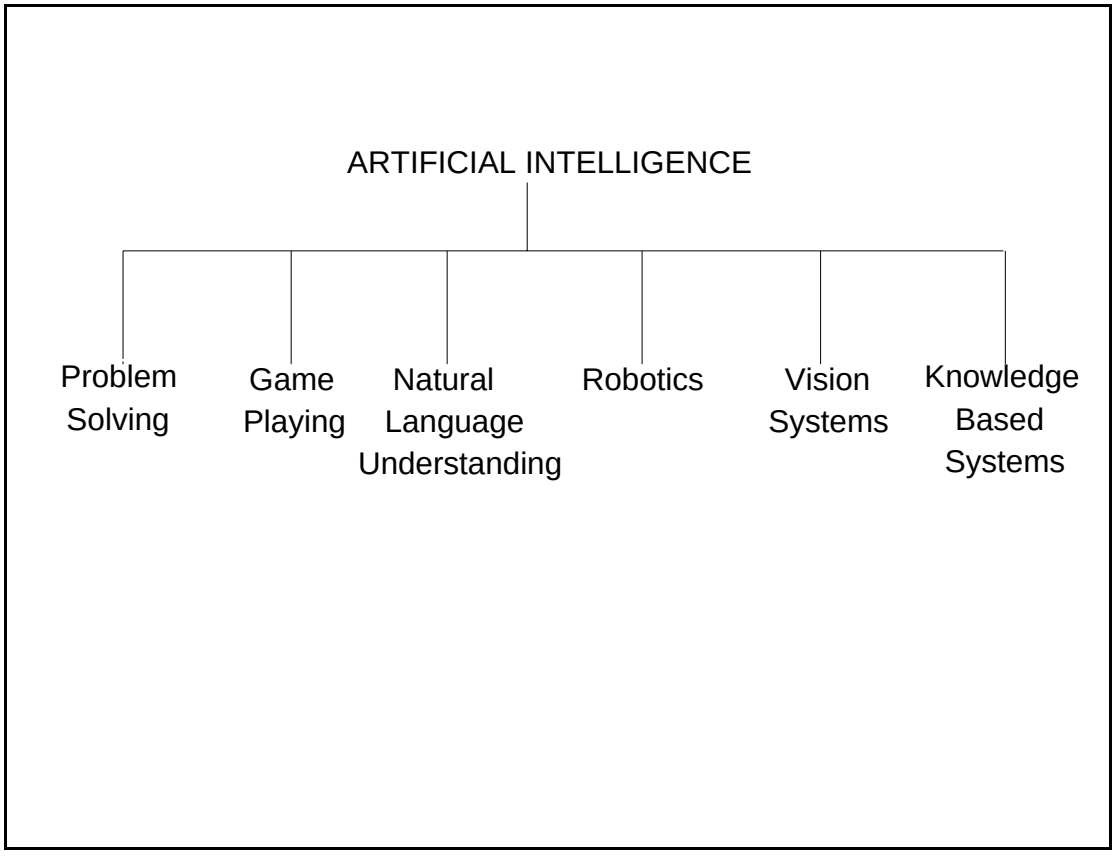


Figure 1.1 - The main areas of research in artificial intelligence

Game playing

One of the first areas of intelligent behaviour to be considered by AI researchers was the **playing of games**. If a computer could be programmed to play a game as well as a human then it would be reasonable to assume that the program exhibited some `intelligence', since intelligence is required to play games well. One of the earliest successful

implementations of computerised game playing was a computer program that was able to play checkers (draughts). Arthur Samuel, working for IBM, developed a program that could play checkers (Samuel 1983). As well as providing the program with a list of all the legal moves that could be taken, Samuel also gave the program various **heuristics** or **rules of thumb** which made suggestions as to which moves to take. In addition to these rules of thumb, the checkers playing program was also given a basic ability to `learn' from the games that it played. Each move that the program made was given a weight which was altered, depending on whether the computer won the game or not. Those moves which caused the computer to win were therefore more likely to be chosen again, whilst those that caused it to lose were less likely to be chosen. After the program had played a number of games and had built up considerable `experience' of the game, it actually beat Samuel on a number of occasions (McCorduck 1979), an experience which, he confesses, was rather unnerving.

Chess is another game which has been very popular with AI researchers. Michie and Johnston (1984, p. 21) report that there are 10^{120} possible games of chess. The number of possible games dwarfs the total number of seconds that the universe has been in existence and hence any computer program which tried to take into consideration all the possible games would never come round to making a first move. This limitation also applies to human chess players and it follows that human chess players do not consider all the possible games either. Following from Samuel's work most chess programs combine the searching of moves with rules of thumb which narrow down the number of possible moves to be considered. The `intelligence' of chess programs is therefore dependent on the quality of rules of thumb that can be incorporated into the program.

Whilst this method has been used in many successful programs that can play reasonable to good games of chess, there is evidence to suggest that the use of rules of thumb may not be a true representation of how human chess experts play. The world chess champion, Gary Kasparov, recently played the chess program DEEP THOUGHT, developed by Carnegie Mellon University (Ball 1989). The computer and human played two games, with Kasparov winning both easily. The current version of DEEP THOUGHT is capable of examining 760,000 positions per second in concert with its use of heuristics to guide the search strategy. Despite the large number of positions searched, the performance of the computer was far below that of the human expert although Kasparov conceded that the computer may be able to win in 1994 when DEEP THOUGHT II is unveiled. However it is expected that this program will be capable of examining 1 billion positions per second and

this suggests that the program would only be winning through the use of `brute force' rather than `intelligence'.

Natural language understanding

Another area of artificial intelligence research is **natural language understanding**. Natural language is normally used to refer to ordinary human languages "to distinguish them from constructed formal languages, such as the predicate calculus and FORTRAN" (Winograd and Flores 1986, p. 107). In addition to being a major area of research in artificial intelligence, successful natural language processing offers a number of practical advantages to the users of computer systems. Even the most modern and `user-friendly' computers are still difficult to use, requiring many cognitive and tactile adjustments to be made by their users. In almost all cases they must be used with a keyboard or possibly a desk based pointing device in conjunction with a formal command language. Natural language understanding systems therefore aim to assist the users of computers by allowing them to use less restricted forms of input.

One of the earliest programs to offer a reasonable level of language `understanding' was ELIZA, written by Professor Joseph Weizenbaum of the Massachusetts Institute of Technology (Weizenbaum 1976). The program was written to mimic the behaviour of a Rogerian psychiatrist. A Rogerian psychiatrist takes the responses made by patients and redirects them to the patients, making the patients talk through the problems that they have and allowing them to solve the problems for themselves.

Patient	: Well, my boyfriend made me come here.
ELIZA	: Your boyfriend made you come here?
...	
Patient	: Perhaps I could learn to get along with my mother.
ELIZA	: Tell me more about your family

Figure 1.2 - A sample dialogue with ELIZA (Weizenbaum 1976, pp. 3-4)

The sample dialogue with ELIZA, shown in **Figure 1.2**, illustrates this process in practice. It also demonstrates the method used by Weizenbaum to implement the system. When the user enters a line of text like "Well, my boyfriend made me come here" a pattern matching routine is used to convert this input into a related output line. In this case the

transformation is performed by dropping the "Well" from the beginning of the sentence, replacing the "my" with "your" and "me" with "you" and presenting the user with the transformed sentence. Other patterns that ELIZA searches for include topics such as mothers and fathers which cause the system to request information about the more general topic of the family. If the system is unable to find any pattern in the last sentence it will try and return to the previous topic of conversation or suggest a new topic of conversation.

The dialogue produced by ELIZA can be quite convincing, indeed Weizenbaum notes that his secretary once asked him to leave the office as she wanted to tell ELIZA about a personal problem she was facing. Weizenbaum was greatly disturbed when computer programmers, who knew the tricks that the program used, argued that ELIZA was a demonstration of a computer program which "understood" natural language (Weizenbaum 1983).

Other attempts to "understand" natural language have been more sophisticated with two of the most successful being Winograd's SHRDLU (Winograd 1972) and the work of Schank and his colleagues (Schank (1984) is a good introduction to this research). Winograd's program allowed users to "speak" to a computer which controlled a world made up of a table top, blocks, boxes and cubes of differing sizes and colours. It was possible to tell the program to pick up certain objects and place them in other positions and to ask the system questions about the blocks world or the past actions performed by the system. Because the system was only concerned with a very simple "micro-world" made up of blocks, it was able to resolve most ambiguities that arose by referring to the previous actions that had taken place within that world. Unfortunately both SHRDLU and the various programs devised by Schank and his colleagues were only able to operate successfully if they were restricted to very limited domains. Those attempts that were made to widen their field of applicability, to allow the systems to operate in areas other than "blocks worlds", came up against a major theoretical problem, namely how to represent common sense knowledge.

Common sense knowledge, for example, the knowledge that if an object is dropped it will fall, that this falling object will break on impact if it is fragile or will bounce if the surface is absorbent, has proved to be very difficult to formalise using existing techniques of knowledge representation.

At the same time as it was realised that common sense knowledge was becoming a major theoretical and practical problem some researchers, especially those working with Professor Edward Feigenbaum at the Heuristic Programming Project in Stanford, California (Feigenbaum and McCorduck 1983), began using the knowledge of human experts to create

computer based systems that operated at a similar level of competence to that of human experts. These programs were called **Expert Systems** since they attempted to embody the skills of a human expert.

b. EXPERT SYSTEMS

Expert systems, sometimes known as knowledge based systems, attempt to perform tasks that are usually undertaken by human experts in specialised areas of skill. They attempt to do these tasks in a similar way to human experts.

In the process of developing expert systems two important features were noted. Firstly the skills of the human expert could often be approximated using heuristics or rules of thumb in a similar manner to the rules of thumb used in the various game playing computer programs (Feigenbaum *et al.* 1988). Secondly, and more importantly, it was noted that the knowledge of the human expert was very often at a level far above the common sense knowledge that was causing problems elsewhere in the field (Harmon and King 1985). The expert would, for example, be concerned whether there was any power going through to the lights rather than how electricity was passed through wires. Since the problems of common sense knowledge were therefore effectively avoided it was possible to develop systems that gave acceptable levels of performance in certain specialised domains.

Doukidis and Whitley (1988) provide the following definition of an expert system:

> An expert system is a computer program that assists a user by providing information about a particular domain. It does this by manipulating information about the field that has been provided by a number of "experts" in the field. Another important feature of an expert system is that it has the facility to explain/justify the methods it used to provide the information. (p. 6)

This definition emphasises that expert systems are computer programs based on technology that is available at the current time. These computer programs attempt to solve problems by making use of `knowledge' that has been provided by various `experts' in the problem area. For example, an expert system might be developed to diagnose faults in electro-mechanical devices and to provide advice to human technicians, who may not have

had previous experience with this particular kind of device, in a similar way to human experts. The method used to provide this assistance is based on the use of a stored representation of `expert knowledge'. **Figure 1.3** provides a diagrammatic representation of the basic architecture of an expert system, highlighting the principal components used to perform this task.

The most important feature of the architecture of an expert system is the **knowledge base**, which is a stored representation of the expert knowledge. It is kept entirely separate from the **inference engine** which examines and uses it. This has three important implications. Firstly it means that the same inference engine can be used with widely differing knowledge bases - provided the knowledge has been coded in a standard format, the inference engine can manipulate knowledge about electro-mechanical devices, diseases or electronic components without having to rewrite the entire system. Secondly it means that a particular knowledge base could be used with different inference engines. For example, one inference engine may simply see what goals can be arrived at whilst another may actively try and pursue certain goals (Boley 1990). A third important implication of the separation of the knowledge from the way that it is used is that it is relatively easy to update and maintain the knowledge base. If the underlying knowledge changes, it is only necessary to change the knowledge base - the inference engine will remain the same, allowing rapid prototyping of the system and rapid adjustment to changing circumstances. The **working memory** is used as a temporary store for any data that is known to the system about a particular problem and the **external interfaces** are used for those applications which need to be linked to external sensors or specialist packages such as databases, simulation models or graphics packages. Finally, the **user interface** is used by the system to ask questions of the users and to obtain responses from them. It is also the means by which the system can provide an explanation / justification of its reasoning. A more detailed description of the architecture, use and operation of expert systems is given in Appendix I.

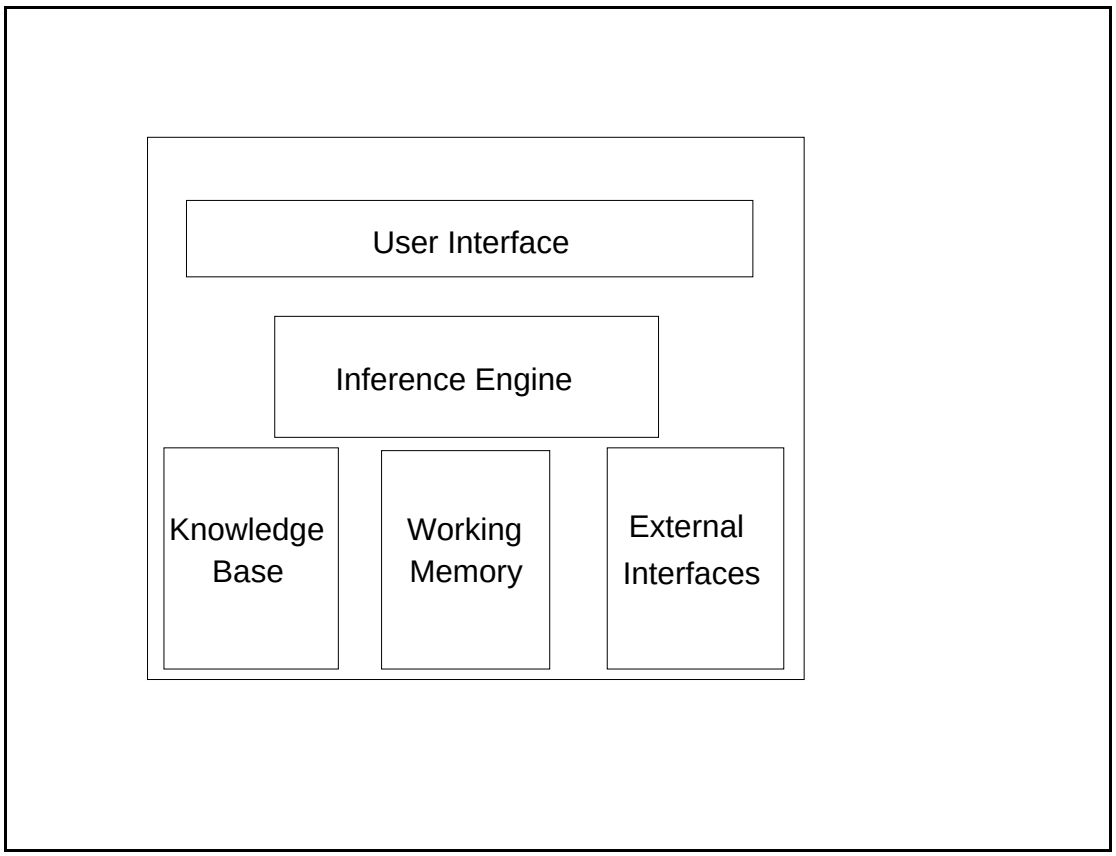


Figure 1.3 - The principle components of an expert system

i. The problem of knowledge representation and use

Practical expert systems depend critically on the representation and use of knowledge about the particular problem domain they are developed for. If some parts of the domain cannot be represented within the knowledge base or are not taken into consideration in other ways, or if the mechanisms for interrogating and using the stored knowledge do not offer the functionality required to solve problems in the domain, then the systems developed will be of little practical use.

One of the main areas of research in expert systems has been to devise suitable knowledge representation and manipulation techniques and many claims have been made about the effectiveness of the resulting systems:

> There is only one language suitable for representing information - whether declarative or procedural - and that is first order predicate logic. There is

only one intelligent way to process information and that is by applying deductive inference methods" (Kowalski 1980).

This quotation (which was made at a time when the logic based programming language PROLOG, developed in part by Kowalski, was first being introduced to the artificial intelligence community) is representative of many researchers who believe that suitably expressive knowledge base formalisms and inference methods have (already) been developed. Kowalski believes that first order logic is such a formalism, a view that is also held by Genesereth and Nilsson (1987) who propose the use of "first order predicate calculus as a language in which to represent the knowledge possessed by a reasoning agent about its world" (p. viii). They continue by imagining that "the agent exists in a world of objects, functions and relations that form the basis for a *model* of the agent's predicate calculus sentences" and propose that "deductive inference is the major reasoning technique employed by an intelligent agent" (p. ix). Similar statements have been made in support of fuzzy logic for knowledge representation: "[F]uzzy relations are a very flexible representation formalism that can be used to model any knowledge formalism" (Hall and Kandel 1988, p. 241) and similar claims have been made for most other artificial intelligence techniques used to represent knowledge.

Other researchers in the field do not make such extravagant claims, for example, Partridge (1987) in discussing the limitations of (current) expert systems technology focuses on the limited ability of current systems to cope with "context sensitivity" and the updating of knowledge bases. Although he believes that the problem of context-sensitivity will eventually be overcome, Partridge feels that current expert systems technology will tend to concentrate on those problem areas where "the necessary knowledge can be represented as a collection of more or less independent rules". In addition expert systems are particularly suitable for areas where "intelligent decision making can be implemented as a logical, truth-derivation mechanism" and where the "knowledge is fairly static" (pp. 2-3). Successful expert system applications have been developed in areas where these assumptions hold and those areas where the assumptions are particularly weak have few, if any, reported systems.

In common with many researchers in the field of expert systems, Partridge draws a distinction between **formal** domains and **informal** domains. Formal domains are made up of objects which are readily identified and exist within a regulated, clearly bounded area. Formal domains, therefore, satisfy the conditions set out by Partridge. In contrast, informal domains are made up of things that need to be interpreted and are based on experience and

subjective, personal identification rather than the `objective' methods used in formal domains. Informal domains do not satisfy Partridge's conditions and there are few, if any, potential expert system applications in informal domains.

The move towards integration

The distinction between formal and informal domains of application can be seen to become increasingly blurred when expert systems which are integrated with conventional data processing equipment are considered. The most common form of such integration is to link an expert system with a database.

The structure of a database is clearly defined and bounded and only certain operations can be performed on it. When an expert system is integrated with a database, however, the range of actions that can be performed by the two packages increases. The nature of the expert system means that tasks that could not be performed within the tightly controlled, formal structure of the database can be undertaken by the relatively less formal expert system.

An example of such a case is provided by Feigenbaum *et al.* (1988) when they describe an expert system developed by American Express to provide advice on whether to grant particular levels of credit to holders of their credit cards (p. 92-114). American Express cards have no preset spending limits and when an unusual purchase request is made, it was normally passed to a special credit department who examined a number of databases to obtain information about the customer concerned and made a decision as to whether to grant the credit. An expert system was developed to take over much of this task and the expert system automatically searches the necessary databases and comes up with a credit decision in a far shorter time than the previous system. In this application, the expert system can be considered to be the informal component in the overall system which enhances the capabilities of the purely formal database systems.

c. OVERVIEW OF THE RESEARCH IN THIS THESIS

i. Semi-formal domains

This thesis will use the concept of **semi-formal domains** as a means to overcome the distinction that is commonly made between formal domains and informal domains. Semi-formal domains combine formal and informal aspects within a single problem area and it is therefore not appropriate to speak of the domains being solely formal or solely informal.

One obvious example of such a combination of formal structure with informal aspects is legislation. Although the structure of legislation can be fairly complicated few, if any, decisions in cases are based simply on whether the appropriate steps of the legal process were applied correctly; most legal decisions are based on judgements of whether the terms of the legislation apply to the particular case. Once this "informal" process has been undertaken reaching a verdict is (relatively) straight forward. Thus Waterman *et al.* (1987) argue that "[T]he legal domain has the unique property of being semi-formalized, i.e. there exists a large body of formal rules that purport to define and regulate activity in the domain ... Concepts in the legal domain tend to be open textured, i.e. in general one cannot state necessary and sufficient conditions for applying legal predicates or have a program assess their applicability in arbitrary factual situations. ... Complex, ill-defined concepts (e.g. strict liability) are defined using new concepts that are just as ill-defined or vague (e.g. responsible, defective) ... This open textured nature of legal concepts makes it difficult to avoid subjective considerations when performing legal reasoning" (p. 26). This thesis will show that the property of being semi-formalised is not unique to legislation and that many domains show characteristics of being semi-formal.

The remainder of this thesis will examine ways in which problems can arise in semi-formal domains and will propose methods to minimise the effects of the problems. The extent to which these solutions overcome the problems will also be discussed. In doing so the thesis will suggest that most expert system applications are effectively semi-formal and that viewing them as being semi-formal will improve the design of such systems by taking into consideration factors that would be overlooked if the expert system was considered to be operating in a formal domain.

ii. Some of the problems with semi-formal domains

The problems with semi-formal domains, as they relate to expert systems, arise in two main areas. The first set of problems are a direct consequence of the knowledge that

needs to be stored in the knowledge base. In particular problems can arise with any knowledge that is expressed in `natural' language. It will be argued that the social processes involved in the use of language as a means of communication mean that much knowledge cannot be formally defined without losing important parts of that knowledge, since no formal basis ever existed for the language. A result of this is that it is possible for different people to use the same terms to mean completely different things. This has a practical implication for the design of expert systems since it means that the users of the system may form different interpretations of the knowledge base to those intended by the designers of the system.

Another kind of knowledge that is likely to cause problems in semi-formal domains is tacit knowledge. This is knowledge which "cannot be told" and therefore existing techniques are not able to represent it. Tacit knowledge is often used by human experts and it may be expected of the users of an expert system. Its nature, however, means that it cannot be represented within the knowledge base and this must be successfully conveyed to the users of the system. Furthermore, if the users do not possess such knowledge then the system must try and offer other ways for them to solve the difficulties that they are faced with.

Other problems of semi-formal domains arise from the nature of computers. Expert systems are implemented on computers and must participate in a process that involves informal social processes. Most computers, however, have very limited capabilities for participating in such processes and the consequences of this must be considered.

iii. The research method of the thesis

The research method followed in this thesis can be broken into two distinct stages. The first part is of a theoretical nature, whilst the second is practical, presenting solutions to the problems raised in the first part of the thesis.

The thesis begins by examining semi-formal domains, paying special attention to the implications of viewing domains in this way. In doing so a comparison is made with the conventional approach to expert system design which considers domains as being formal. Viewing domains as being semi-formal has many parallels to the socio-technical approach to designing information systems and a number of important ideas are taken from research in information systems.

The thesis continues by examining some examples of knowledge that cannot easily be represented using conventional knowledge representation techniques and discusses the extent to which these problems are a result of the domain being semi-formal. A proper understanding of these problems, which gives due consideration to the informal, social processes involved in the use of expert systems is necessary if effective solutions are to be developed.

Furthermore, problems arise from the fact that expert systems are implemented on computers. The ability of computers to become aware of, and deal with, problems of semi-formal domains as they arise, especially confusion on the part of users, is shown to be limited. The thesis then presents a theoretical model which can be used to deal with these problems.

The second part of the thesis describes a number of practical solutions that are developed to tackle some of the issues raised by the first part of the thesis and these are implemented on an expert system development tool which was developed as part of the research. The main features of the system, which was developed in a high level language, are described.

After the development tool has been described a number of different solutions to the problems raised are presented. As the solutions developed are evaluated a number of other features of semi-formal domains become prominent and they are incorporated in the later solutions.

The thesis ends by discussing the implications of the research on the design and development of expert systems in domains that can be considered to be semi-formal. A number of other issues that were raised by the research are also described and further possible areas for research, which would aim to tackle some of the remaining problems of semi-formal domains, are considered.

iv. Contributions of this research

The conventional approach to the design of expert systems considers the domain that the system operates on as being formally defined and as such concentrates on the technical efficiency of the underlying computer program. This thesis argues that the focus of this approach is inappropriate since it fails to take into consideration a number of informal factors that are present within any formal domain of application of expert systems and argues that

there are a number of examples of knowledge that cannot easily be represented using the conventional approach to designing expert systems.

Furthermore, many of the problems arise as a result of the social nature of interaction and consideration of the social environment must be a part of expert systems design. In particular, special consideration must be given to the fact that a computer takes part in the interaction. The limited capabilities of an expert system to, for example, notice that the users of the system are confused, has significant implications for the design and use of the resulting systems.

Although there has been some research into a few of the problems that arise from semi-formal domains, there has been very little practical work undertaken to solve these problems and this thesis contributes a number of software tools that are designed to tackle these problems.

The solutions that are described in the thesis raise a number of important issues that have not been adequately considered in the literature. The first of these is the distinction between `synthetic' domains, which are based on problem areas that are the result of purposeful action by a group of actors, and `organic' domains that arise as an indirect consequence of social interaction between actors.

Another important result of the research relates to how the users are to be supported when the features of the problem that appear most important to them differ from those considered to be important by the developers of the expert system. If these alternative interpretations are not taken into consideration then the usefulness of the resulting system will be limited to that of confirming the opinions of some users and being unacceptable to many others.

The question of interaction with a machine is also considered in detail and the thesis provides an important perspective on the notion of confusion. It offers practical solutions to overcome awareness of confusion that result from this perspective. The thesis also considers how the users can become aware that confusion has arisen and describes a domain independent model that can be used as a basis for improving the functionality and usefulness of expert systems.

Many of the ideas presented in the thesis can be seen to offer methods which open up the black box of expert systems, encouraging the users to examine the knowledge which is contained within them for themselves, so that they can use this knowledge in conjunction with their own particular skills to deal with those problem situations that arise.

The thesis also describes an expert system shell that was developed to support the solutions described and the shell incorporates a number of features which prove to be very useful for the development of expert system applications.

d. OVERVIEW OF THE THESIS

Chapter 2 describes in detail two approaches to developing expert systems. One approach takes a formal, functionalist view of expert systems development, whilst the second considers expert system as operating within semi-formal domains. This second approach benefits from work on socio-technical information systems and the most important elements of this work are described. The chapter ends by considering the social processes involved in communicating knowledge.

Chapter 3 gives a number of examples of knowledge that cannot easily be represented using conventional knowledge representation techniques. Many of these problems arise from the use of natural language communication in expert systems development. The use of tacit knowledge and tacit skills that may form an important part of the problem domain are also considered in this chapter.

Chapter 4 discusses the problems of semi-formal domains that arise from the fact that a computer system is involved in the interaction. It discusses the nature of confusion and shows that confusion cannot normally be known at the time that it occurs and that computers have very limited capabilities for becoming aware that the users of an application are confused. It then proposes a theoretical model of the actions of an expert system which can be used to determine the occasions when users can become aware that they are confused and describes the underlying conditions that have not been satisfied when such a situation arises.

Chapter 5 describes the expert system shell developed for the thesis and many of its advanced features are illustrated.

Chapter 6 documents the various solutions that were developed to overcome the problems of knowledge in semi-formal domains. In the course of testing these solutions a number of further problems were raised and these are discussed in the chapter.

Chapter 7 tackles the problems of a computer based knowledge base. It draws on the theory described in Chapter 4 to provide solutions to the problems highlighted.

Finally Chapter 8 summarises the work described in the thesis, describing how the various solutions provided allow the users of the expert system to examine the boundaries of

a knowledge base that is embedded in a semi-formal domain. It discusses the implications of the thesis for the design of expert systems in general and proposes further areas for research.

CHAPTER 2 - TWO APPROACHES TO DESIGNING EXPERT SYSTEMS

The discussion in the previous chapter has shown how the design and use of expert system applications is often based on a formal view of the domains of application for such systems. It was argued, however, that making the distinction between formal and informal domains has little relevance to many such applications and that it would be far more appropriate to consider these domains as being semi-formal, i.e. having elements that were formally defined as well as elements which were less formally defined. This chapter will summarise the two approaches to designing expert systems and highlight the differences in basic beliefs that distinguish the formal domain viewpoint from the semi-formal domain viewpoint.

a. FORMAL DOMAINS - A FUNCTIONALIST APPROACH

Many researchers in expert systems base their approach to developing applications on what Hirschheim and Klein (1989) label as the **functionalist paradigm**. This approach to designing information systems in general, and expert systems in particular, assumes that "there is one reality that is measurable and essentially the same for every one" (Hirschheim and Klein 1989, p. 1203) and this belief has a number of important implications for the design of expert systems.

The functional design of an expert systems attempts to model a domain using heuristics and rules of thumb, since expert systems are best suited for problems where no algorithmic solution exists. Although such methods are normally non-deterministic, the belief in a single measurable reality means that "it is often possible to codify true expertise in a narrow domain using surprisingly few heuristics" (Keller 1987, p.3).

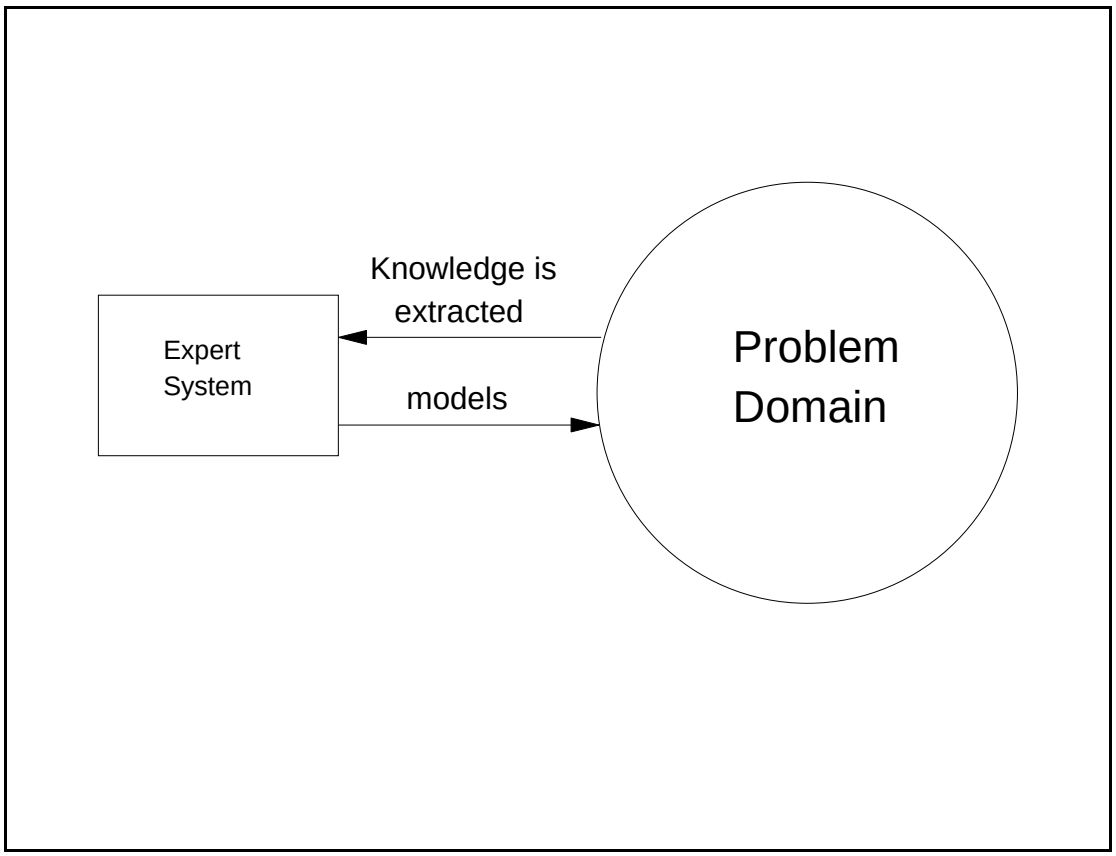


Figure 2.1 - The expert system kept separate from the domain it models

Since the domain being modelled by the expert system is "the same for everyone" the conventional approach to design has the expert system separated from the domain itself, with its knowledge base being considered as a heuristic model of the domain. This is shown diagrammatically in **Figure 2.1**.

i. Extracting `jewels' from the minds of experts

Considering the expert system to be distinct from the domain it is modelling leads directly to one of the most common indications of the formal view of expert systems development. This arises during the process of **knowledge elicitation** in the **knowledge acquisition** stage of expert systems development.

The term knowledge acquisition is normally used to refer to the process of obtaining knowledge from a source (normally a human expert) and transferring it into a form that can be used in an expert system. The first stage, of obtaining the knowledge from the knowledge

source, is referred to as knowledge elicitation. The knowledge that is elicited may be knowledge of facts, of relations or knowledge of strategies for tackling problems in the domain (Hart 1986).

By having the expert system "outside" the problem domain it is necessary to **extract** the knowledge from the domain and represent it in the expert system. The idea of extracting knowledge from the problem domain is extended through the use of metaphors based on the extraction of valuable mineral resources. Thus Feigenbaum and McCorduck (1983) speak of individual computer scientists working with individual experts "to explicate the experts' heuristics - to mine those jewels of knowledge out of their heads one by one" (Feigenbaum and McCorduck 1983, p. 80). Similarly Capper and Susskind (1988) speak of "extracting the gold nuggets from the mine of information that the expert possesses" (Capper and Susskind 1988, p. 31). Michie and Johnston (1984) "foresee a whole industry arising to tackle the job [of developing expert systems], based around a novel type of industrial plant, the `knowledge refinery', which would take in specialist knowledge in its existing form and debug it, pull it together, carry out creative gap-filling wherever the need becomes evident, and turn out knowledge that is precise, tested and certified correct" (p. 132). Even a report by the Council for Science and Society (1989) on the benefits and risks of expert systems considers knowledge to be an extractable commodity that can be used with expert system shells: "In the early days of research in expert systems it was predicted that such shells could be adapted to a wide range of problems, by doing little more than pouring in new facts and rules ... [It is now understood that] knowledge is not just poured into the mind; new data must be translated into mental representations, from which knowledge can be extracted and then integrated with existing skills or ideas" (Council for Science and Society 1989, pp. 10-11).

It could be argued that these metaphors are simply being used to describe the tasks involved in knowledge elicitation. However, the choice of metaphor can have a significant, if largely unconscious, effect on how the designers of such systems consider their domain. By describing knowledge in terms of `jewels' or `nuggets' support is given to the belief that the knowledge is "measurable" and has value irrespective of the context that it is used in. Indeed there is even a suggestion that `removing' the knowledge from the expert *increases* its value in the same way that refined gold is more valuable than gold deposits in rock. The metaphor also encourages the idea that it is possible to improve the quality (or value) of the knowledge by simply increasing its quantity.

ii. The `control methodology'

Another feature of the functionalist approach to designing expert systems is typified by a system described by Roth *et al.* (1987). This expert system is designed to diagnose faults in electromechanical devices and provide assistance to individual technicians who have no experience in trouble shooting this particular device. The system is designed so that "[T]he machine expert guides all problem solving activities dictating what observations and actions the user is to take to solve the problem" (p. 480). The system contains all the necessary knowledge about the domain and therefore assigns the user to the role of "data gatherer and action implementer" (p. 480).

This approach to designing expert systems has been described by Lipscombe (1989) as the **control methodology** and is a direct result of the belief in an observable, measurable reality that is "essentially the same for every one". Once the knowledge about the domain has been extracted, it is argued, the expert system has a better model of the domain than the users and therefore is more able to determine what actions should be taken to solve the problem. In most cases, however, the actions cannot be performed by the machine and the users are instructed to undertake them instead. These actions include obtaining data about the domain to assist in the decision making process and implementing any solutions that are determined by the machine. All the actions undertaken by the users of the system are carefully controlled by the system, based on its knowledge of the domain. "If the system believes that the user has a misconception that is detrimental to achieving his/her goals then ... the system must ... make an attempt to bring the user's knowledge into line with its own" (McCoy 1989, p. 163).

In controlling the actions of the users a distinction can be made between the expert system which lies "outside" the domain and the users who act within it. By lying outside the domain the system has `global' understanding of the domain and the problems that arise, an understanding which is not influenced by the particular features of the problem. It can use this information to control the actions of the users who do not have this knowledge. In particular many of the problems that occur are a result of incomplete or incorrect information on the part of the users -the users do not know enough about the domain. The system, which has better knowledge about the domain, is able to solve problems by supplementing the limited knowledge of the users.

The use of the `control methodology' to design expert systems also affects the **conceptual level** of the **user system interface** (Smithson and Hirschheim 1989). The

conceptual level is concerned with "the users' tasks, goals and knowledge, and, in particular, the way that they are interpreted and represented by both the user and the system" (Smithson and Hirschheim 1989, p. 9). The `control methodology' imposes a conceptual framework on the interaction which places the system in the dominant position with respect to the users. Regardless of its validity, this mode of interaction is likely to prove alien to most users of expert systems.

b. EXPERT SYSTEMS AND INFORMATION SYSTEMS

The functionalist approach to computer system design considers the computer system to be the central component that regulates, monitors and guides the flow of information within an organisation. This emphasis on technology has often been misplaced, with undue credence being given to the self proclaimed virtues of technology. In contrast **information systems** approaches the problem with a rather different focus, viewing the organisational information system as primarily a social system.

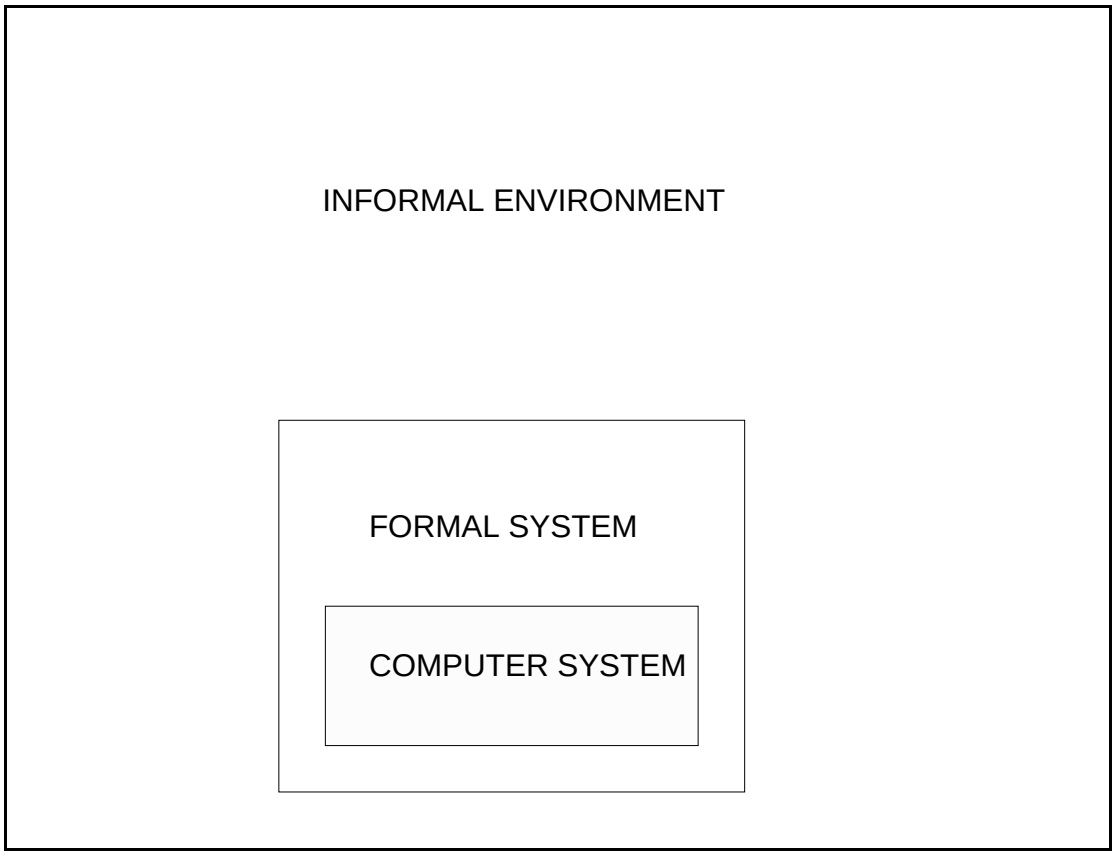


Figure 2.2 - The relationship between formal systems, informal systems and computer systems

Thus while computer science devises efficient routines for combining and using various symbols for a particular problem, information systems considers the effects that these `abstract' symbols have on social situations. For example, the symbols may result in the payment of a welfare benefit which has direct effects on individuals abilities to purchase goods and services.

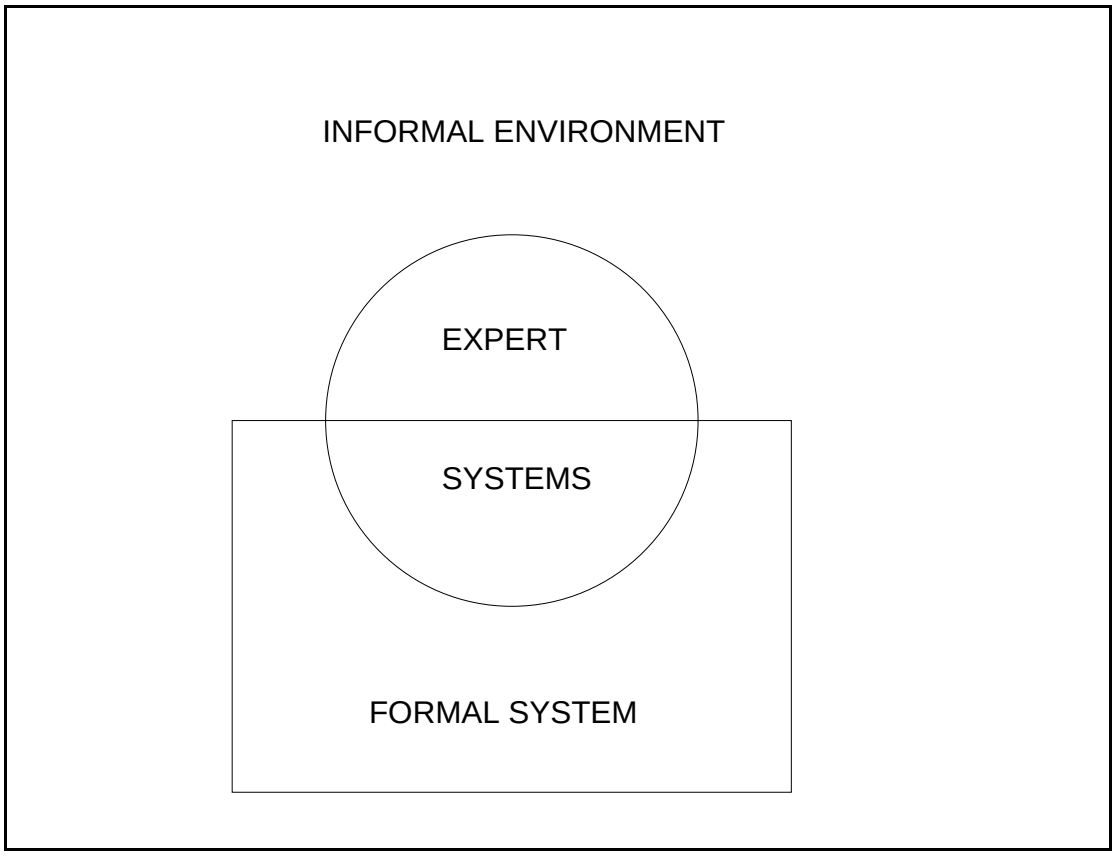


Figure 2.3 - Expert systems and their relationship with formal and informal systems

The focus of information systems research sees the computer system as one part of the organisation. The main day to day operation of the organisation is informally structured. Part of this informality is sufficiently regular or important to be formalised and computer based systems are suitable for automating some of the repetitive, complex or time critical components of the formal system (Backhouse and Liebenau 1990). It is necessary to design computer systems that perform such tasks as efficiently and effectively as possible, but it is important to re-emphasise that the computer system is only one component *within* the formal system which is itself part of the wider informal environment. **Figure 2.2** shows the relationship between these three components. The diagram also demonstrates the possibility of vastly increased complexity, leading to an incomprehensible electronic bureaucracy, that can result from using computers in areas where they are not suitable.

Applying this information systems approach to expert systems (which are simply a special kind of computer system) highlights the need to consider most expert systems as operating within semi-formal domains. Expert systems are best suited to tasks where there is no known algorithm (since more conventional software would be more appropriately used in such cases) but there must be some structure to the problem area. **Figure 2.3** illustrates the area of application for semi-formal domains, showing how the formal and informal parts of the problem domain are combined. Comparing this diagram with the previous one immediately suggests that a formal expert system may not be sufficient to overcome all the problems of semi-formal domains. The diagram also emphasises that the expert system is part of the semi-formal domain, that it is **embedded** *in* the domain rather than being distinct from it. This is shown diagrammatically in **Figure 2.4**.

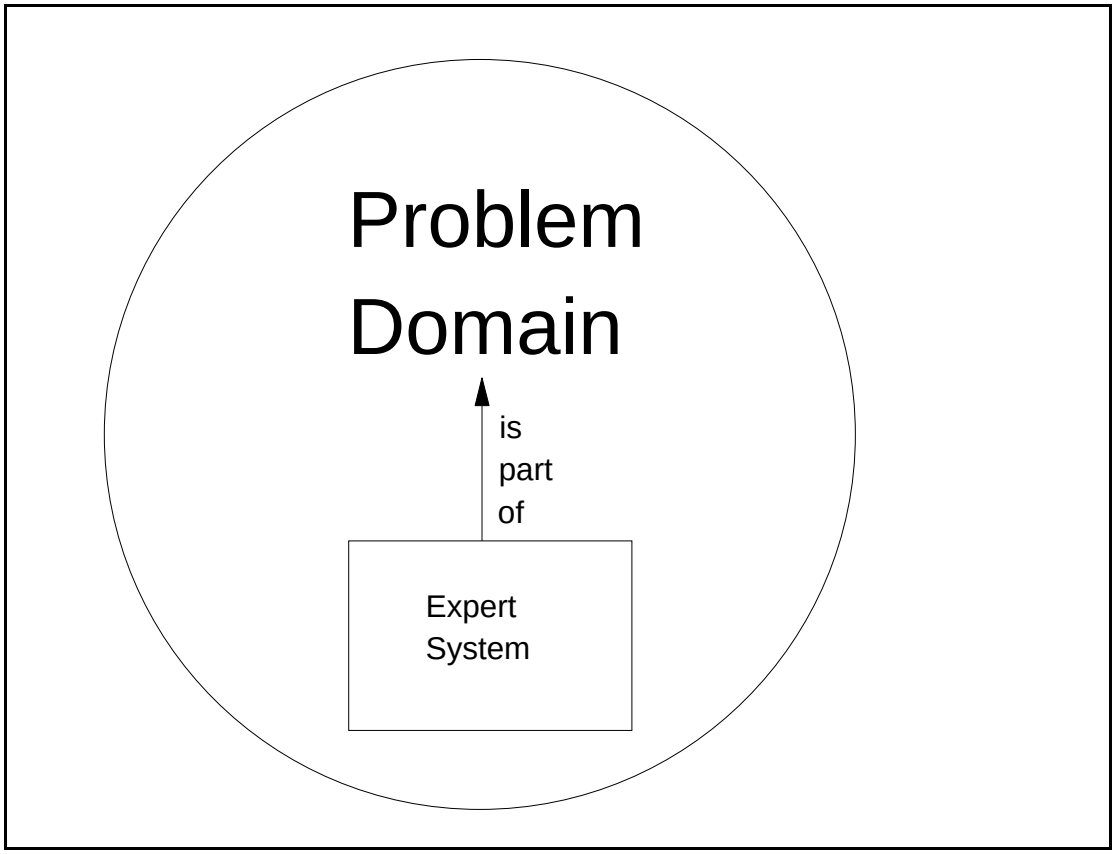


Figure 2.4 - The expert system as part of the problem domain

c. SEMI-FORMAL DOMAINS - A SOCIO-TECHNICAL APPROACH

An alternative paradigm for developing expert systems, which forms the basis for the work in this thesis, can be described as a **socio-technical** approach. This approach views the technical system as only one element in a wider "social environment". More particularly the technical components only have meaning because of the social environment of which they are a part. Instead of a single, observable reality this viewpoint considers reality to be socially constructed, a product of `actors' (individuals, groups of individuals and organisations) who give it meaning. It is the emphasis on these actors that is the primary distinguishing factor between the socio-technical and the functionalist approach.

Taking reality to be determined by actors implies that there is not necessarily a single `reality' but rather it is possible that different actors create widely differing versions of the `same' domain. These differences arise because of the differing backgrounds and circumstances of the actors. The versions of reality that are constructed by various actors depend on their beliefs and knowledge at that time. Different beliefs and knowledge can result in very different responses to a particular event. Generalising this idea slightly, it becomes apparent that time is an important factor in the creation of versions of reality. For example, the same actor may create different versions of reality at different periods of time. When the actor is given some extra (or different) information about the domain the version of the domain that is created by that actor may change.

Another factor that influences the version of the domain that is formed is the spatial location of the actor. This is particularly important if the location of the domain affects the sensory inputs of the actor. For example, a domain where it is too dark for the actor to see clearly or too noisy to hear properly will have a different interpretation to the same domain observed by an actor when these factors are not present. The other actors that are found in the domain will also influence the versions created. In many situations **norms** are created that seek to standardise the versions of reality between a number of different actors (Backhouse and Liebenau 1990).

Semi-formal domains are best described within the socio-technical approach because the lack of formal descriptions of some of the elements in semi-formal domains can be seen to arise from the differing versions of reality created by the various actors in the domain. The functionalist approach assumes a single, (formally) quantifiable reality and this does not offer the suggestion of less formally defined aspects. They are, however, a possible consequence of the socio-technical viewpoint on domains.

i. Some lessons from socio-technical information systems

Before examining the implications of the socio-technical approach on the design of expert systems a number of important ideas that have come from the application of socio-technical considerations to the design of information systems will be described. While much work in systems analysis (the process of designing information systems) takes a functionalist approach to the design of (normally) computer based systems, there is also a significant body of work that has taken a socio-technical approach to the problem. This design viewpoint has had a considerable influence on the work undertaken in this thesis and it would be possible to argue that many of the problems discussed in this thesis are not particular to expert systems; they are problems that are inherent to the design of any computer system. The solutions to the problems that are described in this thesis, however, are particular to expert systems; they utilise the nature and use of expert systems to provide solutions that may not be applicable to, for example, conventional database applications.

The need for a socio-technical approach to computer system design

The socio-technical approach arose primarily to try and explain why `technically excellent' software packages were often rejected by their users or were altered beyond recognition. It became apparent that the potential users of the system played a very significant role in determining whether or not the system was used successfully. Introducing a computer system will almost inevitably alter the structure of the organisation that receives it. This alteration will also change the allocation of power or perceived power within that organisation. Some groups, for example, the data processing centre or the accounts department, may become "more powerful" as a result of the new information system, whilst others, perhaps the supplies department or local managers, may lose some of their "power" as a result of this change. In most organisations data is seen as a "political resource" and any attempts to alter the political structure of the organisation through changes in the control and supply of data are likely to result in, at the very least, social inertia from those who see their influence being reduced. In some cases active resistance to the new information system may arise and counter-implementation techniques may be employed (Keen 1981). All of these factors lie outside of and are independent of the actual technology being introduced; indeed

the change in technology itself may be insignificant yet the effects on the organisation and hence on the success of the innovation may be considerable.

User participation

One well documented attempt to overcome some of this resistance, particularly when the information system is seen as being "imposed" from above, has been to involve the potential users of the system in the development process of the new system. In doing so the users are ideally seen to be taking a stake in the project and so are less likely to reject it. The ETHICS (Effective Technical and Human Implementation of Computer Systems) method (Mumford and Weir 1979) is one systems analysis method which has incorporated user involvement as one of its central tenets. Whilst the involvement of users in the development process is more likely to result in the final system being accepted there are a number of problems inherent in this approach. In large organisations there may be problems in finding a suitable selection of representative users to participate in the development process (Cornford and Farbey 1987). Some unions may view the involvement of users as a token gesture made by management to win over support for the new system and the actual user input to the design might be minimal. In other cases the users that take part in the design process may, after learning to understand the terminology and practices of the systems analysts, become too engrossed in the system to be able to provide a true "users" perspective on the system.

Human centred design

Cooley (1987) offers a different method of designing computer based systems. He is particularly concerned with the long term skill loss that will arise if a system is designed as a "machine" which "acts upon or absorbs [that] human competence" (Cooley 1988, p. 2). Instead he suggests that we think of the system as a "tool" which "enhances human skill and capability" (p. 2). In Japan, he notes, the move towards the workerless factory is slowly being reversed. The problem is that "if you have a workerless system, then everything has got to be highly synchronized. And if one part of the system goes down, that high level of synchronization is suddenly transformed into its dialectical opposite - it becomes a high level of desynchronization" (p. 10). To this end he proposes the design of **human centred**

systems. Such a system leaves the important, creative tasks to the human users allowing them to make use of and develop their skills whilst the computer performs mundane calculations and dangerous tasks. The users are therefore able (i.e. have the knowledge and experience) to control the system when the unforseen occurs and are able to prevent desynchronization from occurring. This human-centred design also encourages the use of formal representations as "a means of communicating *intentionality* rather than a set of instructions" (p. 12). Just as, in the past, architects used drawings to convey the general idea of what was to be built, so the computer system should communicate the nature of the tasks that are to be performed and not necessarily the actual steps involved in the task.

The role of communication

The importance of the **act of communication** rather than the details of what is communicated has been raised by a number of researchers (Lyytinen 1987, Winograd and Flores 1986). An example of this is the COED system described by Kaplan and Harandi (1989). The system they describe is designed to capture "the complex and dynamic decisions that go into the design of any software" (p. 498) since every part of the design process "must be understood in terms of how earlier parts of the design have evolved" (p. 498). The software supports the conversations that take place between the designers of the software and uses the conversations to record the decisions made in the design process. For example, one designer might decide to use a simple linked list to store a few data items and the communication of this decision through conversation with other programmers will be recorded by the system. When, some time later, the software is maintained, the original reasons for the particular choice as well as the code used to implement it will still be available in the system.

Winograd (1988) describes another software tool based on the notion of communication acts, The Coordinator. This is also structured around the act of communication rather than the details of what is communicated. One of the acts that is supported by the system is the issuing of a request to another person. The nature of the act (making a request) means that one of the following responses is expected: a promise to fulfil the request, a counter offer to modify the request or a rejection of the request. The software is designed to expect one of these further actions and acts accordingly.

d. THE COMMUNICATION OF KNOWLEDGE

As was stated above, the communication of knowledge within a semi-formal domain is considered to be an important factor in the design of expert systems and the next two chapters will highlight the problems that can arise when such communication takes place. Much research in the social sciences is concerned with the problems of interaction and communication and some of this work has been applied to artificial intelligence. The research described by Bloomfield (1987), (1988), Born (1987), Gilbert and Heath (1985) and Josefson (1987b), provides particularly useful insights into these problems.

The functionalist approach believes that the knowledge to be elicited is found in easily handled chunks which, once they have been obtained from the expert, can be easily communicated to the knowledge engineer and then used in the expert system. Much research in the social sciences suggests, however, that communica-tion is a far more involved process. Essentially it is argued that the person on the "receiving end" of the communication adds as much to the interaction as the "sender" does (Collins *et al.* 1985). In relation to expert systems this means that while an expert system may convey some of the important information about the domain, not all the information needed can be communicated and some will always exist outside the system, being added by the users.

Collins (1987) develops these ideas further and argues that "[T]echnical experts are needed to show how the rules are to be applied where widespread technical competence (common sense) do not make this clear" (p. 272). The representation techniques used by the system need the "intervention of human experts" to clarify the terms used - a skill that is located outside the system. Collins uses this analysis to classify expert systems into four groups. Class I expert systems "do no more than encapsulate ready-coded knowledge" (p. 270). Although the knowledge is readily available, this does not guarantee that the systems will be cheap and easy to build, or that they will be guaranteed to succeed. Class II expert systems are largely made up of "rules of thumb" or "heuristics". These are rules elicited from domain experts by detailed questioning. Class III expert systems have problems of a different order of magnitude. "The knowledge base of this class may be founded on the ready-coded rules of Class I or the esoteric heuristics of Class II, but differs in that there is an attempt to do away with the need for interventions of human experts between system and lay user" (p. 271). The final class, class IV, are expert systems which are effectively as capable as human experts.

Collins seems to suggest that solution to the problems of communicating knowledge are beyond the scope of formal computer systems. To some extent this is a valid argument since, as the previous discussion has illustrated, every interaction involves the receivers adding their own interpretations to the utterances made by the speakers. To imply, however, that there is no possibility of providing *any* assistance to the users in interpreting the communication, given the capabilities of today's computer systems, does seem rather surprising.

i. Interpretation and the expert system development process

The users of the expert system add their own interpretations to the output of the system and this is likely to be one area where problems can arise. However, there are also other stages in the development of an expert system where problems of inappropriate interpretation can occur.

This section will examine the typical stages involved in developing an expert system application and will highlight the areas where problems of communication are likely to arise.

Once the decision to undertake an expert system application has been made, the first step in the idealised model of the development process is a process of knowledge elicitation. This normally involves the expert communicating with a knowledge engineer. From Collins' model of communication it is apparent that the knowledge engineer will be adding to and interpreting the utterances and actions made by the expert. In almost all cases, the process of knowledge elicitation is an iterative task and the knowledge engineers have the opportunity to present the expert with their interpretations of the utterances or actions performed by the experts.

Gammack and Anderson (1990) examine some of the problems with this traditional view of knowledge acquisition and note that "[T]he expert's utterances are actively interpreted by the knowledge engineer, particularly when encoding the transcript, but also during the ongoing interview" (p.21) and recommend that the knowledge engineer be made aware of this since "the knowledge engineer's utterances and their interpretations will affect the expert's choice of utterances and their interpretations" (p. 21). They continue by arguing that the knowledge engineer's interpretation is verified by the expert so that the expert's intentions can be better conveyed to an end user and this can help minimise the problems at this stage of the development process.

After the process of knowledge elicitation the knowledge must be represented using computable formalisms. In many small scale systems the representation task is performed by the same person that undertook the elicitation. However in larger scale systems, and as the role of knowledge engineer becomes more refined, it becomes increasingly likely that the representation of the knowledge will be undertaken by someone other than the knowledge elicitor. In this case there is a second process of knowledge communication which again can cause problems if the person responsible for representing the knowledge forms an inappropriate interpretation of it. This problem is made worse because of two factors. Firstly the representation techniques are often limited in the flexibility that they offer and hence may require the knowledge to be consciously adapted to suit the knowledge representation chosen. To some extent this problem can be minimised by choosing suitably expressive knowledge representation techniques which do not require the knowledge to be adapted too much, but the problem of communicating the knowledge still remains. A second factor that may also increase the problems of communicating the knowledge is that the person who must represent the knowledge is unlikely to have access to the expert and therefore must, at best, make use of the interpretations of the knowledge elicitor which, as has been shown above, may themselves be inaccurate.

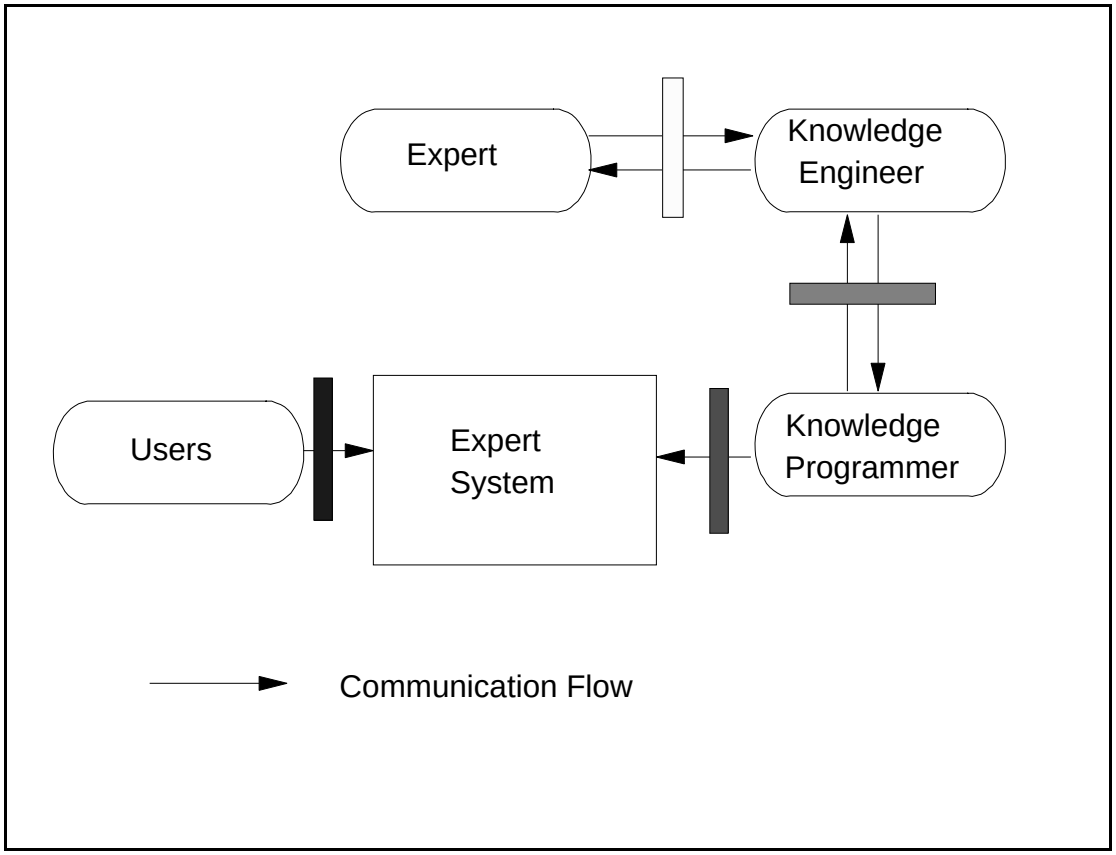


Figure 2.5 - The communication flows in the development of an expert system

Figure 2.5 shows the communication flows that occur in the development of expert systems, highlighting the areas where problems of communication are most likely to arise. A suitably designed knowledge elicitation phase, where there is continuous feedback and verification of the knowledge engineer's interpretations will keep problems to a minimum in this phase.

The final stage in this process is the interaction between the users and the computer based expert system. In this case, the only feedback or confirmation of interpretations that are available are those that may have been incorporated in the system. In conventional expert systems, however, such assistance is minimal or non-existent, partly because the functionalist approach to expert system design does not foresee such problems arising. The most important aspect of the computer based intermediary in the communication process is that there is a distinct spatial and temporal break between the expert and the knowledge in the expert system. There is therefore a tendency for the knowledge to become disembodied from the individuals who provided it and without a social environment to associate the knowledge with, there is a great danger that it will become obsolete (Stamper 1988).

The work described in this thesis can be seen as an attempt to understand the problems of communicating the knowledge of the expert to the users of expert systems through a computer based intermediary. It is hypothesized that the problems that arise are very much due to the social, informal nature of the domains that expert systems find themselves in and that a proper understanding of the problems faced as well as solutions to those problems can only come about by considering the wider social perspective as an integral part of the process of expert system development.

ii. The Dreyfus model of skill acquisition

One example of the problems of communicating the knowledge of an expert is provided by two of the fiercest critics of artificial intelligence - the philosopher Hubert Dreyfus and his brother Stuart. They argue that expert systems will never perform as well as human experts and that this a consequence of the nature of true expertise. They believe that true expertise arises through practical experience and cannot be attained by simply following rules (Dreyfus and Dreyfus 1986a). This has important implications for the communication of an expert's knowledge. If expert knowledge cannot be represented in rule format (or in some other formal representation) then it cannot easily be communicated to a knowledge

engineer or the users of an expert system. To explain how true expertise is achieved they present a five stage model of skill acquisition which will be described in some detail.

Dreyfus and Dreyfus believe that when learning a skill a person normally passes through five stages - novice, advanced beginner, competence, proficiency and expertise. Each of these stages is "qualitatively different" (p. 19) from the previous stage.

The first stage of skill acquisition is the **novice**. In this stage the person learning a skill is told to find certain **context-free** features and is told to act in a certain way when they are found. Any other features, features that relate to the particular situation are ignored and the novice may be concentrating on searching for the context-free features to such an extent that other activities, such as maintaining a conversation, may be impossible.

By using real world examples the novice gains experience and begins to recognise items immediately rather than having to use `information processing' techniques. This ability of the **advanced beginner** to immediately recognise features means that *situational* factors can be taken into account without always having to search for context-free features.

With more experience the advanced beginner will start to be overwhelmed by the number of context-free and situational features and will need to sort them into some form of hierarchy. In doing so the **competent** performer will have to take responsibility for the choice of hierarchy created. This is in contrast to the novice and advanced beginner who simply used the context-free and situational features that had been pointed out previously.

The **proficient** performer will usually be involved in the problem and will have a certain perspective on it. From this perspective certain features of the whole will tend to stand out, whilst others will tend to be less important. Although the importance of various features may vary over time, the proficient performer will not be examining individual features, instead the whole scene is examined. However, the proficient performer is still following a goal.

The final stage of skill acquisition is **expertise**. "An expert generally knows what to do based on mature and practised understanding" (p. 30). Through experience the expert reaches a stage where decisions are no longer made consciously, the expert simply acts[1]. Obviously if the decision is important, or if time permits, the expert will examine the conclusions made. However this examination will tend to concentrate on the intuitions behind the decision rather than being a conscious decision making process.

---

[1]Marvin Minsky seems to arrive at the same conclusion, although for different theoretical reasons, when he quotes Frank Lloyd Wright: "An expert is one who does not have to think. He knows" (Minsky 1986, Section 13.5).

Implications for the design of expert systems

This analysis of skill acquisition has serious implications for the design of expert systems. Current expert systems technology is rule based and the model just described suggests that rules are used only up to the level of competence and play no part in true expertise. Dreyfus and Dreyfus (1986b) use their model of skill acquisition to argue that expert systems can only ever attain the level of competence and not true expertise and that expertise cannot easily be communicated.

This model of skill acquisition also highlights the question of whether there is any need to distinguish between expertise as it is defined in the Dreyfus model of skill acquisition and `expertise' as `rule following behaviour' as it is used in much of the expert system literature? Part of the answer to this question lies in there being no word in the English language which means `more skilful' or `more experienced' than expertise. Therefore if the term is used to signify what the Dreyfus model labels as competence then there will be a tendency to devalue those skills, and the people that possess them, that cannot be articulated and communicated - even if these are more advanced skills at the level of proficiency or expertise. Josefson (1987a) notes that nurses feel that their skills are undervalued because they do not to possess a sufficiently rich language to successfully communicate what they do, whilst Cooley (1987) warns of the long term problems that will arise if true experts are ignored in favour of competent performers who, despite being articulate, do not have the practical experience that will be needed for long term development and the reproduction of knowledge.

e. EXPERT SYSTEM DEVELOPMENT FOR SEMI-FORMAL DOMAINS

Figure 2.6 summarises the two contrasting approaches to developing expert systems that were described in this chapter. The first of these approaches considers the domain to be formally defined, consisting of well defined "chunks" that can easily be communicated by the expert system. It is assumed that the knowledge base of the expert system contains all the necessary information about the domain and that the users are facing problems because they do not know all the information required for the domain. The expert system controls the interaction.

The semi-formal domain approach, in contrast, does not consider there to be a single reality, rather various realities exist which are socially constructed by various actors. Serious consideration is given to problems that can arise in communication since the users may form inappropriate interpretations of the communication performed by the system. The expert system is not seen to control the interaction, rather it is designed to assist the users in coming to an understanding of their problem and the methods that can be used to solve it.

The table presents, under the semi-formal domain approach, the main factors that were taken into consideration during the development of the practical work described in this thesis.

	FORMAL DOMAINS	SEMI-FORMAL DOMAINS
Approach used:	"Functionalist"	"Socio-technical"
Relationship between Expert System and domain	Expert System models the domain	Expert System is integral part of the domain
View of reality:	Single, measurable reality	Many possible versions of `reality'
Conceptual view of system	Designed as a machine	Designed as a tool
	Machine centred	Human centred
	Expert System controls the interaction	Expert System supports the interaction
The form of the knowledge	Knowledge is available in "discrete chunks"	Knowledge is "socially constructed"
Problems arise because:	Users "do not know enough"	Users "do not have an appropriate understanding"
Communication involves:	Simple transfer of knowledge	Users add to communication process
Involvement of users:	Little consideration given	Much consideration of users, some use of `participative' methods

Figure 2.6 - Two contrasting approaches to the design of expert systems

CHAPTER 3 - SOME PROBLEMS WITH KNOWLEDGE IN SEMI-FORMAL DOMAINS

Semi-formal domains combine formal aspects of a problem domain with elements that cannot easily be formalised. A number of examples of knowledge that cannot easily be formalised are illustrated in this chapter, showing the semi-formal nature of many expert system applications. The examples covered include the use of descriptive and subjective definitions in language based communication and the likely problems that can arise with them. It also considers some examples of what the philosopher Michael Polanyi describes as tacit knowledge - knowledge that we know but cannot tell.

a. KNOWLEDGE REPRESENTATION AND THE SYMBOLIC REPRESENTATION OF KNOWLEDGE

One immediate argument against the view that there is some knowledge that cannot be formally stated comes from recent developments in neurophysiology (see, for example, Blakemore and Greenfield 1987). Even the fiercest critics of artificial intelligence accept that the brain probably has some form of knowledge representation based on the millions of neurons found in the brain. However, contrary to early beliefs, it seems to be the *rate* at which the neurons fire, rather than whether they fire or not, that determines what is `stored in the brain' (Searle 1987). These different rates of neuron firings are used to represent knowledge about the world.

It is sometimes argued, therefore, that since the rates at which various neurons fire could be represented symbolically, *all* knowledge can be represented symbolically. Such an argument, however, is based on a mistaken understanding of what is normally meant by the symbolic representation of knowledge in artificial intelligence.

Most forms of **declarative knowledge representation** found in expert systems have a direct one-to-one relation with things in the problem domain. A semantic network, for example, is made up of nodes and links with each node being used to represent physical objects, conceptual entities or descriptors (Harmon and King 1985, p. 35). Each node,

therefore, corresponds to something in the domain. In **Figure 3.1** the nodes in the network represent things such as computers, micro-computers, tools and hammers, all of which correspond directly to features in the world.

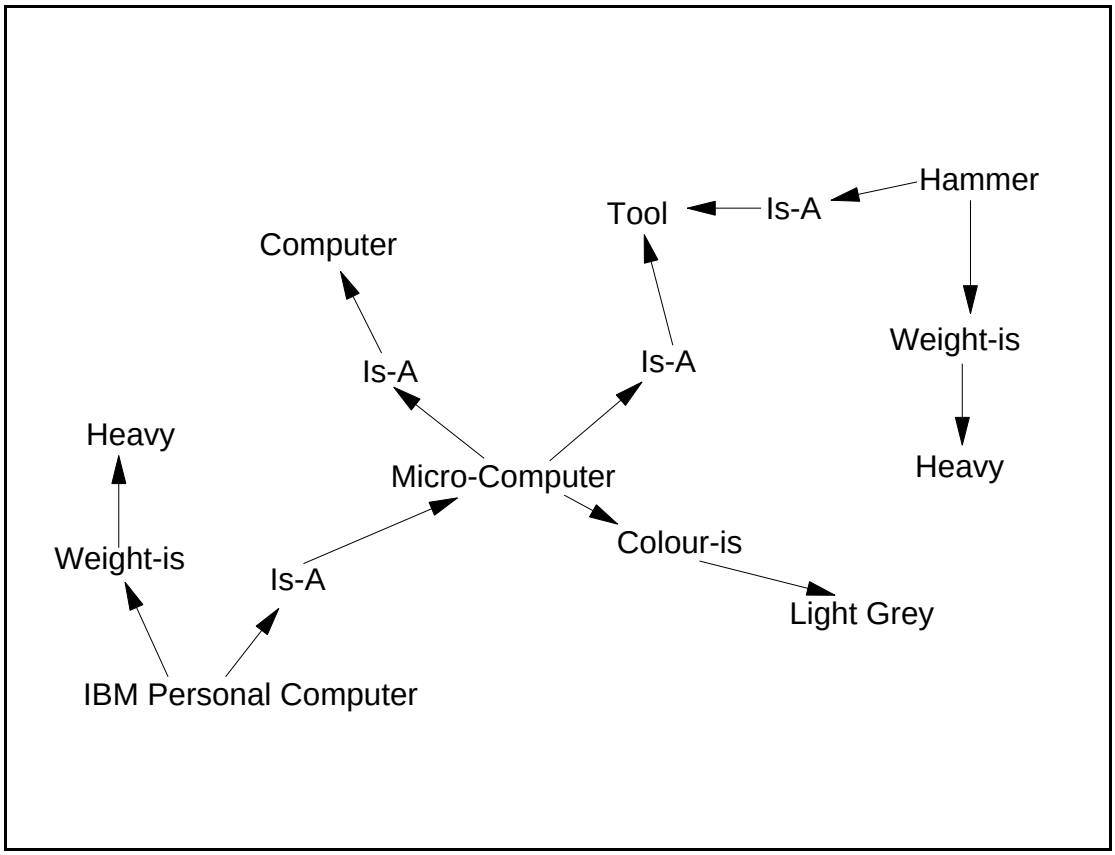


Figure 3.1 - A simple semantic network

In the human brain, however, there is nothing *a priori* to suggest that such a correspondence between the rate at which particular neurons fire and any objects in the real world exists. Dreyfus and Dreyfus (1988) report that to date, current research in **neural networks** (computer systems which attempt to mimic the biological operation of the brain) has not found any such correspondence.

b. DESCRIPTIVE DEFINITIONS - THE PROBLEM OF BOUNDARIES

One of the fundamental problems with knowledge in semi-formal domains arises from the use of language to describe the knowledge. As was described in the previous

chapter, the expert communicates with the knowledge engineer about the problem domain and the knowledge engineer then communicates with the programmer responsible for representing this knowledge in a computer based form. Finally the users interact with the resulting expert system to try and solve problems in the domain.

In each of these cases the main component of the communication is normally natural language, although other forms of communication are sometimes used as well. The most common form of communication that does not use natural language is the use of diagrams. The use of diagrams and graphics can often be more effective in conveying certain ideas than large pieces of textual information. Diagrams are particular useful for describing the spatial or structural layout of certain ideas. For example, they have been used successfully within simulation modelling to show the progress of a simulation model (Crookes and Valentine 1982, Paul 1988). Graphical images are less useful for conveying other forms of information such as causal links and there may also be problems associated with combining graphical images with other forms of knowledge.

When natural language is used for communication things in the domain (objects, concepts, entities, relationships) are given names in the language and there are two possible ways in which names can be assigned to these things; the names can be defined either **prescriptively** or **descriptively**. The distinction between these two methods of naming are important and give rise to many of the problems found in semi-formal domains.

Prescriptive definitions entail that a thing has a particular name because it satisfies clearly defined necessary and sufficient conditions. These conditions determine the occasions when the name can be used legitimately. For example, marginal cost is defined as "the increase in total cost resulting from raising the rate of production by one unit" (Lipsey 1989, p. 182). This means that for something to be legitimately called "marginal cost" it must satisfy this prescriptive definition -i.e. it must be the increase in total cost resulting from raising the rate of production by one unit. If the thing does not satisfy these conditions then, prescriptively, it cannot be called marginal cost. Prescriptive definitions ideally have an **effective procedure** to determine the appropriate use of a particular name.

In contrast, descriptive definitions arise when a thing is simply assigned a label; its `name' is arrived at through use rather than from any predefined criteria or effective procedure. For example, something is called "an expert system" because almost everybody uses that name when describing an expert system, not because it satisfies some prescriptive criteria for the use of the name "expert system".

Descriptive definitions can therefore be seen as one of the informal components of semi-formal domains and, as such, can cause problems when these systems are designed since it is possible that the users of the expert system will use different definitions of the terms found in the knowledge base to those intended by the developers of the application.

i. Examples of descriptive definitions

The problem of descriptive definitions has been examined by Susskind (1987) in his examination of the problems of applying expert systems to the field of law from the point of view of jurisprudence (legal theory). His "argument from open texture and vagueness" suggests that many words used in legislation are vague in the sense that they have no "definite set of necessary and sufficient conditions governing their use and application" (p. 187). Terms such as "fair" and "reasonable" are in this sense vague, whereas the use of the term "gold" (as a substance) for chemists seems to rest on "precise conditions governing our use of that empirical concept" (p. 187).

Roth *et al.* (1987), in describing the protocols generated when using their expert system, note that the knowledge engineer observing the experiment was frequently required to "intervene to clarify terminology, locate test points, and disambiguate expert system statements" (p. 494) so that the technician could proceed with the fault diagnosis. Their approach to the design of the system does not anticipate such problems, however, and no possible solutions are offered.

Kent (1978) describes a number of areas where descriptive definitions can cause problems for the designers of computer systems, in particular database systems but also expert systems. One of the examples he gives is the use of the name "street" asking whether a street is terminated by a city, county, state or national boundary? Does a street imply that motor vehicles can drive on it and does a street include motorways and dual carriageways? Moreover, some streets coincide with highways so should a distinction be made between them?

A contrast to this view of descriptive definitions is given by McCoy (1989). She describes a system that attempts to deal with misconceptions that arise when users interact with a computer system. She defines a misconception to be "some discrepancy between what the system believes and what the user believes (as exhibited through the conversation)" (p. 163). The assumptions that she bases her work on fall readily into the formalist approach to

designing systems. For example, she assumes that the system's model of the world contains an object taxonomy and the user's model of the world contains the same things, about which misconceptions can arise (McCoy 1989). In her system, the world has already been broken into readily identifiable pieces and hence the possibility of using different names for these pieces (or alternatively breaking the domain into different pieces with different scopes, as the examples discussed by Kent have shown) is not considered. The only problems that can arise in her case, come about because the users have incorrectly placed these pieces in the object taxonomy, and not because they use the terms in a different way.

c. SUBJECTIVE DEFINITIONS - DIFFERENT INTERPRETATIONS OF THE SAME NAME

The previous section discussed the problems of descriptive definitions. These arise because there is no set of necessary and sufficient conditions that can be used to determine when a name has been appropriately used. Subjective definitions take the problem one step further and arise when the same name is used with different interpretations.

The nature of descriptive definitions means that there are no necessary and sufficient conditions that determine the appropriate use of a particular name, moreover there are not even any guidelines as to which conditions should be used for a particular name. It is therefore possible that different groups of actors will choose to use different sets of conditions, with each group choosing the conditions on purely subjective grounds. In its extreme form, this means that the same name can be used with completely different interpretations.

Subjective definitions are a further illustration of knowledge that can cause problems in semi-formal domains since they relate to the particular individuals using the terms and the particular use to which they are being put. The meaning of the terms is "established with respect to an encompassing frame of reference lying outwith (sic) the data structures, and this background is never fully representable" (Gammack and Anderson 1990, p. 19).

i. Examples of subjective definitions

Stamper (1985, pp. 59-60) uses the term "unfit for habitation" to warn of the dangers that may arise if the possibility of subjective definitions is not considered. To the local welfare benefit office issuing cheques for building repairs on dilapidated properties "unfit for habitation" has a clear interpretation relating to the eligibility of tenants to receive funds to repair certain types of property. To the urban planning department, on the other hand, the term "unfit for habitation" means properties that are suitable for compulsory purchase orders and redevelopment since they are effectively derelict. So long as these two interpretations are kept separate, problems relating to subjective definitions should be minimised. If they come together however, for example, if the council combines all its data files in a central database, no distinction will exist between the two interpretations of the term and the planning department may act on property whose tenants are still receiving repair payments.

The idea that different actors in the domain have widely differing interpretations of a domain has been used by Checkland in his Soft Systems Methodology (SSM) (see, for example, Checkland 1981). Most conventional systems analysis techniques "assume that what `the system' is is not problematical, that the system's objectives can be defined and that alternative means of achieving them can be modelled and compared using some declared criteria, enabling a suitable selection to be made of the most desirable form of the system" (Checkland 1988, p. 242). SSM differs in that it considers defining what the problem is as being a fundamental concern. Different groups of actors all have widely differing subjective interpretations of the system and its problems. SSM tries to provide a framework for considering and combining all these differing views so as to produce a problem description which is acceptable to all the interested parties.

Subjective definitions can also arise from a particular intended use of a name, as well as when a name is used by a particular group of actors. Winograd and Flores (1986, pp. 55-56) use the question "Is there any water in the refrigerator?" to illustrate how the name "water" can have different interpretations depending on the intended use of the name. The response to the question depends on whether the question was asked to find a source of humidity that damaged photographic plates that were stored in the refrigerator, if the question was asked to find some water to drink (in which case the addition of a few drops of lemon juice, to cover the taste of the pipes that the water came from, would be acceptable), or if the water is required for a car radiator (in which case only `pure' water is acceptable).

Subjective definitions also arise in Roth *et al.*'s expert system and again the control methodology they use offers no indication as to why such problems arise nor does it suggest any solutions. They report problems of misinterpretation of questions that arose from an

"inability to assess the intentions of the machine expert" (Roth *et al.* 1987, p. 494). One example of such a question was "whether the symptoms were observed at only one setting of the device". The question was asked to test a hypothesis that a malfunction occurred at two extreme settings of the device. One of the technicians using the system was not aware of this particular choice of meaning of "observed at only one setting" and so tested the device at only one point, noted that it occurred at this setting and reported that it arose at one setting when in fact it arose at both settings.

d. HUMAN COMMUNICATION AND DESCRIPTIVE AND SUBJECTIVE DEFINITIONS

A sociological argument against the problems of descriptive and subjective definitions is given by Coulter (1985) who talks of the "disambiguation pseudo-problem" (pp. 12-13). From the belief that "any word, phrase or sentence or utterance *can*, with sufficient ingenuity, be accorded more than one meaning" it is often argued that "everything said is *inherently ambiguous*" (p. 12). In actual communication, Coulter argues, this `hypothetical ambiguity' is not normally present. This section will attempt to explain this apparent paradox by making use of the assumptions that underlie the semi-formal domain approach to developing expert systems.

The semi-formal domain approach considers a domain as being made up of many different actors and the consideration of descriptive and subjective definitions suggests that there is no inherent reason why these actors should choose to use the same definitions for names in the domain. However when these actors interact the potential ambiguity in the names used has been minimised or removed entirely. Stamper (1988) suggests that this precision and stability in the terminology used has been arrived at through "an often difficult social process of negotiating agreement and arriving at a common view" (p. 4). The process of interaction, and the need to communicate successfully, means that communities of actors will often decide to use a particular name in some `stable' form. For example, scientists have chosen to use the name "Gold" for the substance whose atomic number is 79. Once such terms have been decided upon, they become a convention or norm for that particular group for a certain period of time (Backhouse and Liebenau 1990).

The chosen use of the term becomes part of the `assumed background' of the actors and no longer needs to be made explicit in communication activities. It is assumed that everyone who takes part in communication within that particular community will base their

utterances on those assumptions. Thus "[O]bjective knowledge is not detachable from people except for limited purposes within the domain of relevance for which the negotiated agreement will stand" (Stamper 1988). Using this explanation, it is possible to see how prescriptive definitions arise. A prescriptive definition is simply a descriptive definition which has been accepted by a particular group of actors to prescribe the use of that term. Marginal cost has a prescriptive definition because most economists use that term to describe the increase in total cost resulting from raising the rate of production by one unit. Once the term has a conventional use for a certain group of actors, in many cases it also becomes the convention for other actors who were not involved in the negotiations over the use of the term.

How problems can arise

By viewing the agreement on the meaning of names as the result of a social process it is possible to see how the problems of descriptive and subjective definitions can lead to actual ambiguity in particular activities. If two communities of actors are brought together for the first time there is a reasonable likelihood that their `assumed backgrounds' and conventions will differ and ambiguities will arise in communication between the two groups. These will be resolved through the (social) use of negotiation, criticism, debate, discussion and compromise. In many cases, the potential users of the expert system and the experts who helped develop the knowledge base form different groups of actors and are quite likely to differ in their use of terms. This is particularly likely if the expert system has been designed for use by individuals who have little or no experience of a particular problem domain. They would therefore be expected to undertake a process of negotiation once they realise that there is a potential ambiguity in the names they use to describe the domain.

Introducing an expert system into this process, however, considerably alters the arrangement. No current computer based system is able to take part in a process of negotiation over the use of various names. The definitions used by the computer system, whether descriptive, subjective or even prescriptive, *cannot* be easily altered through interaction by the users. This is not to say that consideration of the users' perspectives should not be an integral part of the design process, but rather that if (when) these problems of ambiguity arise, the computer is unable to modify its approach to, and usage of, the problematic terms. Furthermore, since the experts are removed from the use of the system

both temporally and spatially, it is unlikely that they will readily become aware of the need to alter or explain the use of the terms used in the system in light of particular problems.

In addition to the suggestion that ambiguity in communication is minimised through a social process of negotiation, the notion of unspecified `background assumptions' leads to the possibility that the actors involved in communication may actually be using differing definitions of the terms used and may not realise that a difference exists. For example, two actors may refer to the "large wheel" when talking about a particular device, yet they may both be referring to *different* large wheels. Both actors will have background assumptions about the wheel being talked about and "[A]s long as it remains possible to interpret the ... responses consistently with those assumptions, the speaker's image of his partner remains unchanged, in particular, undamaged" (Weizenbaum 1983, p. 26). In such cases the users are actually using differing definitions, based on different sets of background assumptions, yet they are not aware of this discrepancy since their communication takes place without any problems arising. They don't clarify the terms that they are using because they don't know that the terms need to be clarified.

i. Example - the use of the term `race'

Many computer based applications that use data about individuals have a reference to that individuals `race'. Data about race is sometimes used to record the "racial" make up of a particular organisation (possibly to monitor and try and overcome problems of discrimination) or it may be used to determine a person's eligibility for certain benefits (for example, individuals claiming Aboriginal ancestry may be eligible for extra assistance (Clarke 1988, p. 502)). Such tasks are often suitable for expert systems applications.

Earnest (1989) provides an interesting discussion of attempts to formalise the concept of "race" for use in computer based applications. In doing so he highlights problems of descriptive and subjective definitions and also demonstrates the way that norms and conventions seek to standardise the concept, even if the concept cannot be formalised.

The first serious attempt to formalise race that Earnest came across arose when he moved to Virginia, U.S.A. in the early 1960s. On all the official forms in this state the second question, immediately after name, asked about race. In Virginia at this time, race was classified as "W" (white) or "C" (coloured). Coloured was taken to refer to all dark-skinned

people, including both kinds of Indians. Earnest took to filling in this question with "C" leaving it to the administrators to decide whether it meant "Coloured" or "Caucasian".

Sometime later Earnest was required to fill out a form for security clearance. Previously he would have simply entered Caucasian, however, with his new "awareness" of racial classification he considered his racial make up. He was aware that his ancestors were predominantly European, although there was certainly some Middle Eastern blood present as well. Given this combination of races, he decided to enter "Mongrel" as his race. After a clarification that he had not misspelled "Mongol", the form was accepted for processing. As the form was for special security clearance strict checks were undertaken into the private life of this "mongrel".

Some time later, Earnest was asked to attend a meeting at the Air Force Office of Special Investigations. After a number of routine questions, the investigators asked him why he had entered mongrel. He replied that he thought this was the best answer to an ill-defined question. He then asked them how they classified people who were mixtures of "races". They too were unclear about this concept and it turned out that the whole problem had arisen because their clearance database was not able to cope with mongrel. Everybody else, it seemed, knew their race. It did seem rather short sighted, however, for the database designer not to include "other" as a possible value.

In the late sixties bureaucrats attempted to identify minorities who might be discriminated against. However, "[T]hey never bothered to define their terms because, like the earlier racists, they had only a hazy notion of where the boundaries were" (p. 177). One form introduced for this purpose had ethnic classes including "Spanish surname" and "Black". How this was to cope with a person who was both black and had a Spanish surname was not explained.

Another approach used at Stanford University provided the classification scheme presented in **Figure 3.2**.

1) Black, not of Hispanic origin
2) Asian or Pacific Islander (persons having origins in any of the original peoples of the Far East, South-East Asia, the Indian Subcontinent, or the Pacific Islands)
3) American Indian or Alaskan native
4) Hispanic (persons of Mexican, Puerto Rican, Cuban, Central or South American or other Spanish culture or origin, regardless of race)
5) Non-minority (persons having origins in any of the original peoples of Europe, North Africa, or the Middle East)

Figure 3.2 - The racial classification scheme used at Stanford University (Earnest 1989, p. 178)

Whilst this scheme handles Hispanic blacks, it is not certain how the system would cope with people from Spain. Should they be classified under group 4 or 5? The scheme again has no option for "other" or "mongrel".

Race, as these instances have shown, is an example of a descriptive definition. There is no prescriptive criteria for the use of the term and therefore different actors (the State Legislators in Virginia, the Air Force Special Investigations Department, the administration at Stanford University) each chose their own set of boundaries around the term. Some, such as Virginia, simply used skin colour as the determining factor, whilst others attempted to classify the person according to their "background". Each particular choice of classification became the norm for that particular group at that particular time. A number of different interpretations of the term "race" have been described and each one became the norm for a particular group at a particular time. When these different groups came into contact with one another, however, the differences between the interpretations were often not made explicit since the need to do so often did not arise. For example, if one group chose to use the term on the basis of skin colour and another chose to use "ethnic background" many occasions would arise where communication between the two was successful. For example, different actors would agree that a native northern European and an American Indian were of different races, even though their reasons for making the distinction were completely different and they would never think to explain the difference between the two uses of the same term.

e. NOTICING PROBLEMS

One of the main advocates of tacit knowledge is Michael Polanyi. He considers such knowledge to be the basic term of a new epistemology (Prosch 1986, p. 52) and gives numerous examples of tacit knowledge (see, for example, Polanyi 1967, 1969). One example, which is of particular relevance to the use of expert systems, is the ability to **notice problems**.

In almost all cases, the noticing of a problem, the realisation that "something is not as it should be", occurs before a formal representation of the problem exists. The noticing of a problem occurs before the nature of the problem can be determined. Once the nature of the problem has been determined and formally specified it is simply a case of providing appropriate resources to the problem to solve the problem that has been "noticed". In some cases it is possible that the suggested solution may not actually work, however it is important to realise that the attempted solution to the problem was only devised once the need for it was noticed.

The noticing of problems involves tacit knowledge because many cases exist where the specification of the solution involves complex skills that are not possessed by the actor who first noticed the problems. Even if the noticing is done by someone who does possess the skills to also define the problem, these skills are rarely used to determine that a problem exists. This is particularly likely to be the case in those situations where finding a solution to a problem is a costly and / or time consuming process. In these cases it is unusual for an actor to be involved in examining a situation to see if a problem has arisen. A far more likely occurrence is that these skills are brought in once a problem has been noticed.

i. Examples of noticing problems

Paul (1988) gives an example of this process in the field of computer based simulation modelling. The traditional text book model of simulation, he argues, uses the simulation model "as an operational model to produce some results, or some conclusions, or for implementation after the operational model has been validated against the real world". This view is "inadequate". He continues by stating that "[T]he construction of a logical model representing the formulation of the problem is, in many instances, the most difficult aspect of the problem. In fact, understanding what the problem is may be the object of the whole exercise. ... It must be remembered that the simulation modelling process is not

designed to find the answer or answers. It is there ... to help decision makers gain an understanding of their problem".

The decision maker has noticed that a problem exists and calls in the simulation modeller. The modeller then makes use of the many tools that are available in simulation and together with the decision maker attempts to fully understand the problem that was first noticed by the decision maker. The process of using simulation modelling only takes place *after* the problem was first noticed and it is only through the use of simulation techniques that the problem can be understood and formalised.

The noticing of problems normally occurs before a formal representation of the problem exists. This means that a problem can be noticed at any time; there is no requirement for certain formal features to be present. In the context of expert systems the most likely problem to be noticed will be that the system is "offtrack" (Roth *et al.* 1987, p. 497) in its reasoning path. The realisation that the expert system is following an inappropriate reasoning path will often begin as nothing more than a suspicion that something is wrong. The details of the problem, of how the system came to follow this reasoning path, are likely to depend on choices made at an earlier part of the interaction. Again the control methodology does not allow for the possibility that the system may be wrong and the user must try and determine how it arose with a system that provides "virtually no support for ... this role" (Roth *et al.* 1987, p. 499).

f. SEEING-AS

Another human skill that makes use of tacit knowledge is the ability to see something as something else. Tilghman (1988) illustrates this by drawing a distinction between **interpreting** and **seeing-as** in the case of a technical engineering drawing. To someone with no previous experience of these diagrams they appear, at first, to be chaotic. After a while, however, the person viewing such a diagram will start to make assumptions about it and will start to interpret the drawing. For example, the person may decide that thick lines represent the outside edge of the object and that dotted lines represent lines to be cut from it. The person viewing the diagram forms hypotheses about the diagram and using these "interprets" the diagram. With experience, however, the person "may come to see the drawing as the machine it represents" and no longer has to "figure out what this is, or that line represents" (p. 307). This behaviour of interpreting by a novice is contrasted with that of an experienced engineer who see the drawing `correctly' all along. Tilghman draws his theory from

Wittgenstein who marked the difference between interpreting, which is an activity, and seeing-as, which is a state (Wittgenstein 1953, Part II, section xi). Tilghman points out that much of the difficulty with distinguishing between seeing-as and interpreting lies in the identical description of what was "seen" that can come from both cases. If the novice interprets certain lines as being the surface and others as representing lines to be cut from that surface, the novice may come up with the same description as that given by the experienced engineer.

Seeing-as is not restricted to high level tasks such as understanding technical engineering drawings, it also occurs in everyday life. For example, legislation introduced in early 1989 requires all petrol stations to display the price they charge for petrol on their `main' road signs in litres rather than gallons (gallon equivalent prices may be displayed on the pumps if necessary). At first drivers will interpret the price in litres by multiplying it by 4.5 to obtain an approximate price in gallons. With experience, however, they will cease to interpret the price and will immediately see whether a price in litres is good value.

i. Examples of seeing-as

The tacit skill of seeing-as is often implicitly recognised - and avoided - in many accounts of developing expert systems. This is frequently done through the use of the "telephone heuristic" - "think about whether the problem can be described in words. Could it be solved over the phone? If the expert needs to see or touch the data, the task may not be well suited to an ES" (Leonard-Barton and Sviokla 1988, p. 95). Effectively, if the expert makes use of the tacit skill of seeing-as then the task is not suitable for the development of expert systems. Harmon and King (1985) and Twine (1988) make the same point.

In a limited number of cases problems are tackled which involve the expert seeing things as other things. However, unless the possibility of seeing-as is considered, the behaviour of the expert seems unusual. For example, Paris (1988) describes attempts to create an expert system to assist a "pathologist in the differential diagnosis of Lymphoma / Epithelial Tumour biopsies on slide evidence alone". He notes that the pathologist first gives the diagnosis and secondly gives reasons for it. "In other words it appeared that the pathologist simply *saw* the diagnosis and then attempted, maybe to please the observer, to justify it" (Paris 1988).

g. READINESS-TO-HAND AND `HIDDEN' KNOWLEDGE

`Hidden' knowledge is another form of knowledge that cannot be formally represented, however it is introduced primarily as an analytic concept for use in the next chapter. The term `readiness-to-hand' was introduced by Heidegger in 1927 and it is a different approach to considering knowledge of the world. Rather than considering things as having certain characteristics and features, readiness-to-hand suggests that much of this knowledge is `hidden' from a person until it becomes `unready-to-hand' and `visible'. A hammer and a blind man's cane have been used extensively in the literature, so the use of a "mouse" input device in a window based computer environment will be considered instead.

The mouse is used to move a pointer around on a screen. In doing so it can "point" to icons, activate pull-down menus and select different items from the screen. However, when using mouse devices the users do not actually consider any properties of the mouse, they simply think about moving the pointer. The mouse itself is ready-to-hand, its properties are not represented in any way; they are hidden to the users. They only "discover" the mouse through its unusability or unreadiness-to-hand, when the interaction *breaks down*; only then does the knowledge of the mouse become explicit.

It is only when the mouse does not move as expected - perhaps it causes the pointer to move in the opposite direction to the mouse - that the users become aware of any of its characteristics, it becomes unready for the task they wish to use it for. The features of the mouse can also "reveal" themselves when a mouse is not present and a keyboard must be used instead. In this case features of the mouse such as its speed and ease of use become unhidden through the unreadiness-to-hand of the mouse. Finally the obtrusiveness of the mouse may make it appear unready-to-hand. If the users want to make small, accurate movements of the pointer, they may find that the design of the mouse hampers this activity. Once again, it is only when the use of the mouse "breaks down" that they become aware of its features.

h. TOWARDS SOLUTIONS TO THE PROBLEMS RAISED

The first set of difficulties with knowledge in semi-formal domains arises from the way in which things (objects, concepts, entities) are named. It was suggested that many things in the domain have descriptive rather than prescriptive definitions that determine the appropriateness of their name. Descriptive definitions are arrived at through negotiation and

compromise; a social process involving individuals and groups acting for a common purpose in a common setting. It can be argued that prescriptive definitions are simply descriptive definitions that a particular group has accepted for use in a certain purpose. Descriptive definitions cause difficulties because clearly defined boundaries do not exist around the concepts being described.

The difficulties with descriptive definitions are made worse when the choice of which boundaries should apply to a name are made on the basis of subjective judgements. Cases can arise where the subjective boundaries of the things being named are mutually exclusive between different groups and unless those using the names are aware of the wide disparity of meaning for the terms being used, unless they are aware of the differing boundaries, serious problems may arise.

When expert systems are designed it is inevitable that some extra formality will be introduced into the language used. Any proposed solutions cannot completely overcome such considerations, however they can be designed with the problems in mind.

Another effective boundary to the amount of domain knowledge that can be stored in the knowledge base is the extent to which the experts use tacit skills when solving the problems. Although these tacit skills cannot be properly represented in a knowledge base, the system should still be designed to take account of this knowledge.

The nature of the tacit skill of noticing problems means that the skill cannot be switched on and off at will and so may arise when the users are using the system to solve a problem. They may realise that the goal the system is trying to arrive at is obviously wrong, whilst another goal may be far more appropriate. In this case, the users should be able to make the system abandon its current line of reasoning and examine the alternative suggested by the user.

CHAPTER 4 - COMPUTER BASED SYSTEMS WITHIN SEMI-FORMAL DOMAINS

The discussion of the two approaches to developing expert systems suggested that the formal approach to design has the expert system operating on a (separate) model of the domain. In contrast, the semi-formal domain approach considers the expert system as one of the components of the social environment making up the domain. It is therefore necessary to consider any implications that arise from embedding expert systems within such domains. This chapter will argue that there are three important consequences that arise from a computer based system being located within the domain. Firstly, it is shown that computer based systems have very limited capabilities for considering situational factors that arise within the domain. Secondly, this means that expert systems cannot become aware of **situated actions** and hence it is not possible to `plan' for many of the problems that may arise. Thirdly, the concept of confusion will be examined and it will be shown that computer based systems cannot normally become aware that a state of confusion has arisen. A domain independent understanding of how confusion can arise will then be considered.

a. THE COMMUNICATIVE RESOURCES OF COMPUTER BASED SYSTEMS

In their discussion of knowledge engineering, Gammack and Anderson (1990) highlight a number of important considerations that relate to the problems of communication between experts and knowledge engineers. They suggest that any process of knowledge elicitation takes place within the context of an unarticulated background and that both parties need to be aware of this background if they are to minimise any possible misunderstandings that may arise. Examples of such misunderstandings were discussed in the previous chapter. Unfortunately, their work only covers the interaction between the human expert and the human knowledge engineer. In particular, they do not consider the problems of human users interacting with (computer based) expert systems (Figure 2.5 shows how these two interactions are related). This is significant since there are many qualitative differences

between humans and computers that become increasingly important when they are combined in a semi-formal domain.

Computer based systems differ from humans in that they are limited in the ways that they can obtain information about the domain which they find themselves a part of. Humans, on the other hand, have far more **communicative resources** available to them. They can see, hear, touch, smell and taste. In addition their whole bodies provide information about the domain.

Suchman (1987) undertakes a critical examination of the communicative resources of computer based systems by considering an `expert help system' for a large and relatively complicated photocopier. The help system is designed to provide assistance to the users and help them operate the machine[1]. It would appear that the provision of such a help system would be relatively simple given that the possible tasks being tackled are limited by the functionality of the photocopier, that the structure of the interaction is procedural, partially enforced and a criteria of adequacy for each action can be specified (Suchman 1987, p. 99). For example, one of the possible tasks that can be undertaken is to make multiple copies of a bound document. The structure of this task is well defined: a copy is made of the bound document and this copy is then used to make further copies. The task has a partially enforced structure which can be verified by the system: multiple copies can only be made once the first copy of the bound document has been made and it is possible to have sensors that verify that the Bound Document Aid has been used in the process.

When the photocopier was actually used, however, a number of problems arose. For example, the users of the copier did not always realise that the single copy of the bound document was used to make the remaining copies and in many cases the help system was not able to respond to difficulties faced by the users.

Suchman's framework

[1]Suchman uses "machine" to refer to the computer based system and as such does not distinguish between "machines" and "tools".

Suchman suggests that these problems are a consequence of the limited communicative resources of the photocopier and proposes a useful analytic framework, shown in **Figure 4.1**, to help understand the problems.

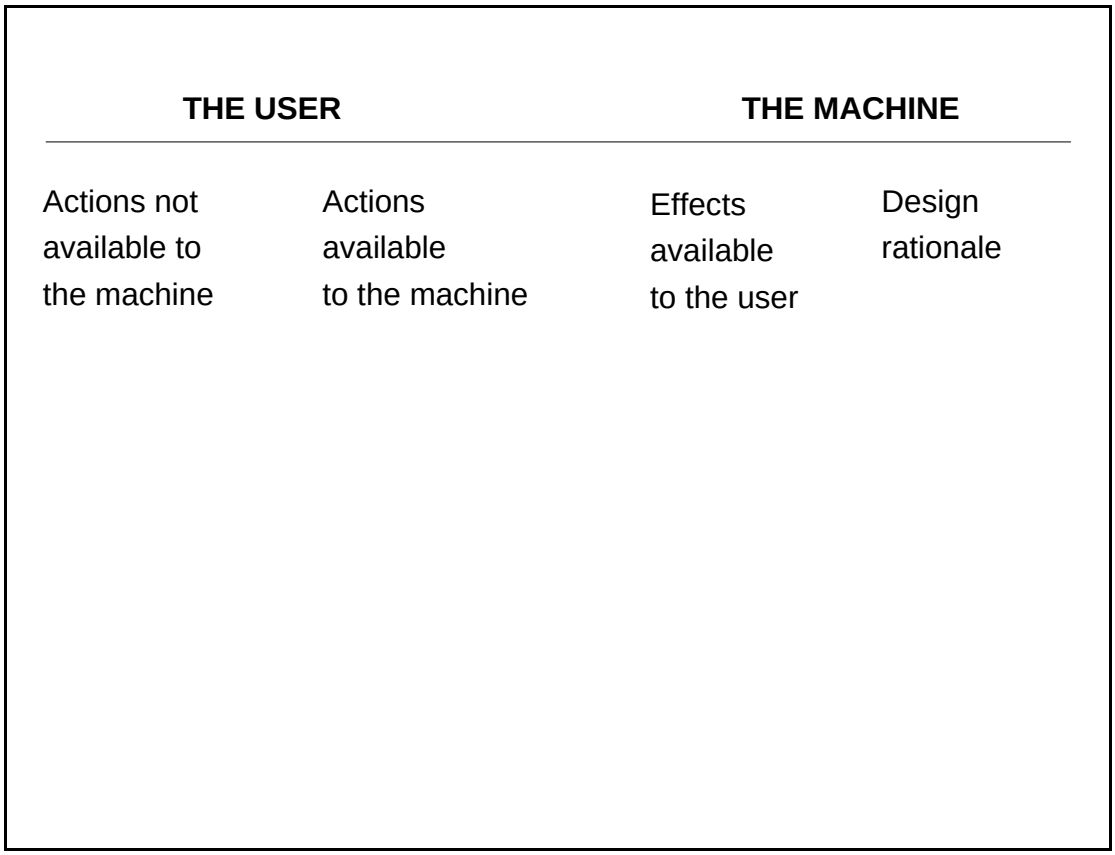


Figure 4.1 - Suchman's framework for analyzing the communicative resources of users and machines (Suchman 1987, p. 116)

The framework is made up of four columns, each of which describes part of the interaction between the users and the machine. The columns are: **actions not available to the machine**, **actions available to the machine**, **effects available to the user** and **the design rationale of the system**. Only the two centre columns are directly available to the users and the machine. The users can read the help screens and instructions provided by the system but do not normally have access to the design rationale behind the display of those screens. In a similar way, the machine has access to the menu items chosen by the users, but not to the intentions of the users that led to that choice.

THE USER		THE MACHINE	
Actions not available to the machine	Actions available to the machine	Effects available to the user	Design rationale
		DISPLAY 1	Selecting the procedure
B: It's supposed to - it'll tell "Start," in a minute A: Oh. It will? B: Well it did: in the past (pause) A little start: box will:			
		DISPLAY 4	Ready to print
B: There it goes A:"Press the Start button"			
	SELECTS START		
		STARTS PRINTING	
Okay.			

Figure 4.2 - Sequence II (Suchman 1987, p. 126)

In the example shown in **Figure 4.2**, the users are able to see the display screens (DISPLAY 1 and DISPLAY 4) and can also tell when the machine starts making copies. The design rationale behind the choice of displays shown, however, is not available to them. Similarly the machine can only tell when the users select START but their deliberations before doing so are not available to it.

Another of the examples described by Suchman (pp. 165-169) involves the users making four copies of a document using the Recirculating Document Handler. In this case when prompted by the system to place the document in the handler, only one sheet is inserted. The limited access of the machine to the actions occurring in the rest of the domain means that all that the sensors note is that "the document" has been inserted into the handler. When the START option is then selected, the copier makes four copies of the "document" (i.e. the single sheet). The system has completed its task and awaits the selection of the next task. The users, however, who do not have access to the design rationale of the system, await instructions to insert the next sheet - which are never displayed.

This problem cannot be overcome by simply adding more sensors to the photocopier. No number of sensors could determine that the document to be copied was made up of more than one sheet unless the copier had access to the conversations of the users (or, if there was only one user, the thoughts of the user). One way for the copier to become aware of the number of sheets in the document would be to explicitly ask the users to enter the number of sheets to be copied, but this is a rather unusual and cumbersome approach. Alternatively the designers of the machine may simply limit the possible factors that can influence its operation. This, however, resorts to the belief that the "computer knows best", found in the functionalist approach to design.

i. The communicative resources of expert systems

Expert systems, as a form of computer based system, have the same limited access to the rest of the domain as the photocopier described previously. In most expert systems the range of actions available are limited to the keys pressed by the users (in addition to any selections made using other input devices such as a mouse). The system is only aware that a particular fact is true because the users selected a certain option.

In some cases, the expert system has sensors that can be used as a direct source of data about the domain. For example, a process control expert system may have access to sensors informing it of the temperature of the material at various stages in the development process. In these cases, however, the sensors are not normally used to check the responses of the users, rather they are used as alternative input devices.

b. SITUATED ACTIONS

Planning is described by Cohen and Feigenbaum (1982) as "deciding on a course of action before acting", with the implication that a plan is therefore "a representation of a course of action" (p. 515). Plans are to be recommended otherwise "less than optimal problem solving" will arise. A plan is therefore a precise, `prescriptive' model of future actions that will efficiently achieve a particular goal. Plans are also sometimes used as post hoc descriptions of actions used to achieve a goal. **User-models** are often used by expert systems to understand the purposeful actions that may arise in an interaction since the models

can be used to determine what problems the users are going to face by determining their likely cause of action, or if they have already arisen to determine what caused them.

Suchman's work, however, suggests that the conventional model of planning fails to take into consideration the problems of the limited communicative resources of computer based systems and, more significantly, mistakes the very nature of planning. Instead of being based on planning, she argues, "purposeful actions are inevitably *situated actions*" (p. viii), by which she means "simply actions taken in the context of particular, concrete circumstances" (p. viii).

Suchman believes that the conventional view of planning mistakes the role that plans play in purposeful action. The model assumes that "[M]utual intelligibility is a matter of reciprocal recognizability of our plans, enabled by common conventions for the expression of intent, and shared knowledge about typical situations and appropriate actions" (p. 27). That is, it is only possible to understand an individual's actions by considering the plan that the person is assumed to be following.

Plans suggest that the precise steps needed to achieve a particular goal are stated before the event. In actuality many different actions can be taken to achieve the same goal; the choice of which action depends critically on the particular circumstances that the action is taking place in and *cannot* be predicted in precise terms in advance. For example, the actions involved in actually posting a letter will depend on whether you meet someone who is going to the post office, whether you pass an internal mail basket or if you actually go to the post box yourself.

Another important feature of the planning model is the assumption that the content and organisation of the background knowledge can be made explicit so that relevant features can be identified and utilised. Suchman suggests that in reality such actions do not use a knowledge *of* the situation, rather they take place *in* the situation, with features appearing relevant to the individual in much the same way as they do for the proficient performer and expert in the Dreyfus model of skill acquisition[2].

The analysis presented by Suchman suggests that "[P]lans are a constituent of practical action, but they are a constituent as an artifact of our *reasoning about* action, not as a generative *mechanism* of action" (Suchman 1987, p. 39, emphasis in the original) and as such are "best viewed as a weak resource for what is primarily *ad hoc* activity. It is only when we are pressed to account for the rationality of our actions, given the biases of

---

[2]Hubert Dreyfus was on Suchman's dissertation committee at the University of California at Berkeley and was very enthusiastic about her project (Suchman 1987, p. xii).

European culture, that we invoke the guidance of a plan" (p. ix). Plans made in advance of an action are necessarily vague because these situational factors *cannot* be determined in advance, whilst those plans described in retrospect "filter out precisely the particularity of detail that characterizes situated actions, in favour of those aspects of the actions that can be seen to accord with the plan" (p. ix).

It is important to emphasize that the concept of situated actions is not arguing that plans are unimportant, or that plans are not used, rather it is argued that plans have a different role in action. In many cases it is beneficial to plan action so as to minimise risks or maximise utility by highlighting important factors to be considered. Plans are used in such a manner "to orient you in such a way that you can obtain the best possible position from which to use those embodied skills on which, in the final analysis, your success depends" (p. 52).

When expert systems are embedded in semi-formal domains, situational factors become increasingly important. Interpretations of the terms used, as well as the availability and use of tacit knowledge / tacit skills will be affected by the particular circumstances of the problem and the users of the system. It will therefore be very difficult to anticipate all the likely problems that may arise in the use of the expert system.

c. CONFUSION

When the users of an expert system form an inappropriate interpretation of its output, or when some other knowledge that could not be represented within the boundaries of the knowledge base causes problems in the interaction, it is common to say that some **confusion** has arisen, that the users are "confused". What precisely is meant by the term "confusion" is rarely made clear and a closer examination of this concept will provide a useful analytical technique for considering the problems of interacting with a computer based system.

To understand what "confusion" is, consider the case where an actor asks the question "Is there any water in the refrigerator?" with the "water" in this question referring to "pure water", suitable for use in a car radiator. On receiving an affirmative answer, the actor proceeds to go to the refrigerator and finds only water with a twist of lemon in it. In this case, the actor may well say that the response to the question was a result of confusion on the part of the other actor, who did not properly understand the meaning of the question. To this actor, confusion has arisen.

Now consider this same interaction from the perspective of the actor answering the question who assumed that the speaker wanted some water to drink. To this actor there has been no confusion. The appropriate response was given to the question asked. Thus whether or not there is confusion in a situation depends critically on who is viewing the situation. The actor asking the question believes that confusion has arisen, the actor answering does not.

More significantly, neither actor was aware of the "confusion" at the time it "arose", since if they did, they would have reacted immediately and clarified the situation. Even the actor who (later) believed that confusion had arisen *only* realised this when the consequences of the difference revealed themselves (became unhidden). Until this time, the questioner was as "blissfully unaware" of the potential problems as the respondent.

This means that it is not possible to be aware of confusion at the time that it occurs. At best it is possible to realise that confusion occurred in the past when its effects reveal themselves in the current situation. Moreover, unless the effects do reveal themselves, none of the participants in the interaction will ever realise that confusion has occurred.

To summarise, it is only possible to know about confusion when one of the participants in the interaction becomes aware of a problem. Once the problem has been noticed it may be possible to determine when the actual confusion arose however it is not possible to be aware of confusion as it occurs. Once confusion has been noticed, however, there is normally a commitment to clarify and resolve any ambiguity or misinterpretation or to attempt to convey the missing tacit knowledge.

i. The commitment to resolving confusion

In human communication such clarifications to overcome confusion are normally attempted and Winograd and Flores (1986) suggest this is due to the commitments formed by speakers and hearers "by virtue of taking part in the conversation" (p. 58). These commitments mean that whenever a problem arises "the speaker is committed to give an account" (p. 60) to explain how the breakdown arose.

They suggest that many of these problems arise because of differences in the "assumed backgrounds" (that domain knowledge that is assumed to be common to all

members of a particular community) of the participants. To resolve the problems, the participants may decide that a particular utterance was simply inappropriate, or alternatively they may "articulate part of the assumed background" (p. 60) and come to an understanding about how the problem arose and how it is to be resolved.

Attempts to overcome problems associated with confusion can only be tackled once that confusion is noticed and this process is based on the tacit skill of noticing problems. This suggests that the circumstances for noticing a particular problem cannot be formally specified in advance. However, the typical mode of operation of an expert system offers a domain independent way of describing the situations when problems are likely to be noticed. This approach *does not* determine which problems are going to be noticed, nor does it determine when they are going to be noticed. It does, however, provide a theoretical description of the possible actions of the system that can lead to noticing of problems, and hence of confusion.

The following description is based on **speech act theory** which was introduced by Austin and was later developed by his student, Searle. A brief introduction to speech act theory will now be given.

ii. Saying and doing

Many of the utterances made in the course of a conversation do not relate to true or false statements about the world, rather, as Austin (1962) points out, in many cases the uttering of words may actually be performing a **speech act**. For example, when, in certain circumstances, the words "I bet you ..." are uttered they are *not* describing what has been done, they are not true or false statements about the world, rather by saying the words the bet has been made. Similarly when requesting a particular action uttering "I request ...", in certain circumstances actually performs the request. The "certain circumstances" that affect the bet or request need to be specified in more detail and Austin hoped to discover what these "circumstances" were by "looking at and classifying types of case in which something *goes wrong*" (p. 14). He labels those things that go wrong as **infelicities** and describes three general principles which need to apply if infelicities are not to occur:

1. There must be some conventional procedure which has a conventional effect, in certain circumstances and that the particular persons and circumstances involved in the speech act are appropriate for the procedure being invoked.
2. That the procedure must be executed by all participants correctly and completely.
3. That the procedure is designed for certain intentions, thoughts or feelings and that those involved must have the appropriate intentions, thoughts or feelings.

Figure 4.3 - Austin's three principles to prevent infelicity in speech acts

To illustrate this point, for the utterance "I bet ..." to be felicitous (appropriate or non-defective), there must be conventional procedures and circumstances that hold for making bets and the particular case must conform to these circumstances. Thus bets can only be placed before the outcome of the event being bet upon and formal bets can only be made in a Bookmakers. Some acts may only be `appropriate' when performed by certain people, for example, a formal bet can only be made with the employees of the Bookmaker who are authorised to take bets, and not, for example, with the cleaners. It is also necessary for all the participants to completely and correctly finish the procedure, thus even if the words "I bet ..." are uttered, the bet will not actually be made until there is some response to it - either in the form of a betting slip, or perhaps the verbal response "OK. The bet is on". Whilst the first two conditions can affect the entire act, whether the act is valid or void, the third condition affects the intentions behind it. If I say "I bet" knowing full well that I do not intend to keep the bet if I lose, then the bet is still valid, but the person who bet against me would have justified grievances.

iii. Speech acts and the noticing of confusion

Attempts to rectify confusion can only be made after that confusion has been noticed and consideration of speech acts offers a theoretical basis for determining when this confusion is likely to be noticed. In the course of assisting the users in solving a problem, an expert system may **request** that certain information about the domain be provided, it may **question** the users about the domain and it may make **assertions** about the domain. These three things are speech acts and in performing them the expert system may cause the interaction with the users to breakdown. Most commonly this is a result of the intentions

behind the act, for example, the expert system may assert a fact which the users do not feel it is justified to assert. This means that their unstated background assumptions may be challenged by the actions of the system. In discussing readiness-to-hand in the previous chapter, three possible forms of breakdown or unreadiness-to-hand were described. The first of these is when something unexpected occurs. The second is when something expected does not occur and the third arises when the tool becomes obtrusive in the act it is performing. These three causes of breakdown can be seen in terms of the felicity conditions of the speech acts undertaken by an expert system.

This description of felicity conditions (Searle uses a related notion of a speech act being non-defective) for these three acts is due to Searle (1969, ch. 3).

The conditions necessary for a request (for a future act of the hearer) not to be defective include the ability of the hearer to perform the requested act. Also the speaker must believe that the hearer can perform the requested act. It must also not be obvious to both the speaker and the hearer that the act would be performed by the hearer in the normal course of events if the request was not made. If the request is to be `sincere', the speaker must actually *want* the hearer to perform the act and it is essential that the request must count as an attempt to make the hearer perform the act.

The act of asking a question is similar to that of requesting an act, except that in asking a question the speaker wants to know about the `truth' of a proposition (or propositional function), as opposed to making the hearer perform an act. The question would be defective if the speaker `knew the answer' and it must not be obvious to both parties that the hearer would provide the information about the proposition at that time without being asked. Obviously there are some questions, examination type questions, where the speaker does `know the answer', but in these cases the speaker still wants to know if the hearer can provide the answer and it is not expected that the hearer will provide the information without the question being asked. If the question is to be sincere the speaker must want the information asked about and the asking of the question must count as an attempt to elicit this information from the hearer.

The final speech act that needs to be examined in relation to the functionality of expert systems is the act of stating or asserting a particular proposition. For an assertion not to be defective, the speaker must have evidence or reasons for the truth of the proposition and it must not be obvious to both the speaker and the hearer that the hearer knows the proposition (or that the hearer does not need to be reminded of the proposition). A sincere

statement requires that the speaker `believes' the proposition and the assertion must count as an undertaking that the proposition represents an actual state of affairs.

d. TOWARDS SOLUTIONS TO THE PROBLEMS RAISED

The first part of the chapter discussed the ability of an expert system to become aware of the domain it is in. In most cases, the communicative resources of a computer limit this to an awareness of the key presses made by the users in addition to any direct sensor readings taken from the domain.

In particular, the expert system is unlikely to be aware that the users of the system have been confused and one consequence of this is that the users themselves must act to overcome any confusion that they notice.

The three speech acts performed by an expert system: requesting, questioning and asserting, are likely to cause the users of the expert system to become aware that confusion has arisen if they believe that the felicity conditions of these acts have not been met. To overcome these problems, the users must be shown that these conditions have actually been met.

CHAPTER 5 - INTRODUCTION TO THE PESYS SYSTEM

This chapter describes the expert system development tool that was chosen as the basis for the practical work described in this thesis. A description of available expert system development tools and the reasons for the particular choice made are described in Appendix II. This chapter describes the main features of the development tool, highlighting those additions found in the system that are not found in other tools. The structure of the knowledge base, inference engine and user interface are also described in some detail. Finally a number of applications that have been developed using the tool are described. It should be noted, however, that the characterisation of the system given in this chapter is not complete since it does not take into consideration those specific additions that relate to semi-formal domains which are described in the next two chapters.

a. AN OVERALL DESCRIPTION OF THE PESYS ENVIRONMENT

PESYS (*P*ascal *E*xpert *SY*stem *S*hell) incorporates many useful features that allow the developers and users of the system to maximise the potential of this technology in semi-formal domains. This chapter outlines the facilities offered by PESYS other than those described in the next two chapters. A user guide to the PESYS system, together with a simple tutorial for developing expert systems is provided in Appendix III.

PESYS offers the developers of applications full control over the inference engine used with a particular knowledge base. It is possible to select the inference method (or combination of inference methods) that are to be used as well as whether a log is to be kept of the actions performed and the accuracy to be used in numerical comparisons. The shell also includes an implicit design method which encourages the proper, structured design of the knowledge base and includes the use of "sub-knowledge bases" which can be used to structure the application.

The implementation of the inference engine is very efficient and allows the users of the system to rapidly arrive at solutions to their problems. Their interface with the system is designed to be clear and consistent in use offering access to all the necessary functions of the

system without any unnecessary inconvenience. A comprehensive What-if facility is also available to allow the users to see the possible effects of changing certain data values. However, despite providing considerable functionality, the system itself is very compact (the executable file is less than 150K in size) and this leaves most of the computer's memory available for the knowledge base and working memory.

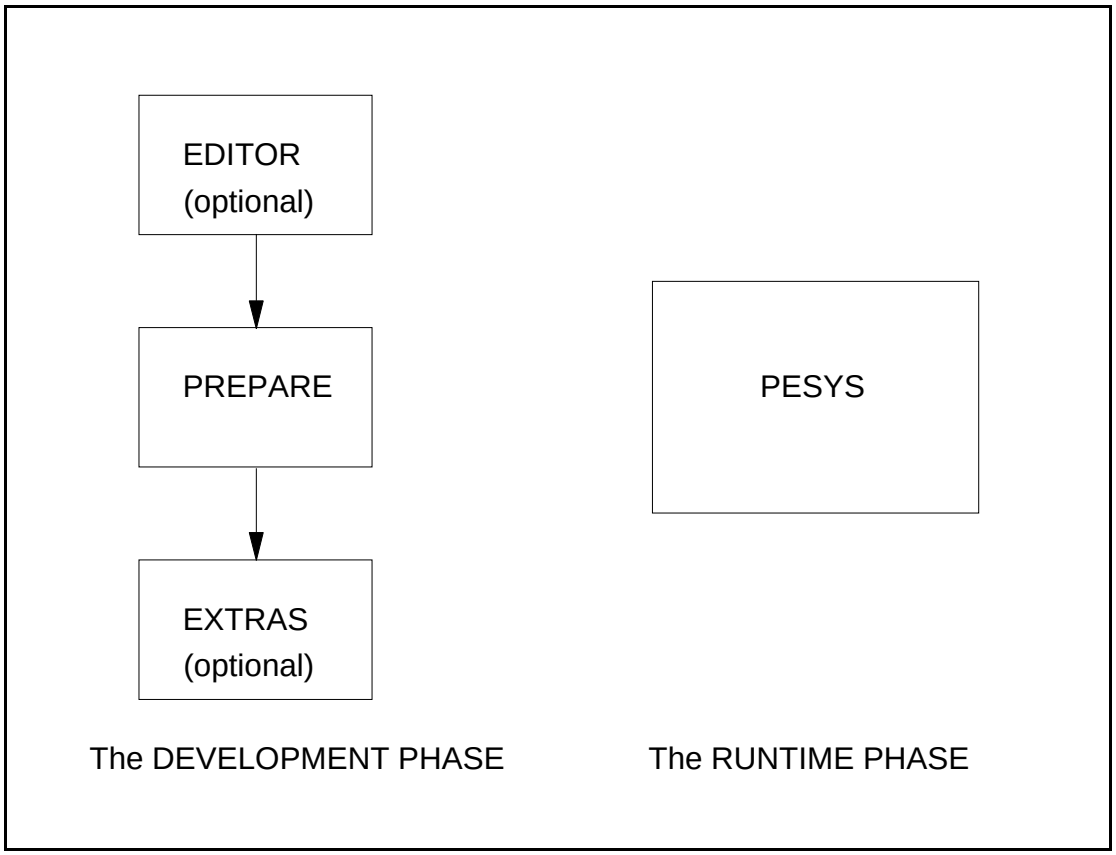


Figure 5.1 - The development and runtime components of PESYS

The PESYS environment is made up of two components. The first component relates to the development of expert system applications and supports the writing and preparation of knowledge bases. The second is the runtime environment and it is this program that users interact with when trying to solve problems in the domain. **Figure 5.1** shows the programs in these two components.

b. THE STRUCTURE OF THE KNOWLEDGE BASE

The knowledge base in PESYS is made up of a number of related files. To assist in the management of these files, all the files that relate to a particular knowledge base share the same **main name**, for example, all the files in a knowledge base used to repair faults in a photocopier might have the main name "COPIER". The various files that are created by the different programs in the development stage of the environment have different extensions to the main name that describe their contents. **Figure 5.2** shows the files that are created at each of the stages of the development process.

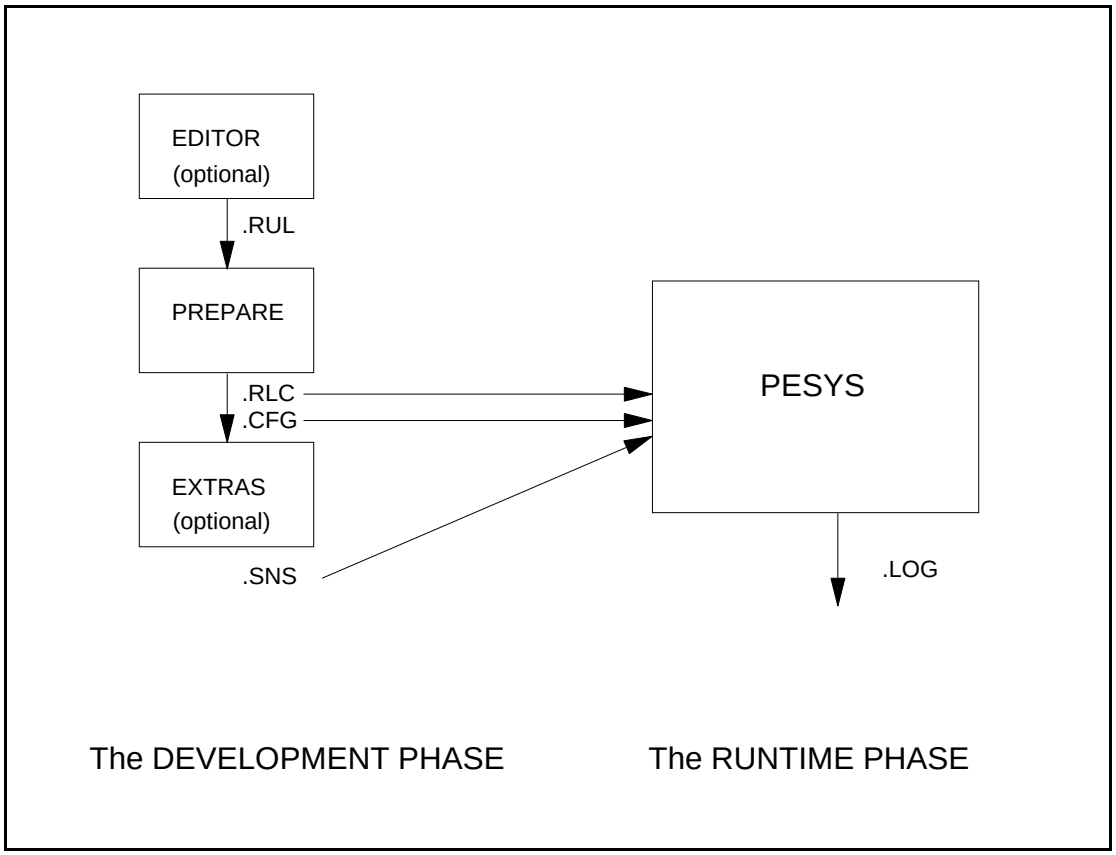


Figure 5.2 - The files used in PESYS

i. The files in a knowledge base

The first file to be created has the extension **.RUL** (RULe). This file contains the original rules associated with the knowledge base. The rules are entered by the knowledge base designer using a simple text editor such as the program EDITOR. All the files in PESYS are stored in plain ASCII format. This allows them to be examined and edited using any suitable editing program or word processor. Once the rules have been edited, the .RUL

file is passed to the PREPARE program which precompiles the knowledge base and forms the **.RLC** file (RuLe Coded). The PREPARE program also allows the designers of the application to specify the configuration of the knowledge base, choosing which inferencing methods are to be used as well as general system characteristics such as whether a log of the interaction is kept and the accuracy used in mathematical comparisons. The configuration information is stored in the file **.CFG** (ConFiGure).

The program EXTRAS allows the developers of an application to specify the text of questions that are asked when the users are required to enter values for variables in the knowledge base. The text of these questions is stored in the file **.SNS** (common SeNSe information). When the knowledge base is used by the runtime system a log of the interaction may be kept, this is stored in a numbered file with the extension **.LOG**.

ii. The basic structure of a rule in PESYS

The primary knowledge representation technique used in PESYS is the if-then or production rule. This knowledge representation technique is very common in expert systems. PESYS, however, includes a number of special features in its implementation of production rules that need to be described further. A typical production rule is shown in **Figure 5.3**.

```
RULE-1
IF
the first if-clause is true
and the second if-clause is true
THEN
the first then-clause is true
and the second then-clause is true
```

Figure 5.3 - A sample rule in PESYS

As this rule shows, a typical production rule is made up of three main components. The first component is the **rule identifier**. This is followed by the keyword **if** and a number

of **if-clauses**. No limit is imposed on the number of if-clauses in a rule, there can be none, one or as many as can fit in the memory of the system. After the if-clauses comes the keyword **then** followed by the **then-clauses**. The number of then-clauses is also only restricted by the available memory. The keywords if and then are written on separate lines to improve the legibility of the knowledge base. Again there can be any number of then-clauses which are separated from the next rule by at least one blank line. Due to the way that PESYS stores all the words found in the knowledge base there is no restriction on the case of any of the words in the knowledge base, including the keywords if and then.

iii. Advanced features of rules

Negation

The if-clauses of the rules may be used to ask the users questions about the domain. Clauses that contain the word **not**, however, can be very difficult to answer in practice, often causing many problems for the users. If the system asks the question "Is this true: you have not put the cat out?" should the users select Yes signifying "Yes the cat has been put out" or should Yes be selected to specify "Yes the cat has not been put out"? A far simpler question, from the users' point of view, is "Is this true: you have put the cat out?". PESYS therefore removes any occurrences of the word not in any of the clauses of a rule (this is performed by the PREPARE program) and keeps track of the number of times it occurs. If not is found once or, in general, an odd number of times then the clause is said to be **negated**. Zero or an even number of nots means that the clause is **unnegated**. The negated status of the clause is then stored separately in the .RLC file and is used by the system to determine how the users' response to the (unnegated) question should be handled, i.e. if the users answer Yes to a clause that was originally negated, then this is equivalent to denying the original clause. Thus answering Yes to "The cat has been put out" means that the original clause "The cat has not been put out" has been denied. This is a very simple step to implement in a computer based system yet it makes the system considerably easier to use in practice. There is a minor restriction in that it is not possible to accommodate abbreviations such as can't and hasn't within this method.

Goals

Many expert system shells have a series of goals or hypothesis associated with the rules in the knowledge base. The inference engine of the shell then tries to use the rules to arrive at one of these goals or hypothesis. However, by keeping the goals separate from the rules that they refer to, there is a strong possibility of problems of redundancy arising, particularly if a large scale application is developed. Consider the case where a new (goal) rule is added to the knowledge base. The then-clauses of this rule refer to a goal yet it is possible that the developers of the system will forget to add the associated new goal to the (separate) list of goals. If this happens the extra rule will never be used, or if it is used, will never be recognised as a goal. Similarly if a goal rule is deleted without removing the associated goal redundancy will occur and the system will try and arrive at that goal although its associated rule no longer exists.

PESYS overcomes this problem by associating the goal status of a then-clause directly with that clause. Any then-clauses which refer to goals are marked with the keyword **inform** and an associated **inform level** of 1. This marks the clause as being something that the users should be informed about and the level (1) tells the system that the clause is goal. This approach avoids all the problems of redundancy that arise when the goals are kept separate from the rules they are associated with. If a new rule is added with a then-clause that is a goal, this is marked directly in the rule. Similarly if this rule is deleted from the knowledge base, the associated goal is also deleted and it is therefore impossible for the list of rules to be different from the list of goals. The PREPARE program removes the informs and their levels from clauses of the rules and stores them separately in the .RLC file.

The use of inform statements and inform levels is used in other ways in a PESYS knowledge base as well. During an interaction with the system, the users may find it useful to be told certain pieces of information. These are normally `intermediate' results, possibly informing the users about the current state of the problem solving process, and whilst they are not goals they are still useful. These are implemented by attaching the keyword inform to these statements and giving them a level of 2. The keyword inform tells the expert system that the users should be informed about a particular piece of information and the level (2) tells it that this is simply a useful piece of information and once the users have seen it the system should continue to arrive at the final conclusion which is a statement marked with an inform level of 1.

c. ADVANCED RULE STRUCTURES

In the example rule shown above the if-clauses of the rule are combined using the logical connective **and**, i.e. the if-clauses are only true if all the if-clauses are true. In many applications it is also useful to combine clauses using the logical connective **or**. Clauses combined using the or connective require only one of the clauses to be true.

One method of implementing the or connective, found in many expert system shells such as Xi+, allows an object to have more than one possible value. For example, a ball may be one of a number of different colours as in: "the ball is red or white". This technique is only applicable for multiple values of one object and cannot cope with clauses such as "it is raining or someone pours a bucket of water over your head". In order to cope with this second type of combination of clauses, in PESYS all clauses combined using or are assumed to be separate clauses, so the first example would need to be rewritten as "the ball is red or the ball is white" and each clause is written on a separate line of the rule.

```
IF
it is raining
or someone throws a bucket of water over your head
```

Figure 5.4 - The use of the or connective in a PESYS rule

Or-clauses are stored in the expert system as a list and each if-clause is, in fact, a list of or-clauses. This is best understood using a diagram, **Figure 5.5**. In this diagram, the if-clauses are shown vertically, whilst the or-clauses are shown horizontally. For a rule to fire, each if-clause must be known to be true. Each if-clause is known to be true if one or-clause is known to be true.

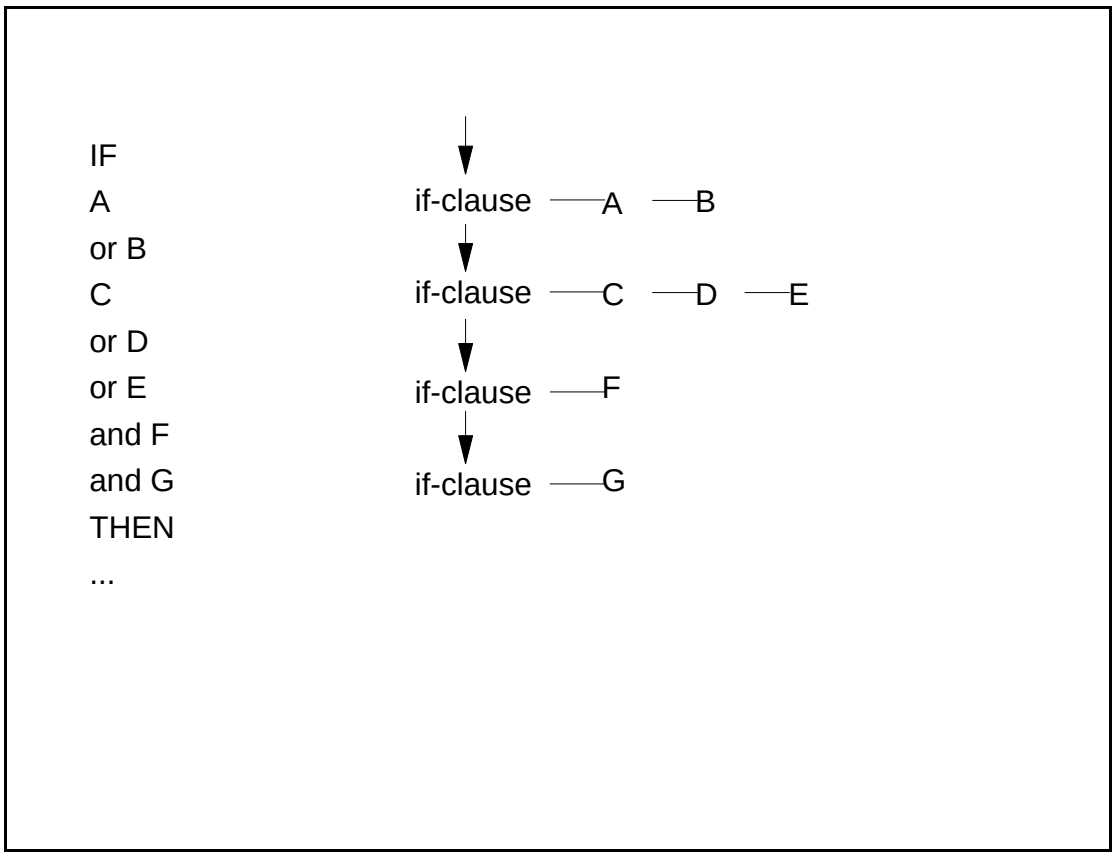


Figure 5.5 - A diagrammatic representation of or-clauses

Or-clauses are formed as the knowledge base is read in. The first clause that is read in is added to the start of an if-clause and the next clause is read in. If it begins with the keyword OR, then it is added to the end of the previous or-clauses, otherwise a new if-clause is created. This process continues until all the clauses of the rule have been read in. This also shows how clauses combined with and are implemented. Clauses that begin with an and, either explicitly or implicitly, do not begin with an or and are therefore stored as a new if-clause. The list of or-clauses therefore only contains one element which must be true for the if-clause to be true.

Atleast clauses

Designing the inference engine in such a way also allows the easy incorporation of another very useful feature into the structure of a knowledge base. The rules that have been described previously have either required all the if-clauses to be satisfied or, in the case of

or-clauses, one of each of the or-clauses to be satisfied. In some applications, however, a compromise is required. For example, an application may require a rule to fire if a combination of the clauses is known to be true. A careers selection knowledge base, for example, may list a number of qualities associated with each career, however it is unlikely that any person would have *all* the qualities for a particular job. It would therefore be useful if the system could see how many of the attributes a person had and if these were more than some preset level (perhaps 75%) the particular career could be suggested. In other cases, all possibilities need to be considered before a choice can be made. A car repair system that reported the first fault it came to and then halted would not be as useful as one that listed all the faults before stopping.

The **atleast** command is designed to provide this facility by allowing the developers of a knowledge base to specify a list of or-clauses all of which need to be tested but which only returns true if a certain number of them are true.

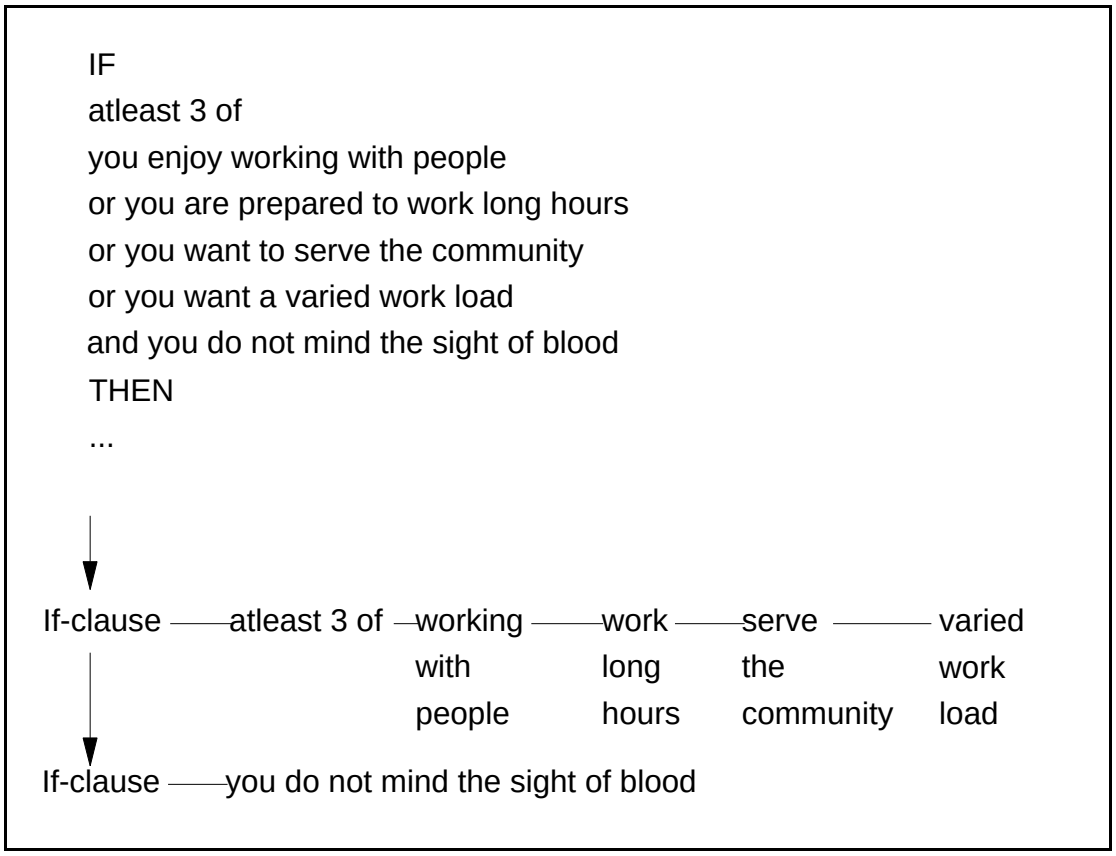


Figure 5.6 - An example rule using atleast clauses

In this case, the system will examine all the clauses found in the atleast statement but will not stop as soon as one clause is found to be true. Instead a count is kept of all the clauses that are found to be true and if this is found to be greater than or equal to the number

in the atleast statement then the if-clause is assumed to be true. Atleast clauses are stored in the same way as or-clauses, with the first or-clause being the keyword atleast and the **atleast level**. In all other ways atleast clauses are the same as or-clauses.

In the rules shown in **Figure 5.6** the system will examine the four clauses of the rule and the if-clauses that they are a part of will be true only if atleast 3 of them are true.

d. THE RUNTIME ENVIRONMENT

The program in the runtime environment is called PESYS and it is this program that the users interact with when solving a problem in a domain. The program itself is made up of a number of different components and these are shown in **Figure 5.7**. The programs in the development phase of the system are described in the user guide, Appendix III. The structure of the knowledge base has been described previously and the structure of the working memory is described in more detail in Appendix IV.

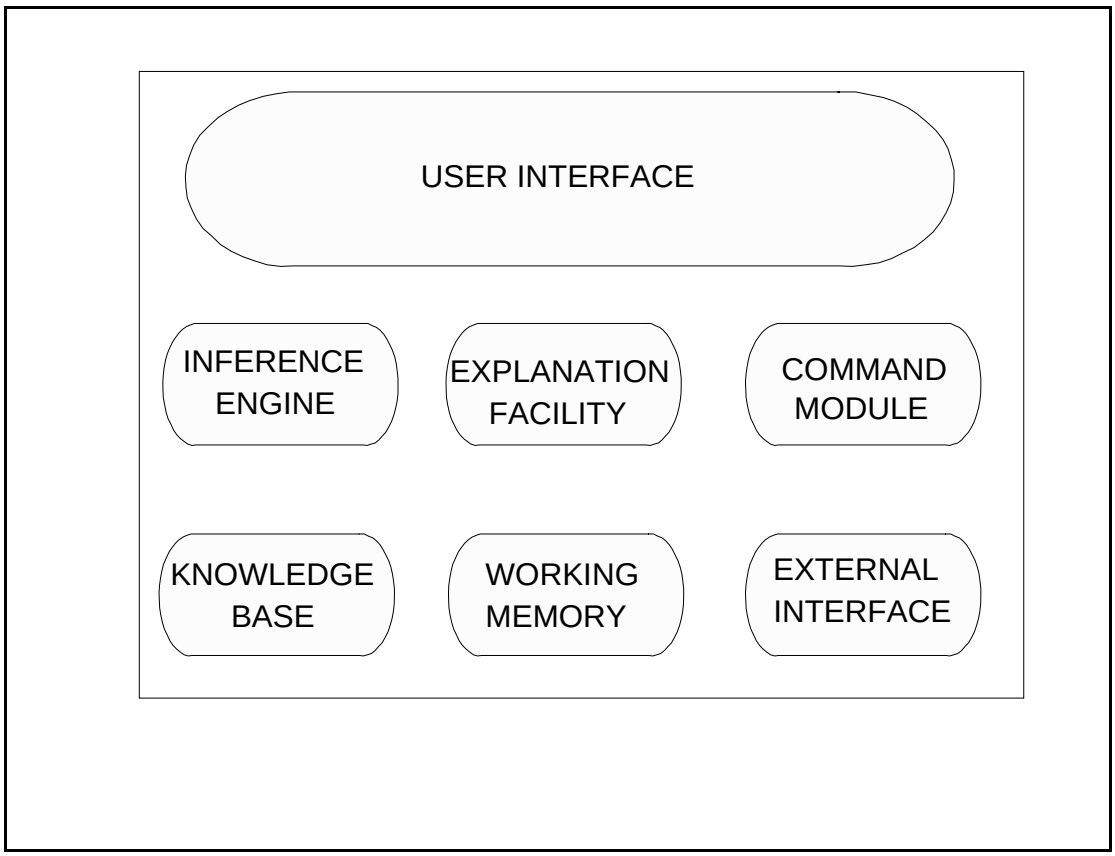


Figure 5.7 - The principal components of PESYS

i. The user interface

The user interface in PESYS is one of the largest components of the system and provides the users of the system with a simple and consistent front end to the functionality offered by the system. It is designed to be easy to use and easy to learn. It is through the user interface that the users answer questions, are informed of any results and perform what-if analyses. All these actions take place in special windows on the screen. The windows can be moved around the screen by the users (by simply pressing the **F4** function key and selecting the new position) if, for example, the default position obscures some other important information displayed on the screen. The system uses only two colours, normal and highlight, and they are specified by the developers in the configuration stage of the PREPARE program although they may be altered by the users at runtime. Only two colours are used since the package may be used with displays that have a limited colour capabilities (such as overhead projecting equipment) and by limiting the number of colours to two, it is easy to rapidly customise the screen displays to a particular piece of display hardware.

Inputting information

The most commonly used part of the user interface is concerned with asking the users questions and requesting information. PESYS has three main ways in which this can be done: through the use of YES/NO questions, through the entering of values and through the use of a `most likely' mechanism.

The simplest form of question is the YES/NO question. In this case, the users are presented with a clause and are asked to indicate whether it is true or not. This choice is done by moving a highlight to the appropriate value. If necessary, the users can also have a simple explanation of the question that has been presented.

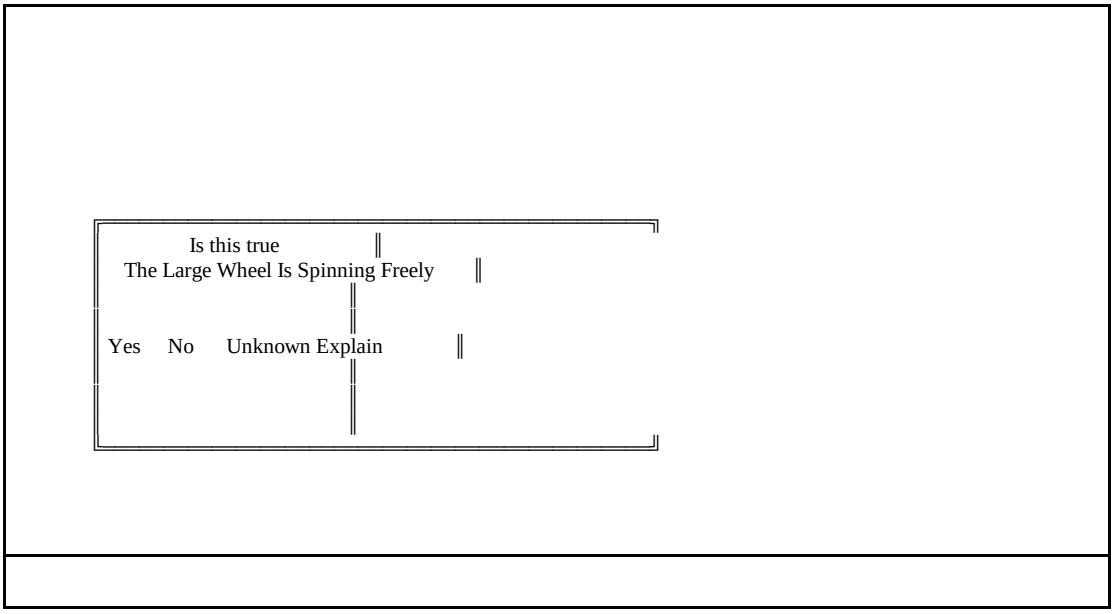


Figure 5.8 - A question in PESYS

In other cases, the users are requested to enter values of certain variables. In these cases a suitable prompt is displayed, describing what value is required and the users then enter the value requested.

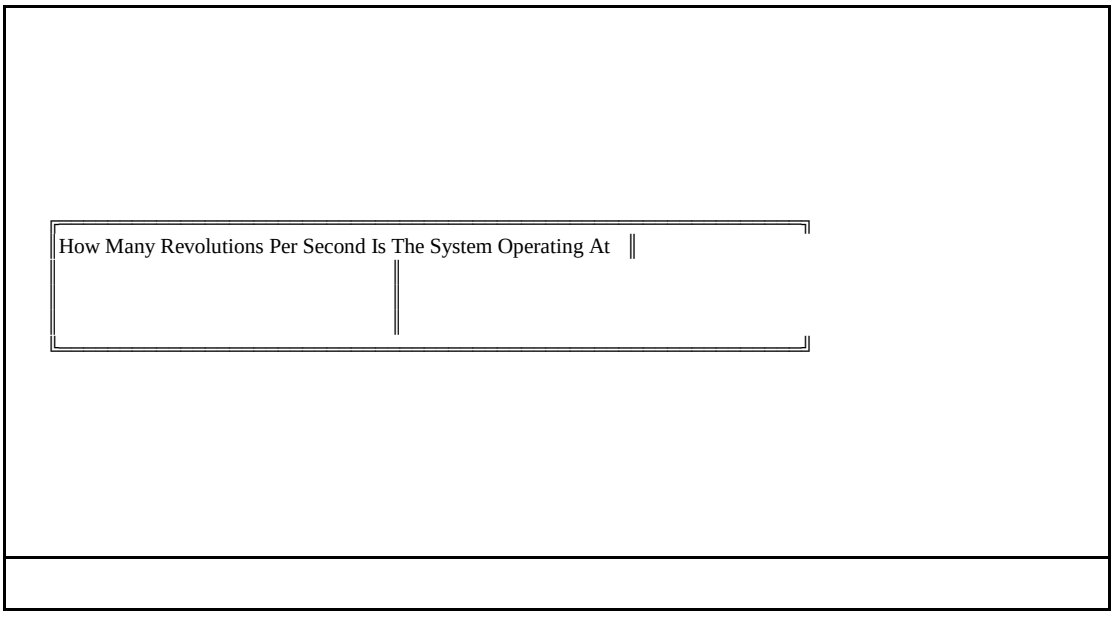


Figure 5.10 - A request in PESYS

The final form of input that the users may come across allows them to enter minimum and maximum values for a particular variable and they are then given a sliding scale which allows them to mark the most likely value. The entry of the minimum and maximum values is performed in the same way as requests, described above. The selection of the most likely value involves the users moving the marker ("X") to the point on the scale

that corresponds to the most likely value. The value that the marker corresponds to is given at the side of the window.

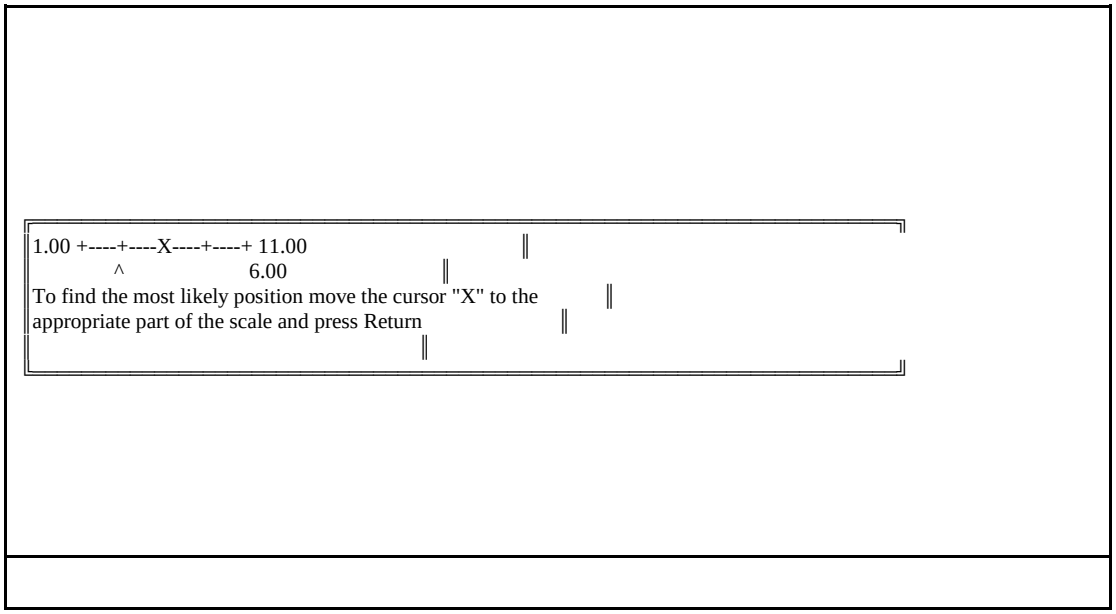


Figure 5.12 - Selecting the most likely value in PESYS

Informing the users of any information

The user interface also informs the users of any clauses that have an inform level of 1 or more. An inform level of 2 or more is simply some information that the users should be told about and is displayed on the screen until they press any key, signifying that they have acknowledged the information. The inference engine then continues with the interaction. Inform 1 statements are goals and these are displayed in a different screen for the users to see.

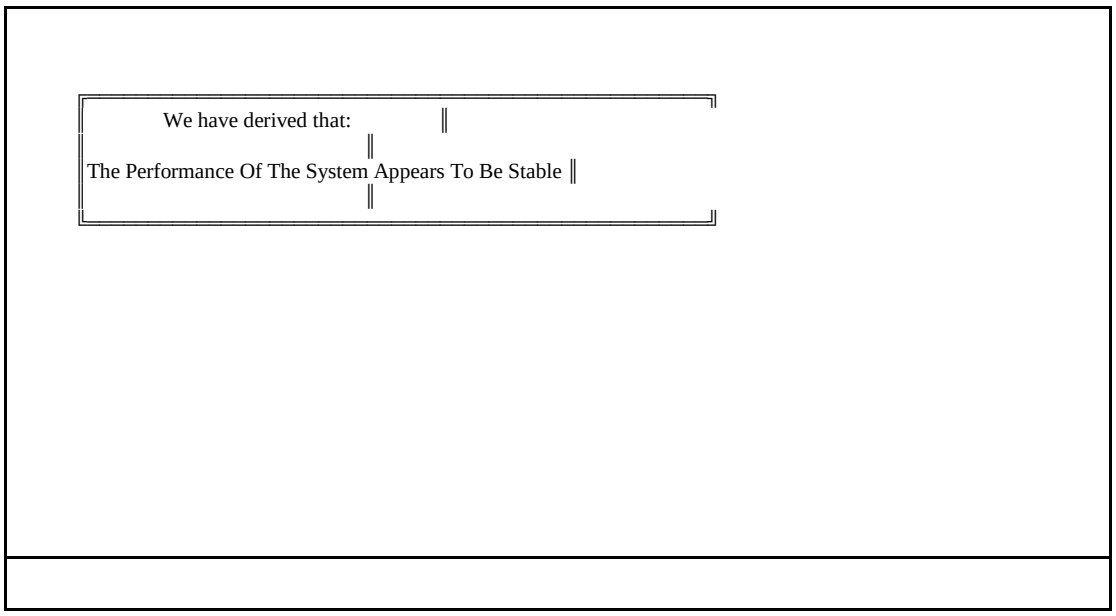


Figure 5.14 - An inform 2 statement in PESYS

Once a goal has been arrived at PESYS allows the users to perform a what-if analysis to see the effects of changing certain values. The altering of existing values for the what-if analysis is again handled by the user interface. The users are presented with a list of facts that the system has determined when working through the knowledge base. Those that are known to be true are marked with a tick √, those that are known to be false are marked with a cross X, and those that are unknown are not marked. If the users want to alter any of the clauses they can highlight the clause to be altered and press **RETURN**. For variables the previous value is displayed and they are asked to enter a new value, whilst for other clauses the truth of the clause cycles through the possible values. If there are more clauses than can be shown on one screen, the words "More to follow ..." appear at the end of the list and using the cursor keys it is possible to scroll the list to display these further elements. Finally, the users press the **CTRL** and **RETURN** keys to accept the changes or the **ESCAPE (Esc)** key to abandon the changes. The same process is performed when the users select the basic facts before performing a forward chain (see below).

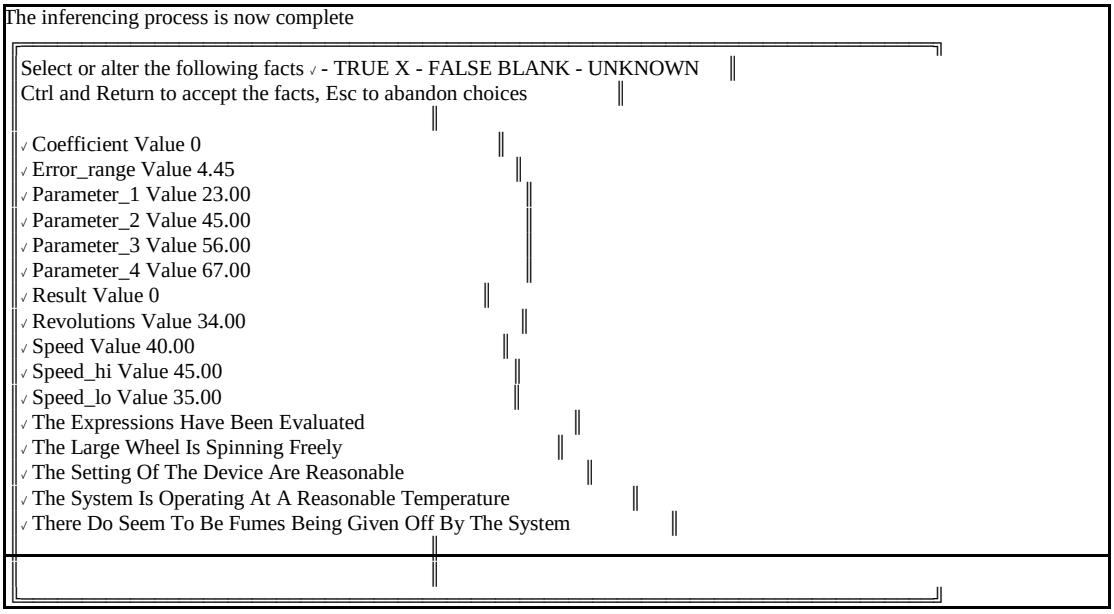


Figure 5.16 - The What-if facility in PESYS

ii. The inference engine

While the user interface is the part of the system that the users interact with, it is the inference engine that actually manipulates the knowledge base to determine which questions and requests are to be passed to the user interface and which results can be asserted. The PESYS system offers two forms of inferencing, **forward chaining** and **backward chaining**, and also supports specialised versions of these two approaches.

Forward chaining

Forward chaining, sometimes known as data driven inferencing, operates by taking a list of facts that are known at the start of the interaction and uses these to see which goals can be arrived at. A rule **fires** if all its if-clauses are known to be true, which in the case of PESYS means that one of the or-clauses for each if-clause is known to be true (see above). When the rule fires, the then-clauses of that rule are added to the list of known facts and the process continues until a rule is fired which has an inform 1 then-clause. In this case, the firing of the rule means that a goal has been arrived at and the system can stop. Alternatively the forward chaining process continues until all of the rules have been examined once without any of them firing. This means that it is not possible for any more rules to fire and

no more clauses will be added to the list of known facts and hence further inferencing has no effect. It is important that all the rules must have been examined once and not fired before forward chaining is abandoned since it is possible that the firing of the last rule may add a clause which is required for the first rule to fire.

Forward chaining, therefore requires that some facts are already known to the system otherwise no rules would ever fire and the forward chaining process would never arrive at any goals. Whilst it is possible to ask the users to enter any facts that they know to be true at the keyboard this is a rather cumbersome method and requires the users to use exactly the same descriptions of the facts as the developers of the application. PESYS offers a different approach to this problem. The if-clauses in the knowledge base rules are either found in the then-clauses of other rules or are not. Those if-clauses that are not found in the then-clauses of other rules are called **low-level facts** since they cannot be determined by firing rules and can only be asked of the users. When PESYS loads up a knowledge base it automatically creates a list of low-level facts and if the developers of the application configure the knowledge base appropriately, it is possible for the users to be presented with this list of low-level facts and for them to mark those that are known to be true or false and to enter them into the working memory of the system before the forward chaining takes place.

Backward chaining

Backward chaining, also known as goal driven inferencing, takes a different approach to arriving at goals. Forward chaining took the available data and found what goals could be arrived at from that data. In contrast backward chaining takes a particular goal and obtains the data that is necessary for that goal to be arrived at. In doing so, backward chaining provides a more focused approach to inferencing than forward chaining.

The backward chaining algorithm begins by creating a list of rules that contain inform 1 then-clauses. The inference engine then takes the first rule in this list. For the goal (inform 1 clause) to be known the rule must fire. The rule can only fire if all the if-clauses are known to be true. The system then examines each of the if-clauses, and hence each of the or-clauses, in turn. In the simplest form the if/or-clause is already known. If this is the case the system moves onto the next if/or-clause. If the clause is not already known, the system then checks to see if the clause can be arrived at by firing another rule. This is done by creating a list of all the rules that contain the if/or-clause in its then-clauses. If this list is

empty, then no rules exist which can fire and add the clause to the working memory and the only option left to the system is to ask the users directly. If rules do exist then each is examined to see if it would fire using the facts that are known at that time. If none fires immediately then the first rule in this list is taken and the entire inference procedure is repeated recursively. Once one of the rules has fired, the original if/or-clause is checked in the working memory of the system and is used for the previous rule. This process continues until either a rule with an inform 1 then-clause is fired or until no further rules can be fired.

Other forms of inferencing

In addition to having the two separate forms of inferencing, it is possible to combine them using **mixed chaining**. The particular form of mixed chaining implemented in PESYS has the inference engine perform a forward chain whenever any facts are added to the working memory by the backward chaining mechanism.

The final form of inferencing supported by PESYS is known as **pure forward chaining** and is concerned with **commands**. Commands in PESYS are executed and assumed to be true. If the system is performing a forward chain, however, this may result in spurious rule firings, i.e. rules firing simply because the commands in their if-clauses are executed. In practice, therefore, commands are not normally executed when performing a forward chain. Pure forward chaining, however, does allow commands to be executed.

The choice of which inference strategies to use is made by the developers of an application in the PREPARE program and is stored in the configuration file. The predefined choices can be altered by the users if necessary by selecting the configuration menu. This allows, amongst other things, the users to alter the choice of inference strategies used for a particular application.

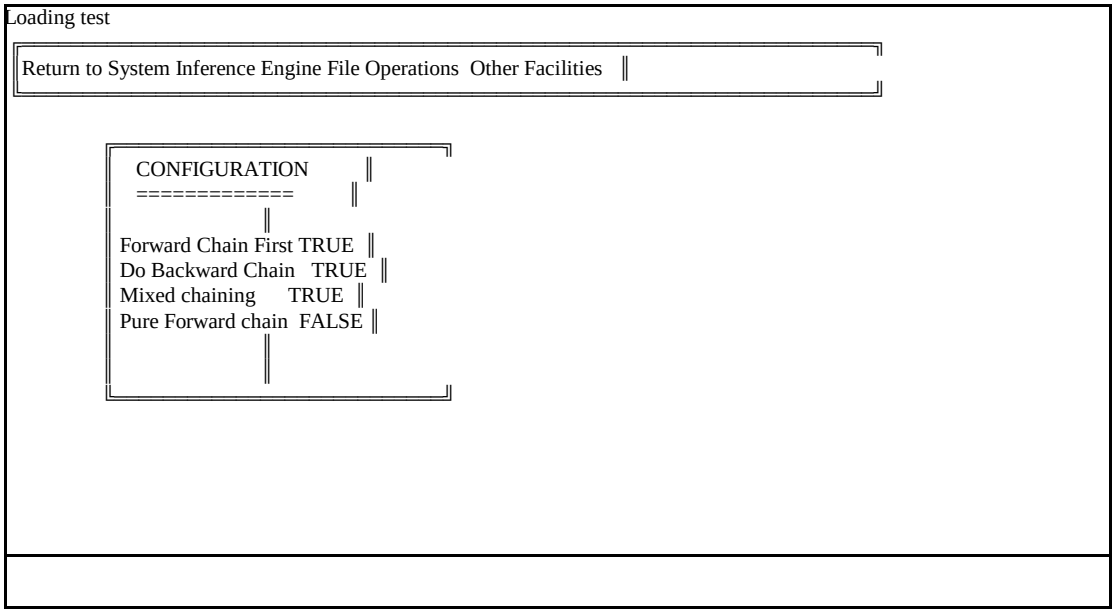


Figure 5.18 - The configuration menu in PESYS

iii. Commands in PESYS

The functionality available in the knowledge base is increased by making various commands available to the developers of applications. These commands are not intended to form a special procedural language and only offer mathematical calculations and the use of variables, input output control, external interfaces and the use of sub knowledge bases. The available commands are described in more detail in the user guide and will only be briefly outlined in this chapter.

PESYS supports elementary mathematical capabilities and allows the use of numeric and text variables. The command LET is used to assign the result of an expression to a variable. The expression can support the basic arithmetic operations and supports brackets to alter precedence. The values of expressions can be used in comparisons and these are treated like other rule clauses.

Considerable control over input and output is provided, with commands available to display information on the screen (PRINT, CLS, PRESS_KEY). It is also possible to input values from the keyboard or from a file using the ENTER, RE_ENTER and RANGE_OF commands. It is also possible to create and use data files through the use of commands such as OPEN_OUT, WRITE, WRITELN, CLOSE_OUT and OPEN_IN, READ, READLN, CLOSE_IN.

Links to external applications are supported through the commands DOS and DOS_E and sub-knowledge bases are supported by the command USE.

e. APPLICATIONS DEVELOPED USING PESYS

PESYS has been successfully used for three years in teaching students on the M.Sc. in the Analysis, Design and Management of Information Systems at the London School of Economics and Political Science about expert systems. It has also been used by another researcher in the department as part of her doctoral research. The following sections describes the main case studies that have been developed by other researchers using PESYS.

i. Effort estimation for software development

The process of software development is very complex and managers require accurate estimates of the effort involved in this process. The research undertaken by Levy (see, for example, Levy 1990) investigates effort estimation for the preliminary system design stage of a project life cycle.

The research considers the tasks that are to be performed during this stage of the project and uses these tasks to provide an estimate for the effort required. In addition to developing an effort estimation model the research is examining a number of actual case studies to examine the validity of the ideas presented.

PESYS is used as the main component of a software tool that can be used to demonstrate the concepts of the effort estimation process. The expert system is used in a number of stages including entering of data, the calculation of the effort and the presentation of the results.

The system has been successfully demonstrated to a number of organisations and the ease of data entry, presentation of results and explanation facilities have been favourably commented upon. The system has also been used to analyze questionnaire results in the same way.

ii. An expert system to assist in filing tax returns

PESYS was also used to develop an expert system that helped individuals file tax returns without having to involve accountants and lawyers (Whitley *et al.* 1989). The system was to be used in India where the tax laws are complicated yet the revenue generated is relatively low. Computers were being introduced to simplify this process and so an expert system was developed to see if it would be easier to use than other computer systems.

The application made use of many of the advanced features of PESYS described in this chapter in addition to some of those described in the next two chapters. The resulting system, although not perfect, was considerably more useful than other attempts to develop systems for this domain.

iii. Other applications developed using PESYS

In addition to these two major applications developed using PESYS a number of smaller applications have been developed by M.Sc. students over the past three years, each of which made use of certain features of the PESYS system.

A number of applications have examined the use of PESYS in conjunction with other pieces of software. The most common form of such a link has been with database applications. The first attempt at this allowed PESYS to access a database by making use of a compiled dBase III+ application. The database contained information about foreign exchange rates and the values obtained from the database were used by an expert system application.

PESYS was also used as an intelligent front end to a database application to assist in data entry. In this case, the expert system was called from the database application.

The use of PESYS from within a different application has also been utilised by a financial simulation model. In this case, the knowledge base is used to `reason about' the choice of investments.

PESYS was successfully used to implement the rules of cricket in an expert system and in an application designed to providing education so as to prevent collisions at sea. In both these cases, extensive use was made of the explanation facilities and readable knowledge base of PESYS.

Finally PESYS has also been used to identify Leptospires. In this application, the student investigated the extent to which an expert system should be used as a support tool for the users.

These projects are summarised in **Figure 5.20**.

An expert system application for	**Area of PESYS examined**
Foreign currency analysis	Linking an expert system to dBase III
The rules of cricket	Using legislation in an expert system
Preventing collisions at sea	The use of explanation facilities and multiple goals
Identification of Leptospires	The extent to which users should be able to co-operate with the expert system in problem solving
Intelligent data entry for a database	Links to database packages
Financial simulation modelling	Using an expert system as part of a simulation model

Figure 5.20 - PESYS applications

CHAPTER 6 - SOLUTIONS TO THE PROBLEMS OF THE KNOWLEDGE BASE

Chapter 3 introduced a number of examples of knowledge that can cause problems when expert systems are implemented in semi-formal domains. This chapter will describe and evaluate a number of additions to the PESYS system that attempt to overcome or minimise some of the problems raised in that chapter. The first method, which proved to unsuccessful and was not used in the final system, attempts to combine the use of natural language rule clauses with a knowledge representation technique based on the underlying ideas expressed in the clauses. The second solution attempts to assist the users in forming the appropriate interpretation of the clauses used in the knowledge base. This method is extended through the use of non-linear documents as a third solution to the problems raised. Non-linear documents provide a semi-formal approach to problems of differing interpretations since they allow the users of the system to relate their own interpretations of a situation to those used by the designers of the knowledge base. In doing so they can determine whether or not their own situation lies within the boundaries of the knowledge base.

a. A NATURAL LANGUAGE PATTERN MATCHING SYSTEM

Many of the problems of knowledge in semi-formal domains that were discussed in Chapter 3 arose because of the prominence placed on natural language communication between the experts, knowledge engineers and users of the expert system. Other means of communication, such as the use of diagrams may help overcome these problems, however the dominant means of communication in the development and use of expert systems is still natural language and therefore this chapter will concentrate on the problems that relate to this area. It will also provide assistance with some of the problems of tacit knowledge described in Chapter 3.

It can be argued that many of the problems of misinterpretation in semi-formal domains are introduced when attempts are made to represent the linguistically expressed knowledge of the human experts in some other formal representation. One way of overcoming this problem, therefore, would be to keep the knowledge in the knowledge base in a natural language format. In the case of the production rules found in PESYS this means that the if-clauses and then-clauses of the rule are simply natural language clauses (plus the predefined commands available in PESYS).

Doing this may, however, impose considerable restrictions on the developers of an application. The matching algorithm that, for example, is used to compare the if-clauses of a rule with the then-clauses of other rules, will only operate if the two clauses are exactly the same and this forces the developers of an application to use exactly the same wording and spelling if clauses are to match. In many cases a difference of wording will arise depending on whether the clause is found amongst the if-clauses or then-clauses of a rule.

IF the machine was still warm THEN ....

and

IF ... THEN the machine is still warm

The problem becomes more acute when different rules are entered at different periods of time and are heightened when different developers are involved in writing rules for a particular knowledge base. These problems suggest that it is very unlikely that the same phrasing will be used for all occurrences of a `particular' clause and therefore some solution must be developed to overcome this problem.

Consider the clauses "An information system could be used to control the distribution of resources" and "The distribution of resources could be controlled by an information system". Most people who examine these two clauses will state that they are basically the same because they both refer to the same `idea' with identical parameters. One solution to the problem of matching clauses would be to represent the clauses in terms of their underlying idea and use this representation in the matching process. The original rules could still be written in natural language form and the PREPARE program would convert each of the clauses into this new form, storing the results in the .RLC file. This coded version of the knowledge base is likely to be more compact than the original and will therefore operate more efficiently.

i. A method to find the underlying idea in a clause

Consider the three clauses shown in **Figure 6.1**.

The computer is switched on
Someone has switched on the computer
The television has been switched on

Figure 6.1 - Three clauses about switching on

The first two clauses have a very similar meaning (the extra information provided in the second clause is unlikely to be significant to the operation of the expert system) and all three clauses refer to the idea of something being switched on. In the first two clauses it is a computer that is switched on, in the third it is a television. The underlying idea could be taken to be the device (computer or television). However, as will be shown below, there are advantages to associating the idea with the main `action', namely switching on. The terms `idea' and `action' may limit the identification of what is being conveyed by a particular clause and hence the term **relation** will be used to instead. Thus in a clause such as "The patient's age is greater than 21" the relation is taken to be "greater-than".

In the previous examples, therefore, the relation would be "switched-on". However it is still necessary to distinguish between a television that is switched on and a computer that is switched on. This is done through the use of **parameters**. Every relation has associated parameters which determine the kind of thing that the relation is concerned with. In the case of "switched-on" only one parameter is needed, namely the thing that has been switched on. In the examples, this thing is either a computer or a television; more generally it might be an electrical device.

Defining relations

If this method is to be implemented in an expert system it is necessary for the computer to identify the relation and the associated parameters. Current computer based systems have no inherent `intelligence' and implementations of natural language `understanding', in anything other than very restricted domains, are highly prone to error.

This means that the computer, per se, has no way of identifying the relation and parameters in a particular clause. The only solution available, using current techniques, is for the system to be explicitly told about such relations. The simplest way to do this, in a convenient, computable form, would be to declare that

Switched-on Kind-of Relation

i.e., "switched on" is a kind of relation. To identify the relation in a clause, the system would simply examine each word in the clause and determine whether it was a kind of relation, i.e. it would look up the word in the knowledge base to see if it was marked as a kind of relation. If the word was not, the system would move on to the next word and this process would continue until a relation was found. Once the relation was found, the word would be marked as being "in use" and the system would attempt to find the parameters for the relation.

Defining parameters

The parameters for the relation need to be defined in a similar manner to the relation, however there must be some way of distinguishing between the parameters and incidental words found in the clause. One common method used in artificial intelligence work is to give the parameters a `type'. In the previous examples, the type of the parameter was electrical device and so the parameter for switched on would be declared as

Switched-on Needs Electrical_device

i.e. "switched-on" needs a parameter which is an electrical device. It is possible to list the types of certain words in a similar manner to the relation

Computer Kind-of Electrical_device
Television Kind-of Electrical_device

Thus to identify the parameters of a particular relation the system will simply examine each word in the clause that has not been marked as being "in use" and will check to

see if it is a kind-of electrical device. This process is then repeated until all the parameters for the relation have been identified.

Some problems

One problem that immediately becomes apparent arises when a relation has a number of parameters of the same type. For example, the relation "likes" may have two parameters, both of type person. The method, as described, would therefore take the clause "John likes Mary" and, whilst it would correctly return the relation as "Likes", would incorrectly return the parameters as John and John. This occurs because the system simply begins at the start of the clause when finding the next parameter and John is the first thing which is a kind of person that it comes across on both occasions. This problem can easily be overcome by marking as in use any parameters found and hence the same word will not be considered for two different parameters and therefore the system will return the correct parameters, namely John and Mary.

Another practical limitation of the current method is that it only allows a relation to have one set of parameters. In practice this is unlikely to be true and so imposes an artificial limit on the operation of the system. It would be more natural for the system to allow many sets of parameters for a particular relation. Each set of parameters is therefore examined until values can be found for all of them. Thus the clause, "John likes rice-pudding", would require the declarations in **Figure 6.2** to be made.

```
Likes kind-of relation
Likes needs person person
John kind-of person
Rice-pudding kind-of food
Likes needs person food
```

Figure 6.2 - More sets of parameters for the relation Likes

In this example, the system will try the first set of parameters (person person) and will find one of them (John), however a second person will not be found. Since all the parameters have not been found, the system will then move onto the next set of parameters (person food) for the relation likes, which in this case would be satisfied.

This automatic method of converting the clauses into a relation and parameters can be implemented in the PREPARE stage of the development process and the clauses in the .RLC file will be made up of the relation and its parameters rather than the original full text clauses. This means that the comparison of clauses will be much more efficient, both because it allows differently worded clauses to be compared and also because significantly fewer words will normally need to be compared, see **Figure 6.3**.

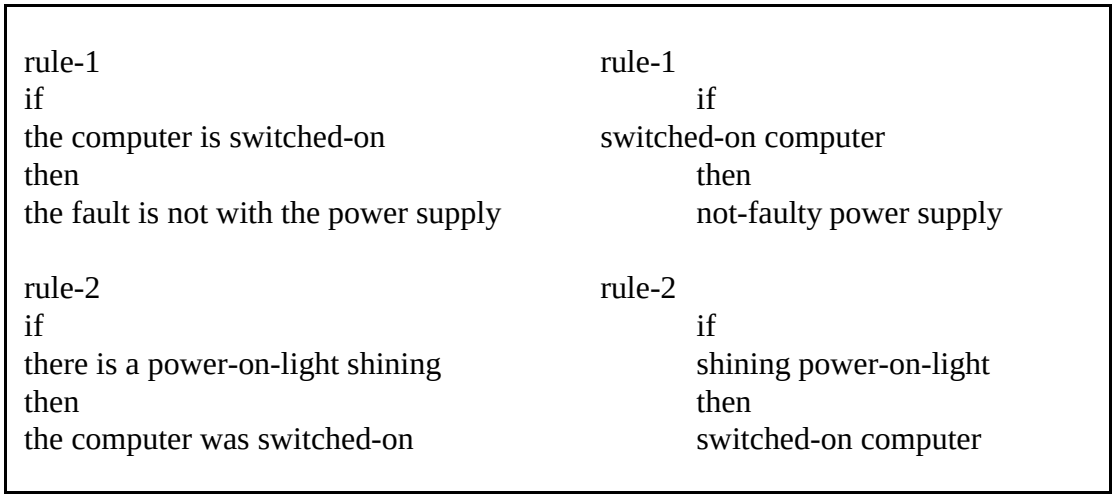


Figure 6.3 - Full text and encoded relations

ii. Inheritance

In the examples discussed so far, the actual parameters have had the same types as the formal parameters; John was a person and rice-pudding was a kind of food. In many cases, however, this direct relationship will not exist; the actual parameter will be an object that has a particular type, which is itself an example of a more general type which matches the formal parameter, i.e. rice pudding would be an example of a pudding, which is a kind of food. Artificial intelligence researchers have developed **inheritance** mechanisms to deal with such situations whereby an object can inherit properties from more general objects (Winston 1984).

Using some form of inheritance has a number of useful computational features. Firstly it is considerably more efficient in terms of storage space. Consider an expert system that stores information about a number of people. Instead of storing all the attributes that people can have with each person, they only need to be stored once as a `generic' person.

Individual people can then inherit these characteristics from the generic human. Inheritance can be implemented in a manner consistent with the previous examples:

John kind-of male-human
Male-human kind-of person

When the system is searching for a person and finds John, it will determine that John is a male-human. However, rather than stopping at this point, it will continue and use the data from male-humans to determine that John is also a person. By implementing the inheritance in this way, no limit is imposed on the number of stages involved in the inheritance process.

This is performed in the system by obtaining a list of all the things that John is a kind of from the knowledge base of the system. Each of these items is examined in turn and if it is not of the required type then it too is examined further. This process continues until either the correct parameter has been found or there are no more paths to be considered.

Some problems with inheritance

In many implementations of inheritance problems arise when more than one path can be followed within the inheritance. In these cases a decision must be made as to which path should be followed irrespective of the domain concerned. An example of this problem is based on the former American President Richard Nixon. Nixon is a Quaker and Quakers are pacifists. Therefore an inheritance system which has information about Nixon would, by inheriting through his being a Quaker, determine that he would not be prepared to fight for his country. However Nixon is also a Republican and Republicans believe that certain things, such as one's country, are worth fighting for and hence inheritance through the fact that Nixon is a Republican would determine that he would be prepared to fight for his country. The choice of inheritance path is therefore crucial for deciding whether the system returns that Nixon would or would not fight for his country, see **Figure 6.4**.

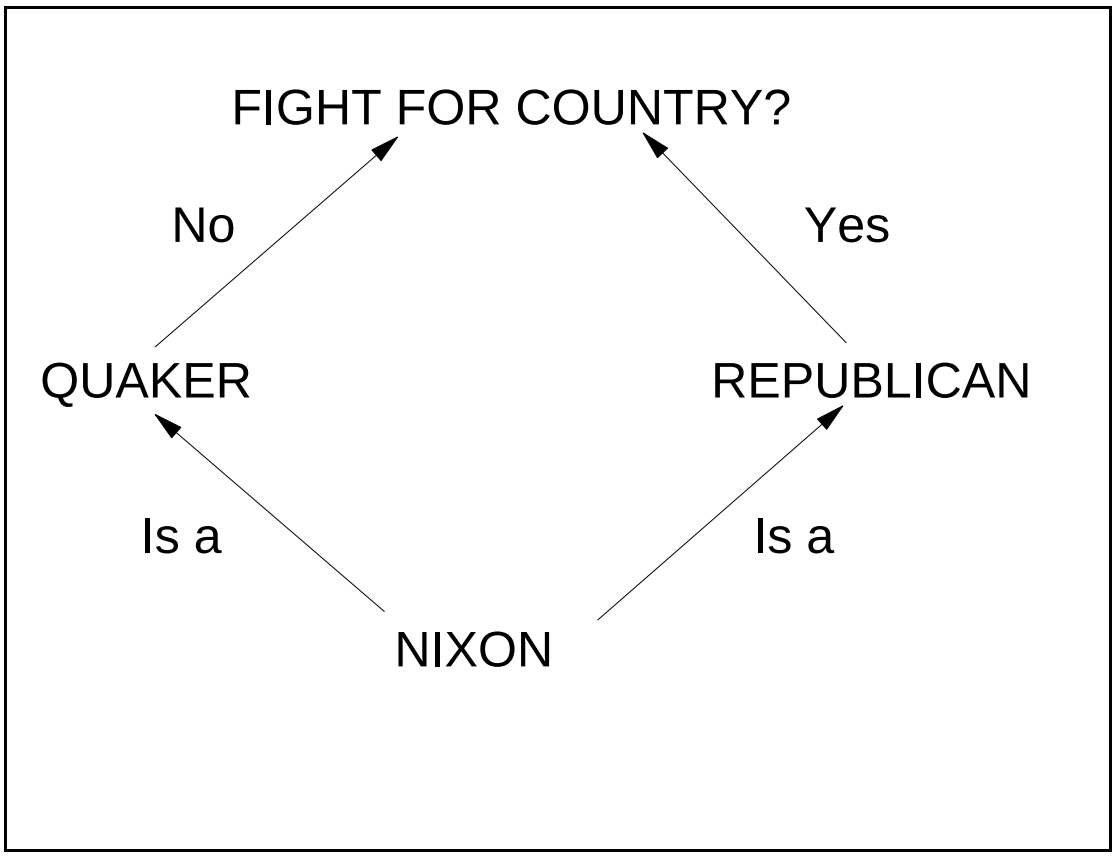


Figure 6.4 - The `Nixon diamond'

A second problem found in inheritance mechanisms arises when default values are used. Default values are often inherited from `generic' objects and are used when no specific values can be found. Thus a generic computer may have a screen, a keyboard and a disk drive and it is not necessary to specify that every particular computer has a disk drive since this can be inherited as a default value from the generic computer. If a particular computer does not have a disk drive, then this is stored as a fact associated with the particular computer and the default values are not used. This general mechanism causes some problems, however, when a particular object has attributes in addition to the default values. For example, an Apple Macintosh has a mouse as well as a keyboard input device. If the mouse is stored at the level of the Apple, the standard inheritance mechanism will not determine that the Apple Macintosh has a keyboard.

To a large extent, these problems are avoided in the PESYS system since only the type of a parameter is searched for, not the attributes of that type. In general, the problems of default inheritance and multiple inheritance paths only lead to problems when the attributes of these types are considered, for example, Nixon being both a Republican and a Quaker only

causes problems when inheriting whether he is prepared to fight for his country or not and the inheritance mechanism operates successfully when simply determining whether he is a Quaker or a Republican.

iii. Statements presented to the users

Although the converted clauses are now much shorter, leading to faster comparisons and more compact storage, they are not particularly easy to read. If the users are presented with the question "Is this true: Switched-on John Computer", they are unlikely to have a clear indication of what the question actually means and the problems of misinterpretation, already present in semi-formal domains, will be significantly worse.

Since the relation based form of the clause stores all the `significant' parameters of each relation it is possible to `recreate' a more understandable version of the clause. This can be done by defining a general pattern for the relation and parameters. For example, a pattern for the switched-on relation with a person and a machine could be "Person switched the machine on". When the pattern is used the actual parameters replace the formal parameters. Thus the relation "Switched-on John Computer" would become "John switched the computer on". Note that in the `recreated' version of the clause, the actual relation is not (necessarily) displayed.

The pattern for a relation can be represented in a similar way to the rest of the data, with the relation name being followed by the keyword and then the translation pattern with the parameter types declared:

Switched-on Translates Person Switched The Machine On

and it is possible to define different patterns for different sets of parameters associated with a relation. In a similar manner to finding the appropriate set of parameters, the system examines each translation pattern until it finds one where all the parameters are present in the pattern.

This method of recreating clauses is easily adapted to deal with questions which ask the users to enter a particular value for the clause, rather than whether the whole clause is true or false. For example, the system may ask "What thing did John switch on?" and then

present a list of possible answers, including a television and a computer. The users can then choose those items that John switched on.

This can be implemented by using a pattern which includes the type of variable to be varied. In the previous question the pattern was "What thing did person switch on ? machine" where the last word indicates the item that varies. When the file is PREPAREd a list of things that inherit the type "machine" can be created (since this is a computationally intensive task) and this list can be used whenever the question is displayed.

iv. Implicit relations

Clauses such as "The ball is red" cannot easily be incorporated into the method as it has been described. The problem begins when trying to find a relation in the clause. The only obvious relation is the word "is". Selecting this as the relation requires the definition of a set of parameters associated with the relation. One possible set of parameters would be a spherical object and a colour. However, such an approach would lead to many sets of parameters for the relation. The other possibility would be to define the parameters as something and something. This approach removes all the benefits of the use of types for the parameters and does not distinguish between the word "the" (which is a kind of determiner) and the word "ball" (which is something to do with the relation).

Consider the clause "the ball is a red colour". Most people would agree that it conveys the same idea as "the ball is red". Why is this so? The answer seems to be related to the fact that "red" is a kind-of colour and "colour" is a kind-of relation. Thus the word red seems to imply the relation colour. Thus the general inheritance principles discussed previously could be adapted to work with relations as well as parameters.

The implied relation can be declared in the normal way:

Red Kind-of Implies
Red Implies Colour
Colour Kind-of Relation

The use of implicit relations means that the designers of an expert system application are no longer required to use the same tense / phrases to describe the relations. Synonym relations are possible and this is particularly useful for dealing with the different forms of

verbs. For example, the relations creates, created and creating are all variations of the relation create and hence imply the relation create. By making use of the implied relation it is only necessary to define one set of parameters and translations.

v. `Learning' about relations

For the system to operate successfully, it is necessary for the developers to provide information about the various relations, parameters and translations that are used in the clauses. This information should be provided in as simple and automatic a way as possible and can best be implemented by making use of the method used by the system to convert the clauses into this relation based form.

The method begins by taking a clause and searching for a relation. Therefore, if no relation is found, the system needs to be told about the relation. This can either be done explicitly, by stating that something is a relation, or implicitly, by showing how an inheritance link arrives at a relation. The actual forming of the statements such as "... kind-of relation" can be automatically performed once the name of the relation has been stated. Whenever a new relation is specified, at least one set of parameters should also be entered by the developers of the application.

After the relation has been successfully identified the system moves on to finding the parameters of the relation. There are two ways in which this can fail. Either the existing sets of parameters are not suitable for the given clause, in which case new parameters need to be specified, or the types of things in the clause have not been fully described in which case extra inheritance information needs to be entered.

Finally, the system can prompt the developers of the application for any translation patterns for the relations, using either the statement or question form of the clause. The system can then also automatically generate lists of things which inherit the type machine, for example.

vi. A case study

A small demonstration knowledge base was created to test the usefulness of this method. A domain was chosen where a substantial part of the knowledge base was described

in natural language clauses since, as was discussed in Chapter 3, descriptive definitions in natural language are likely to be a major cause of problems in semi-formal domains. In this particular knowledge base, no commands were used and therefore all the clauses were text based clauses. Further details of the knowledge base are given in Appendix V.

The knowledge base was designed to provide advice to someone considering whether or not to use information systems to obtain a competitive advantage in a particular business situation. The underlying knowledge used was taken from the article "The information system as a competitive weapon" (Ives and Learmonth 1984).

Ives and Learmonth define an application of information systems technology as having a strategic role if "it changes a firm's product or the way a firm competes in an industry" (p. 1193). They discuss the work of Porter who identified five major competitive forces that affect a firm:

The threat of new entrants
The intensity of rivalry among existing competitors
Pressure from substitute products
The bargaining power of buyers
The bargaining power of suppliers

Figure 6.5 - Porter's major competitive forces

Thus to remain competitive the firms can aim for any of three major goals. They can aim for overall cost leadership, whereby they use information systems to reduce the cost of their product below that offered by other suppliers. They can use information systems to differentiate their product from those supplied by their competitors, perhaps by offering individual design services. Finally the firm can remain competitive by focusing on a particular market niche and information systems can also be used to help with this task. Information systems technology can also be used to prevent the threat of new entrants and reduce the intensity of rivalry amongst existing competitors.

The model that Ives and Learmonth develop is based on the **customer resource life cycle**, that is the life cycle of the customers' interactions with a firm from the initial selection of suppliers, through the acquisition and use of the product to the final after sales service. By concentrating on the customer throughout the life cycle, special attention can be paid to enhancing customer services which will have a direct effect on customer loyalty and will differentiate the firm from its competitors.

As well as describing the model in some detail Ives and Learmonth also provide a number of examples of actual information systems in each of the thirteen stages of their model. The knowledge base is designed to cover all thirteen possible areas where the introduction of information systems technology may offer the company a competitive advantage and includes a number of actual examples for each stage to provide guidance to the user. Thus rather than asking "Can an information system be used to determine how much of a resource is required?" the users are asked if a condition similar to one of the examples exists in the organisation. If a similar example exists then information systems technology is potentially useful for that stage of the customer resource life cycle.

vii. A discussion of the problems with the method

Although the system was implemented and operated satisfactorily, a number of problems were noticed which suggested that there might be fundamental difficulties associated with this method, difficulties that would prevent the adoption of the method in the final system. In particular, these difficulties were associated with defining parameters for relations and defining the types of parameters associated with various relations.

Problems identifying parameters

In most cases the relation in the clauses was identified without too many problems, and it was only when the parameters were identified that problems arose. An example of this arose from the clause "The resources required can be specified". In this case the relation is easily identified as "specified", but problems arise when defining the parameter for the relation. In this case, the parameter is something to do with the "resources required", however it is difficult to determine what type "resources" are. In this particular case, they were described as being of type "object".

Ideally the parameters chosen should be as general as possible, allowing a wide range of cases to be covered by a single relation. In many cases, however, there are few clues as to suitable parameters and the process often degenerates into an ad hoc process that depends entirely on the particular clause being examined. Moreover, the choice of parameters also becomes dependant on who is making the decision and the time that the decision is being

made. Effectively, the problem of subjective definitions is arising as the knowledge base designers choose their own parameters according to their own backgrounds.

A possible solution based on semantics

Stamper *et al.* (1987) describe work that has been undertaken to try and arrive at a **semantic normal form**. They argue that the various normal forms that exist in relational databases approach the problem of normalisation from the point of view of the existing database structure, not from the underlying domain that the database is trying to model. For example, a database that is in third normal form has been designed so that valuable information, such as the address of a warehouse, is not lost when certain data items are removed. For example, if all the items in the warehouse are removed, the address of the warehouse should still be accessible. Similarly the updating of information should be performed without the risk of redundancy. Thus the telephone number of the warehouse would ideally only be stored once in the database, so that only one field needs to be altered if the phone number changes.

Whilst this approach is very useful for dealing with data once it has entered the database, it provides no way of dealing with changes that occur in the underlying domain that the database is trying to model. For example, the database may still be able to obtain the address of the warehouse when there are no items stored in the warehouse, however it is not able to cope with a change in the organisation whereby parts are no longer stored in the warehouse but are instead delivered directly from third party suppliers on the basis of a request from the company.

Stamper *et al.* propose the foundations for a **semantic normal form** which attempts to make explicit the underlying semantics of the problem domain. The methods used to perform this task are semantic analysis and ontology charting which are described in more detail by Backhouse and Liebenau (1990). **Semantic analysis** involves examining a textual description of a particular organisation or process and analyzing the main terms used. These terms are classified according to the role that they play in the description, whether they describe things, conditions or states. Once the basic terms have been obtained and classified they are presented in an **ontology chart**. The ontology chart shows what things must exist before other things can exist. This necessity is represented by having things on the right of the diagram ontologically dependent on those to their left.

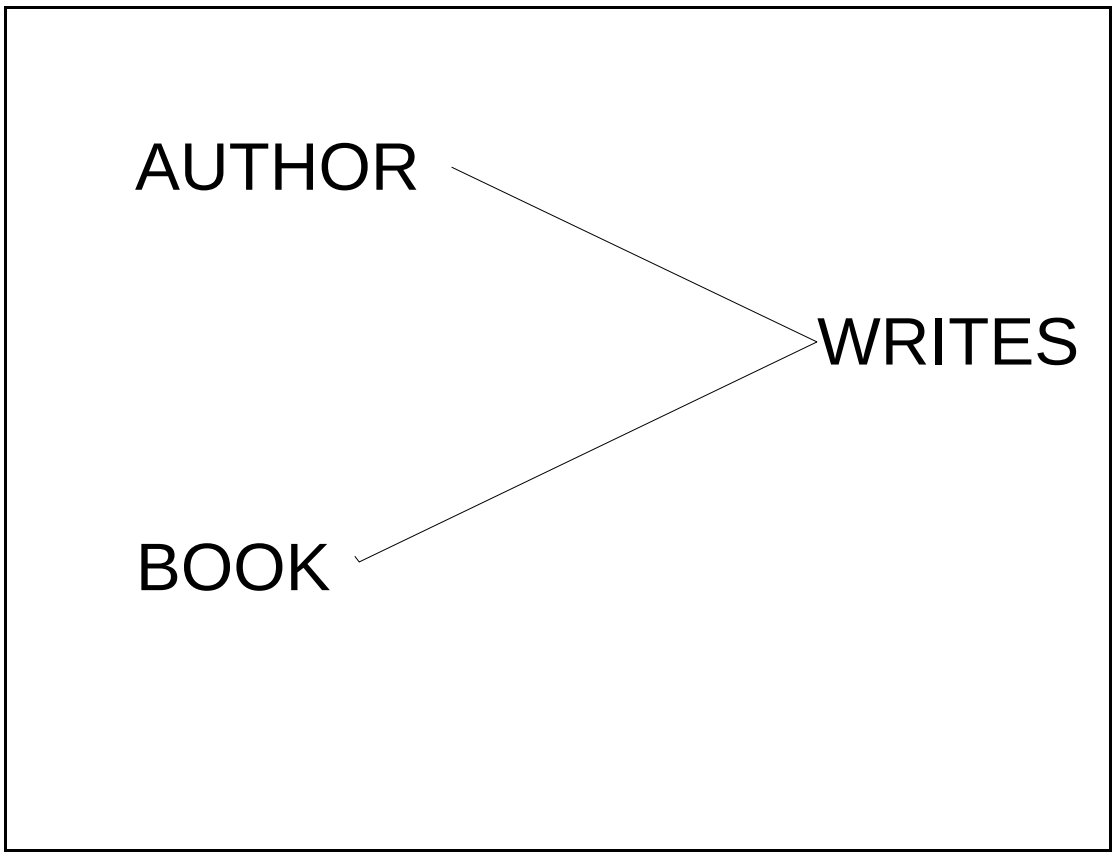


Figure 6.6 - A sample ontology chart for books, authors and writing

Thus, for example, a book does not depend ontologically on its author, it can exist for many years after the death of the author, after the author does not `exist' in any real sense. However, the "writing" of a book depends on both the author and the book. The book cannot be written without the author, see **Figure 6.6**. In the warehouse example, the delivery of parts depends on the parts existing in a warehouse, but this warehouse does not have to be owned by the company.

Ontology charting and semantic analysis may well prove to be useful in the development of "stable" expert systems and further research in this area is needed. However at the time when the problems of defining parameters were first noted research in semantic analysis was not sufficiently well advanced so that it could be used in this thesis. Furthermore, there were other problems associated with finding parameters that would still cause problems, even if the knowledge base was in a semantic normal form.

Problems with type hierarchies

Defining type hierarchies in the development of the relations also caused considerable problems. It was found that in many cases things were simply given the type "object" as no more descriptive form could be found. This causes a number of problems. In particular it removes the potential advantages associated with a strong typing of parameters. If everything is called an object then no distinction can be made between "objects which have motors in them" and "objects which can operate on abstract data". In those cases where distinctions were attempted, they tended to result in ad-hoc combinations of terms, carefully tied together using underlines. In the previous example, the clause "a round_the_clock order_entry_system has been created" lead to round_the_clock being described as an object. On a more practical level, having everything classified under a single type increases the potential for the system to chose the `wrong' parameter when attempting to convert a particular clause. This is particularly important when inheritance is used - everything may eventually become an `object' through inheritance.

Synthetic and organic problems

This problem of defining types runs counter to more conventional work in expert systems. The LEONARDO expert system shell, for example, includes frames and inheritance in its basic knowledge representation format yet there are few problems reported finding appropriate types for the inheritance mechanism. Did the problems with the example application arise because of the particular implementation of the domain chosen, or were they more fundamental?

One possible explanation relates to the semi-formal nature of the domain chosen. If the knowledge base is written for a domain such as the repair of a mechanical device, then problems with the inheritance of types are less likely to arise because some form of hierarchy already exists in the device being repaired, in essence the problem is **synthetic**. The designers of the device will have decided that, for example, the large wheel would be part of the drive mechanism, whilst the array of lights is part of the display panel, which is itself part of the control mechanism. The process of designing the device has lead to a prescriptive definition of the components and their interrelationships. This prescriptive definition gives each component a place in a prescribed hierarchy. Thus a wheel is part of the drive mechanism because that is what it was designed for and, although there may be problems

associated with identifying the hierarchy created by the designers of the device, these problems can be overcome.

If, however, the clauses describe social `creations' then they are likely to be descriptively defined and there simply may be no underlying hierarchies that can be utilised, the problems are **organic**. The clauses therefore describe things that were arrived at through a process of social interaction between many different actors, operating with different beliefs, desires and intentions and hence there is no purposeful, central principle which underlies the process; there are no `designers'.

b. PROVIDING ASSISTANCE WITH INTERPRETATIONS

i. The use of prompts

After careful consideration of the limitations inherent in attempts to develop this method for semi-formal domains, the decision was taken not to use the approach in later versions of the PESYS system. Instead, it was decided that the clauses of the rules would be kept in their full natural language form and attempts would not be made to try and formalise them in any way. The problems of writing the `same' clauses in different ways were accepted and it was decided that the developers would have to ensure that no mismatches arose. It is possible to support the developers of an application by providing an editor that is designed to make the repeating of certain clauses as simple as possible.

Despite the failed attempts at finding some representation for the knowledge in semi-formal domains, the problem of conveying the appropriate interpretation to the users remains. As the discussion on descriptive and subjective definitions showed, even a text clause that has not been formalised in any way is liable to misinterpretation. The clause may also attempt to describe the use of tacit knowledge or tacit skills. The system must therefore provide some way of conveying to the users the particular interpretation used by the system. This extra information should allow them to `understand' the question and provide an appropriate response.

A very simple, but effective solution to this problem involves using prompts. These prompts would be associated with individual clauses and would be displayed for the users if they require assistance. The prompt could contain a more detailed explanation of the question posed, it might provide some indication as to which interpretation was used, it may

suggest why that interpretation was chosen over other possible ones or it may be used to attempt to convey the use of tacit knowledge.

Attaching prompts to the clauses in a knowledge base in PESYS is a relatively straight forward process. All that is required is for the developers to include the keyword **prompt** anywhere in the clause. On the next line, the name of the prompt is specified. Problems of redundancy and retyping are minimised by using identifying names for the prompts in the knowledge base and using these names to display the full text of the prompts. This means that if the wording of the prompt needs to be altered, this can be done easily and only needs to be done once. Moreover, the text of the prompt itself only needs to be entered once.

The PREPARE program examines the rules and, in addition to the housekeeping tasks described earlier, removes any occurrences of the keyword prompt from a clause and interprets the next line as being the name of the clause rather than the next clause of the rule. The program EXTRAS presents each of the prompt names to the developers of the application and they can then enter the text associated with that particular prompt. The actual prompts are stored in a file with extension .PRO (PROmpt).

When the users are presented with Yes/No questions and the clause in question has a prompt associated with it, it is possible to add another option to the question display. If this option is selected by the users the text of the prompt is displayed on the screen until the users press a key. It is also possible to specify commands as prompts and if the prompt is requested, the appropriate command is executed. This allows the developers of the system to make use of other facilities to provide the prompts. For example, it would be possible to use graphics to help explain the question.

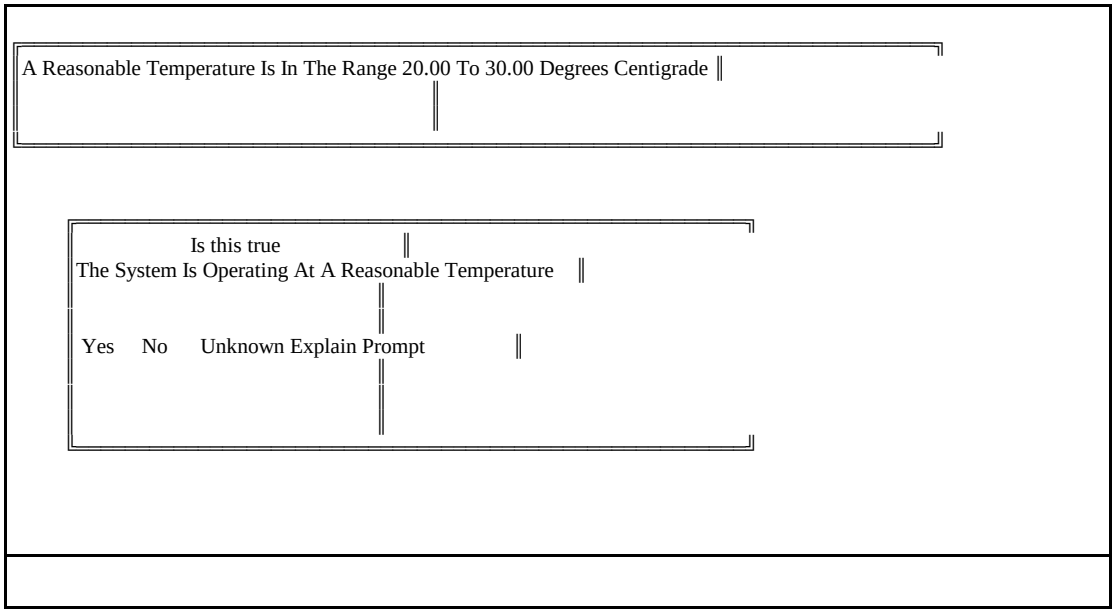


Figure 6.7 - A question with a prompt

Prompts in requests

In addition to questions, however, the users may also require further assistance when the system requests certain pieces of information. This causes problems when using the line editor provided by the Pascal compiler. The line editor does not trap certain key presses that can be used to indicate that the users want access to the prompt information. In PESYS prompts are requested by pressing the **ESCAPE (Esc)** key. However when this is done in the standard line editor, the result is simply an abandoning of the line currently being edited.

These problems were overcome by writing a new line editor. This editor provides all the facilities offered by the default editor, such as control over cursor movement by a character or a word and also allows text that is longer than the chosen window to be entered as the line is scrolled to keep the edited text in the screen window. The most important feature of the new editor, and the reason for it being written, is that it traps certain error conditions such as when the users press **ESCAPE (Esc)**. These error conditions can then be used by the main program to allow the users the possibility of having a prompt displayed. The option of making use of the explanation facility of the system is also provided at this point.

It is possible that the users are half way through entering a value in response to a request before they realise that they may be entering an inappropriate value. If they press **ESCAPE (Esc)** and use the prompts provided, they may decide that their answer was in fact

correct and they want to continue editing it. In other cases, they may want to abandon their previous answer and type in a new value. The default line editing routine does not allow the re-editing of existing values, however this can easily be incorporated into the replacement line editor. This has the added advantage of allowing the system make use of default values which can be altered or replaced by the users. In common with the handling of default values in most packages, if the first key pressed represents a `character', then this is taken to imply that a new value is being entered and the previous value is deleted. Any other keypress, such as a cursor key, is taken to indicate that the existing value is to be edited and the system acts accordingly.

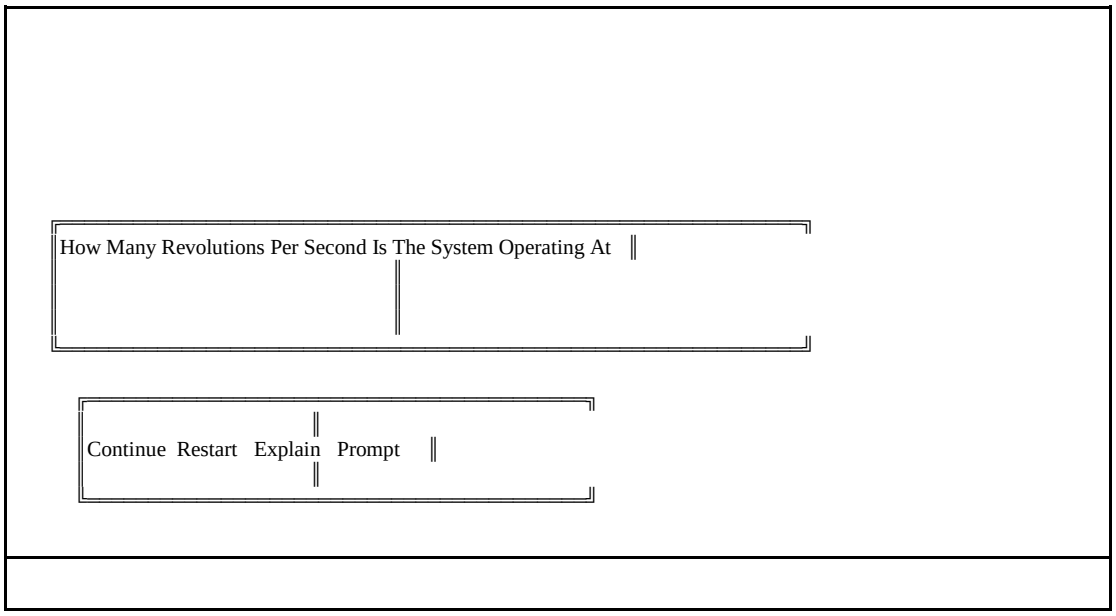


Figure 6.9 - Prompts within a request in PESYS

Forms

Normally once a request has been satisfied in the PESYS system, it is not possible to alter the value entered. At best the system provides a What-if analysis facility to alter the value after the inference process has been completed. In many applications, however, this is insufficient and any facility that allows the users to alter values, no matter how limited, would be useful.

A forms facility was developed in PESYS to provide some of the extra functionality required. This is simply a method whereby a background text screen can be displayed with slots which allows the users to enter values for variables and where the results of expressions can be displayed. Variable slots contain the names of variables which the users can enter

whilst expression slots contain expressions which are evaluated using the current data values before being displayed on the form. The advantage of using forms is that the users have the ability to cycle through all the variables on the screen making sure that they all have appropriate values. The ability to re-edit existing values is particularly useful here. By keeping a number of related variables on the same form, the users have the opportunity to alter the values indefinitely until satisfactory values are chosen and the system can then continue with the inference process.

The variable slots in PESYS call the variable inputting routines described in the commands section of PESYS and therefore it is possible to enter strings and reals as values for variables with specific questions being posed for each variable. It is also possible to enter minimum, maximum and most likely values using the slider with the most likely value being displayed on the form. The number of characters of the variable value that are displayed is independent of the actual value stored. For example, a form may only display the first ten characters of a name.

ii. A case study

The PESYS system was used by Ashwajit Singh to develop an expert system to assist in filing income tax returns in India[1] (Whitley *et al.* 1989). The system was designed to solve a particular problem but also provided useful feedback on the usefulness of the system of prompts provided. Further information about the application is provided in Appendix V.

Income taxes have long been the principal means of taxation in industrial countries. They are able to generate a great deal of revenue and offer scope for income redistribution with relatively few distortions. In developing countries such as India, however, they are often difficult to administer, raise little revenue and offer little scope for income redistribution. The Indian Income tax authorities undertook to simplify the tax legislation and introduce computers to the process of filing tax returns. It was hoped that this would reduce the burden on accountants, allowing them to undertake more rewarding work (Whitley *et al.* 1989).

The expert system was designed to be used by someone who has no experience of Income Tax Legislation and aimed to assist them in filing income tax returns. It was

[1]This work was supervised by the author along the lines indicated in this thesis.

therefore decided that the system should present users with simplified questions which would guide them through the process of filing their returns. This meant that the text of the legislation was interpreted by the knowledge engineer / expert who provided simplified versions to be presented to the user. During this process it was noticed that a significant bottleneck arose when interpreting the legislation.

In some cases, however, the users of the system would require further information about the interpretation of the clause and in this case it was decided to display the actual legislation in the prompt. Thus the users were given a simplified version of the legislation when answering questions and responding to requests whilst the full text of the legislation was available if needed.

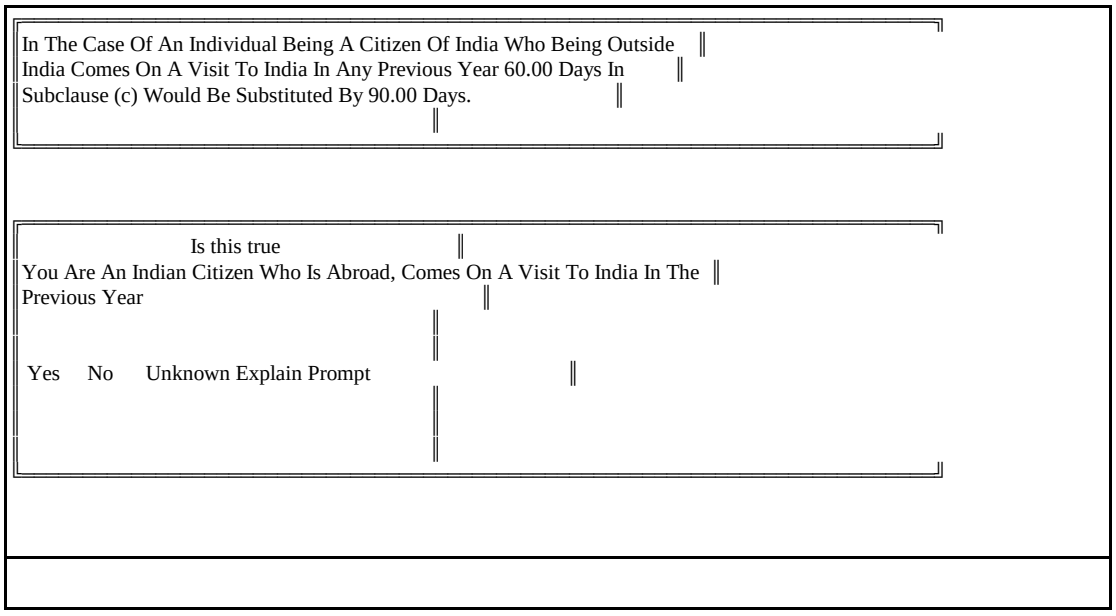


Figure 6.11 - A prompt in the Indian Income Tax application

iii. Discussion of the limitations of prompts

Although the prompts offered by the system provided information about the choice of interpretation used in the system, also possibly explaining why that interpretation was chosen, they do not convey to the users how other possible interpretations relate to those used by the system. For example, the users might be told that if they had left India for the purposes of employment outside India then their residential status might be altered, but this would not be of much assistance if they had undertaken voluntary work outside India and they wanted to know if they were still covered by this particular part of the legislation.

Thus the prompts are only really useful if the users' interpretations are closely related to those of the developers of the system, if they lie within the boundaries of the system and the prompts are used to confirm this belief. The nature of knowledge in semi-formal domains, in particular when it is based on descriptive and subjective definitions suggests, however, that the users are as likely to have interpretations that lie outside that explicitly covered by the knowledge base. A prompt that is only useful for confirming a particular interpretation is therefore not entirely satisfactory. What is needed is for the users to see how their own particular interpretation relates to that used by the developers of the system.

Furthermore the use of prompts with tacit skills can do little more than convey to the users that a certain skill is to be used and if they do not possess that skill no further assistance is offered. Thus a prompt would simply state that the users are expected to view the engineering drawing as the thing it represents and would, in general, not provide assistance as to how this seeing-as could be attained by those users that did not have the necessary tacit knowledge / tacit skills.

c. THE USE OF NON-LINEAR DOCUMENTS

This chapter has so far discussed two attempts to overcome the problems of knowledge in semi-formal domains. The first of these was based on a method which tried to find the basic ideas expressed in the clauses of rules in the knowledge base. It was found that this method was not particularly suitable for domains where there was no predefined structure. The second method attempted to provide information that described the intended interpretations of the clauses found in the rules. Whilst this was a more successful attempt, it was restricted in that it only provided information about the chosen interpretation and if the users had formed other interpretations they could not easily relate their own interpretations to those used by the system. What is required is some way of linking various interpretations that is not based on a `deeper understanding' of the domain which may simply not exist. The final solution examined in this chapter is based on the work of Wittgenstein and Searle on the boundaries between different items.

i. Wittgenstein and the similarities between names

Wittgenstein (1953, §65-69) examines descriptive definitions to see if any common features exist between different examples of the same name, any features which could be used to make the definitions prescriptive. He argues that "if you look at them you will not see something that is common to *all*, [the things that share a descriptive definition] but similarities, relationships and a whole series of them at that. ... And the result of this examination is: we see a complicated network of similarities overlapping and criss-crossing: sometimes overall similarities, sometimes similarities of detail" (§66). Wittgenstein suggests that no similarities exist between *all* the things that use a particular name, rather that **family resemblances** exist between different, overlapping parts of the things given a particular name.

The term "expert system" is an example of a descriptive definition. Possible similarities between expert systems that could be used as prescriptive definitions for the term include:

! It is implemented in an artificial intelligence language - but the CATS expert system (Harmon and King 1985) is implemented in the FORTH programming language.

! The knowledge base is represented as if-then production rules, but PROSPECTOR's knowledge base is represented using semantic networks (Harmon and King 1985).

! The inference engine of the system is kept separate from the knowledge base - but "[A]ll too often, expert system developers must alter a problem's logic to make rules fire in the right order" (Vrba and Herrera 1989, p. 76).

! The system mimics the performance of a human expert - but accounting packages often perform the same tasks as human experts, i.e. professional accountants, and are not normally classified as expert systems.

! The system is based on the knowledge of a human expert - but the British Nationality Act expert system (Sergot *et al.* 1986) was developed using only the published legislation and no human experts (i.e. trained lawyers) were involved in the development process.

In fact the only similarities that exist between *all* expert systems seem to be that they are computer programs (but then so are word processors, spreadsheets and databases and they are not normally referred to as expert systems) and the fact that their developers use the name expert systems for them![2]

[2]After Wittgenstein "Now you are only playing with words" (1953, §68).

One possible attempt to overcome this problem would be to introduce some kind of weighting system. This would mean that if something satisfied, say 70% or more of the available criteria for a name, then it could legitimately use that name. By attempting to impose a formal structure where none exists or is appropriate (which is the case with descriptive definitions) an interesting paradox arises. Consider the case of adding grains of sand, one by one, to a heap on the ground. Eventually this heap of grains will be called a *pile* of sand. If the number of grains of sand in the pile are counted (say **N**), then a `formal weighting' process, based on the number of grains, must imply that the addition of *one* extra grain of sand to a heap of **N-1** grains suddenly makes the resulting amount a pile. The capabilities of this *single extra* grain of sand to take the sand beyond the boundary of a heap and into a pile is counter-intuitive but *only* arises if a prescriptive definition of a pile is attempted.

The problem, therefore, arises when attempts are made to try and prescriptively specify boundaries where no formal boundary exists. Searle (1969, pp. 5-12) argues that the problem of borderline cases does not show that we do not understand the concept being considered, but rather that we recognise a borderline exists means that we have grasped the concept concerned.

ii. Non-linear documents

The final solution to some of the problems of knowledge in semi-formal domains discussed in this thesis makes use of **non-linear documents** (NLD) to implement a system based on Wittgenstein's family resemblances. The term non-linear document technology is similar in many ways to the notion of "hypertext" which is a glossy but uninformative term.

The standard way of moving through a computer based `document' has its origins in the reading of paper documents in that the reader starts at the top of the document and then moves linearly through the document until the end is reached. Of course paper documents do allow reading to take place in a non-linear fashion. For example, when reading an academic paper the reader may often jump to the list of references if the text mentions something that the reader wishes to examine further. Once the references have been examined, the reader continues reading the main text again. The use of footnotes has a similar effect of altering the linear reading of the document.

The necessity of the reader to keep track of the previous position imposes a significant limit on the amount of non-linear reading that is possible in a paper based document. In the case of a computer system, however, such limits are less significant. By implementing the movement through a NLD as a recursive procedure, a computer based system can automatically return the reader to the previous part of the document as required. Moreover this process can be repeated an indefinite number of times, limited only by the available memory of the machine.

iii. Non-linear documents in PESYS

Non-linear documents within PESYS are made up of two main components. The first of these is the text of the `document'. This is stored as a series of screens and each screen holds text on a particular topic, idea or interpretation. On those occasions where there is more text than can be held on a single screen it is possible to link a series of screens together with the users simply pressing **PageDown (PgDn)** and **PageUp (PgUp)** to move between the different screens. Each screen is given a unique name which can be used to identify it, it also has an internal code number that is used to locate the screen in the NLD file.

The second component of non-linear documents are the links that are used to allow the users to move through the document in a non-linear manner. Each screen has a number of links which are displayed in turn. When a particular link is selected the system displays the new screen that is associated with that link. **Figure 6.13** shows how a number of different screens can be linked together. This new screen itself has its further links which allow the users of the NLD system to move through the entire document in a non-linear manner. By implementing this movement as a recursive procedure it is possible to automatically backtrace through the screens examined.

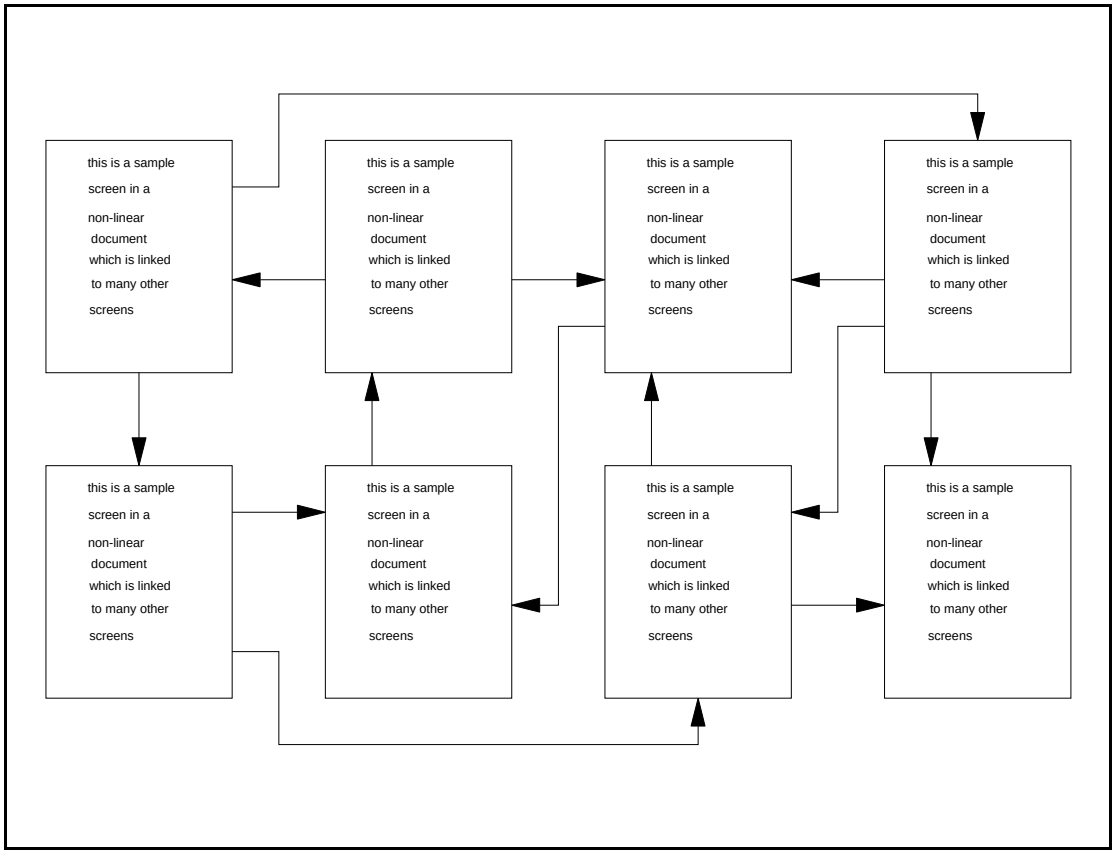


Figure 6.13 - How various screens may be linked together

The NLD is entered by specifying a particular screen. An index to the NLD is kept in an ordered file and this is used if access to the NLD is performed using the name of a particular screen. The system uses this file to find the record that is associated with the name. This record contains the number of the screen and the screen can then be displayed. The alternative method of using the system involves providing the screen number immediately. However, this is only possible if the number is known, as is the case in the system help facility.

Creating a non-linear document is quite difficult to describe since it is necessary to provide links to other screens which may not exist at a particular time. It is therefore necessary to continuously update the information when a previously undefined screen is now defined. When creating a new non-linear document, it is often advisable to layout the basic structure of the different screens, describing where the links are to be made.

iv. Implementing non-linear documents in PESYS

Each non-linear document in PESYS is made up of three files. The first is simply an ordered index containing the names of all the screens together with their identifying numbers. These numbers are used to locate the records associated with the screens in the other files. The main file is made up of records containing the textual information displayed in each of the screens. In addition to containing the text associated with a particular screen, each record also contains information as to where the links between screens are stored and also which screens should be accessed should the users press **PageUp (PgUp)** or **PageDown (PgDn)**. The final file contains the links between screens and each record in this file is made up of the text of the link, where this text appears on the screen, the location of the next link on that screen and the screen that the link is directed to.

The records in the file are accessed using the random access file handling facilities provided by the Pascal compiler. These are described in more detail in Boisgontier and Donay (1988). Essentially the file is made up of a number of records each of which can be accessed by setting the file pointer to the number of the required record. The first record is number 0. In the non-linear documents, screen number 0 is used as a general help screen / index that can be used by the users to move to any other screen in the file. The random access file handling provided by Turbo Pascal allows both reading and writing of records at a particular point in the file. This means that it is possible to edit or delete individual records as necessary. It is important, however, that the individual links are properly maintained otherwise it will be possible for users to move to previously defined screens only to find that they no longer exist.

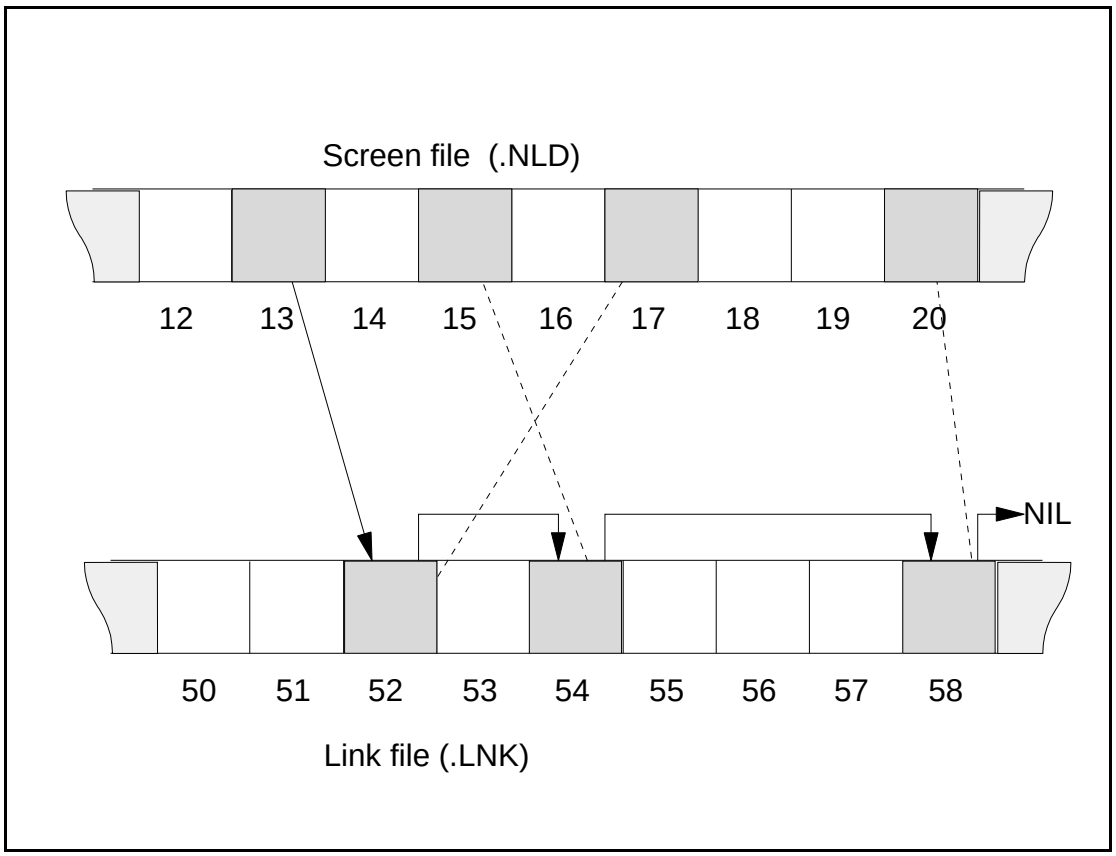


Figure 6.14 - How the screens and links are stored in a file

Figure 6.14 shows how the two main files associated with non-linear documents operate. In this example, the first link from the screen found at position 13 in the file is found at position 52 in the link file. This link is to screen 17 and the next link on the screen is found at position 54, which points to screen 15. The third, index file is simply used to find the number of a given screen by using its name. This is done by performing a binary search on the ordered file and reading the appropriate record number from the located file. As **Figure 6.14** shows, each screen contains links to a number of other screens and these are marked by having the screen record contain the record number of the first link record. This link record then contains the number of the next link record as well as the screen it links to. Note that the use of random access means that it is not necessary for the links or screens to be stored in any particular order.

Non-linear documents implemented as commands in PESYS can therefore be included automatically in a knowledge base to provide information at a particular point in the interaction. Moreover, since commands can be used as PESYS prompts, it is possible to specify a particular screen in the command for a prompt and the system will immediately

display this if the users select prompt when answering a question. This is the most likely way of using NLD technology in PESYS.

NLDs are also used to provide help with the PESYS system in general and can be selected by pressing the help key **F1** at any time. Whenever this key is pressed, the system provides the help about the current item. By carefully linking these help screens the users are able to obtain assistance about the current option, other options available at that time and also the overall use of the system.

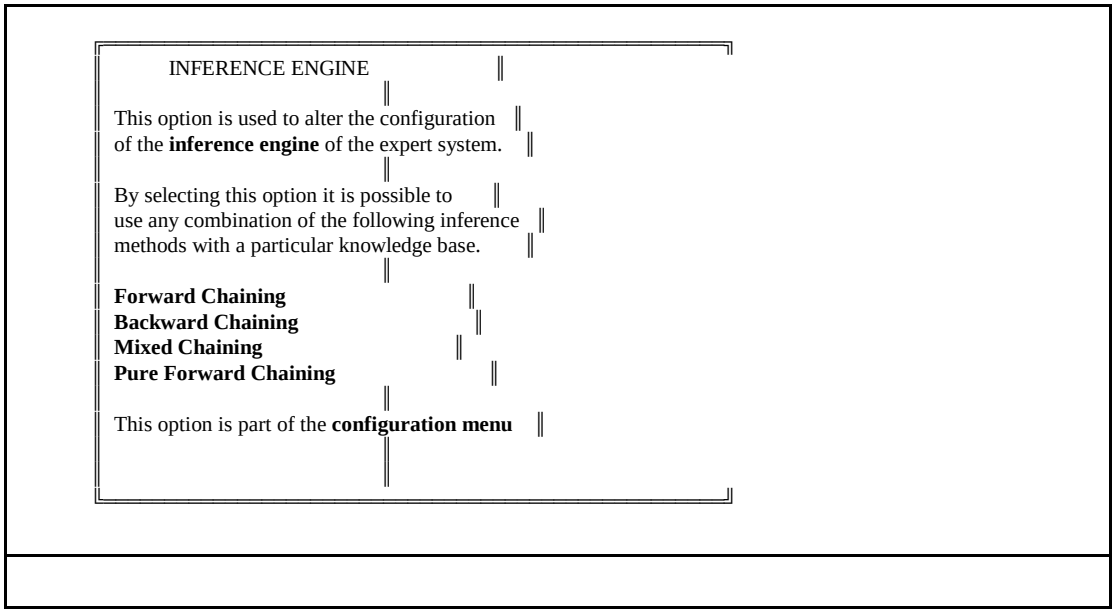


Figure 6.15 - A help screen in PESYS

v. Using the non-linear documents in PESYS

The process of developing a NLD file for a particular application is rather difficult to explain and is presented as part of the user guide in Appendix III. However, the use of non-linear documents in the runtime part of the system is much less complicated and will now be explained.

When the NLD is selected (either by pressing **F1**, by selecting a prompt or when the NLD command is executed as part of the knowledge base) the chosen screen is displayed on the screen. Also highlighted at this time will be the first link to another screen. By pressing the **TAB** key it is possible to move the highlight through all the possible links on that screen. If the **RETURN** key is pressed on any of these highlights then the system will move through the link to the next screen. This process can continue indefinitely.

Pressing the **CTRL** and **F1** keys at the same time retraces steps through the chosen screens. This allows movement back through the available screens. Alternatively, pressing **ESCAPE (Esc)** will immediately return the users to the main program. Pressing **F1** whilst in a NLD results in the index / help screen (screen 0) being displayed.

vi. Non-linear documents and multiple interpretations

Non-linear documents can be used to allow the users of a particular system to examine multiple interpretations of a particular term. The initial screen presented to the users on requesting assistance will convey the interpretation used in the application. However links will be provided to other related interpretations. Those relations that have direct family resemblances with the one used in the system will be linked to the initial screen, whilst others may be linked less directly. The users of the system can then examine the various screens that are available on the system until they decide that they have been able to incorporate their own interpretations with those used by the system. When this has been done, they can return to the questions or requests posed by the inference engine.

As an example of this process consider the use of non-linear documents with an expert system that makes use of the idea of race as it was discussed Chapter 3. Suppose the system asks the users to specify their race and they are unsure of what category they should use. They use the NLD to provide assistance with this question and the system displays information about the possible categories available as is shown in **Figure 6.17**, based on the classification used in Stanford University.

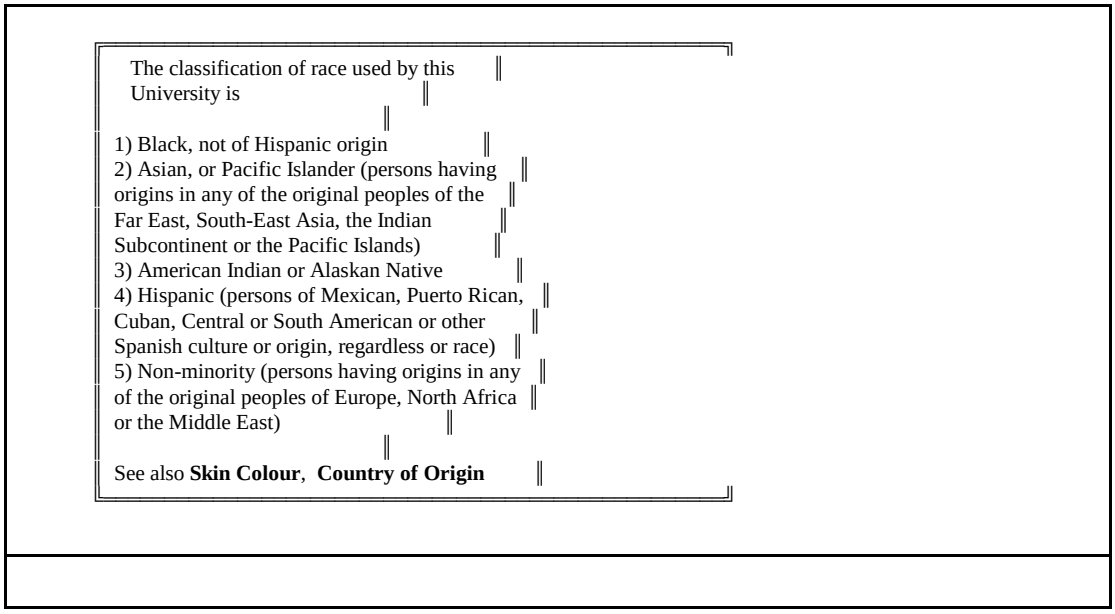


Figure 6.17 - A screen from a non-linear document explaining `race'

However, the users of the system have an American Indian and Hispanic origin and are therefore unsure as to which category they should use. One of the available links on the screen is `skin colour' and when this is selected a new screen is presented, see **Figure 6.19**, which explains that skin colour was only used to determine race in the early 1960s in states like Virginia, where the only options available were white and coloured. In addition, the system may provide some indication as to how skin colour relates to the categories available in the system, perhaps suggesting that, for the purposes of the expert system, being an American Indian takes precedence over any Hispanic origins.

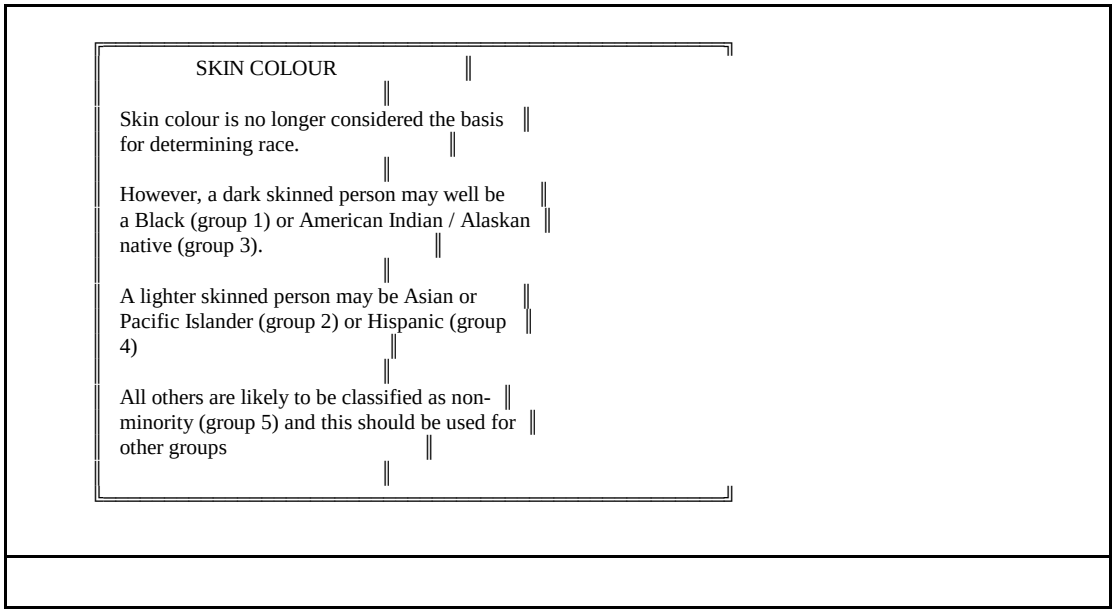


Figure 6.19 - A Family Resemblance to `race' - skin colour

vii. Using non-linear documents to examine the boundaries of the knowledge base

Non-linear documents allow the users of the expert system to examine different interpretations of the terms that cannot be accurately represented in the knowledge base. The different interpretations are linked in various ways and by using Wittgenstein's family resemblances it is possible to guide the users to other interpretations that are similar to the one used by the developers of the system.

By viewing as many different interpretations as necessary, the users of the system are able to examine the boundaries to the knowledge base. They can see the interpretations that are within the scope of the application and they are also able to examine others that lie outside the boundaries of the knowledge base. In doing so they are able to form a better understanding of how their own interpretations are related to those of the developers. Even if their own particular situation lies outside the boundaries of the knowledge base, they will be able to determine precisely why they are outside the knowledge base.

viii. Non-linear documents and tacit skills

Non-linear documents can also be used to help convey some of the tacit knowledge used by an application. Whilst the very nature of tacit knowledge means that it is not possible to fully formalise this knowledge it is possible to provide more assistance to the users than simply stating that tacit knowledge is used in a situation. For example, consider the tacit knowledge used in seeing-as in the case of an engineering drawing. The previous prompt approach would simply state that the users were expected to interpret the diagram.

By using non-linear documents, however, it would be possible to provide some assistance to those users who do not possess the appropriate tacit knowledge. For example, the system could display instructions stating that thick lines are used to indicate the outside edges of the component and that these should be considered first so that the overall shape can be determined, see **Figure 6.21**. It can then provide assistance with other parts of the drawing as necessary.

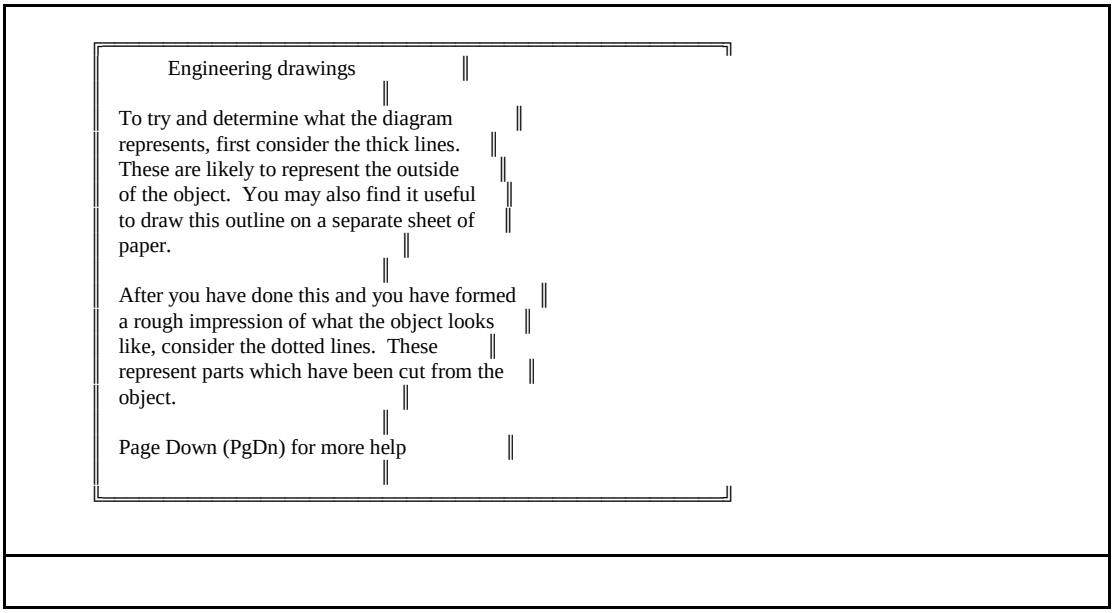


Figure 6.21 - A non-linear document providing assistance with tacit skills

Whilst this approach will not be perfect, since the very nature of tacit knowledge means that it cannot be specified, it will at least be of some assistance. In some cases, the users of the system may possess some of the tacit knowledge required to solve the problem but they may not feel confident enough to make use of it. If, however, the system can provide some support in this role they may have the confidence to make use of the skills that they do possess in conjunction with the functionality offered by the system.

CHAPTER 7 - SOLUTIONS TO THE PROBLEMS OF COMPUTER BASED KNOWLEDGE BASES

Chapter 4 discussed the problems that arise because a knowledge base is implemented on a computer system. It was argued that a computer system cannot normally become aware that the users of the expert system application are confused. This chapter describes solutions to these problems based on the theory of speech acts that was presented in that chapter.

a. USING EXISTING COMMUNICATIVE RESOURCES

Suchman's work on plans and situated actions, which was described in Chapter 4, provides an excellent model for examining the interaction that takes place between humans and computer systems. The conclusion of her thesis, however, seems to go to considerable lengths not to state an implication which is present throughout her work. "Today's machines", she argues, "rely on a fixed array of sensory inputs, mapped to a predetermined set of internal states and responses. The result is an asymmetry that substantially limits the scope of interaction between people and machines" (Suchman 1987, pp. 180-181). This part of her analysis provides an important insight into the problem of human-machine communication. Unfortunately, her analysis of the "problems for the design of interactive machines" are overwhelmingly biased against an obvious theme of her research. The three problems she states are, (emphasis added) "the problem of how to lessen the asymmetry by *extending the access of the machine to the actions and circumstances of the user*. Secondly, the problem of how to make clear to the user the limits on the machine's access to those basic interactional resources. And finally, the problem of how to find ways of compensating for the *machine's lack of access to the user's situation* with computationally available alternatives" (p. 181).

A major theme of her work has been the fact that there are *two* parts of the interaction that are not conventionally made available to the other participant in the communication. The speech and intentions of the users are not conventionally available to

the machine, whilst the design rationale of the system is not available to the users. Suchman's solutions, as emphasised above, seem simply to tackle the limited availability of the user's actions to the machine and she ignores entirely the possibility of making the user aware of the design rationale of the machine. In part, this is probably due to the fact that she discusses the problems associated with operating a photocopier which is not the most obvious candidate for allowing the user to examine the design rationale of the system, yet she is also trying to find general areas for further research and her omission of this viewpoint is surprising.

i. Confusion and examining the design rationale

The discussion of confusion and the means by which a computer system can become aware that the users are confused suggests that it is impractical for the system to try and anticipate or notice that the users are confused. The realisation that they are confused is likely to be an action that cannot readily be made available to the machine and, just as importantly, this realisation can occur at any time. The examination of speech act theory in the context of expert systems proposes three occasions when this realisation is likely to occur; three occasions when the differences in background assumptions and interpretations between the users of the system and the developers of the knowledge base are likely to reveal themselves. The occasions when the assumed background between the users and the designers reveal themselves, when they become unready-to-hand, are when the system makes an assertion, when it asks a question and when it makes a request.

> If he does not make such and such a change, I'm going to ask him why. And if he can explain to me something that's logical and based on sound data, then I'll go ahead and do what he wants to do. (Benner 1984, p. 138).

This quotation describes how a nurse overcomes a discrepancy between a request made by a doctor and the request that was expected by the nurse. Although taken from a field (nursing) that has few apparent similarities with expert systems it offers a useful insight into the domain independent nature of how the use of speech acts can help overcome any uncertainty on the part of the users.

The nurse is similar to the users of an expert system in that she has an unarticulated assumed background about the problem domain which does not seem to correspond to that conveyed by the `expert' in the field as it is exhibited in the request made. When this occurs she is prepared to accept the advice of the expert only if the expert is able to explain "something that's logical and based on sound data", i.e. something that allows her to understand why the expert's opinion is to be accepted, something indicating why the felicity conditions for requests have, in fact, been satisfied. In the case of the nurse this will be done by the doctor pointing out something that she has not considered or by the doctor offering a different perspective on the problem. If this explanation is not satisfactory, however, the nurse will not voluntarily accept the doctor's request.

The Dreyfus model of skill acquisition, discussed in Chapter 2, suggests that the production rule formalism used in expert systems is not capable of describing true expertise. It is, however, an excellent method for conveying competent skills and as such is ideally suited to justify something to users who are prepared to be corrected so long as they can see the reason why that alternative should be taken. In this case, in fact, the simplicity and clarity of the description of the competent skill is a positive advantage.

ii. Designing support for overcoming confusion

In order to show how tools to support this process are implemented in PESYS, the act of making an assertion, which arises when an expert system states a goal (or in the case of PESYS an inform statement), will be considered. Those features that relate specifically to requests and questions will be described later.

By asserting an inform statement the expert system may cause the users of the system to realise that they may be confused. This can occur when the system asserts a result that is not expected or alternatively when the system fails to make an assertion that is expected. The users will therefore want to resolve the problem. In the PESYS system assertions are made whenever a rule fires and one of the then-clauses is marked with an inform level of 1 (goal) or above (useful information). If the assertion causes the users to realise that they are confused, then in most cases the computer based expert system will not be aware of this and the users will have to take the initiative to overcome the problem. The asserted clause is displayed on the screen and the system waits for the users to press any key. By pressing the

SPACE bar the users can indicate that they believe that some confusion has arisen which they want to resolve.

Upon determining that confusion has arisen, steps must be taken to try and resolve the misunderstanding that arose. The nature of the confusion is not, in many cases, available to the system and the system cannot therefore offer any automatically generated solutions. The theory of speech acts, however, describes general conditions that are assumed (at least on the part of the receiver of the act) to apply when an assertion is made. As was discussed in Chapter 4, for a non-defective assertion the speaker is expected to have evidence or reasons for the truth of the proposition being asserted. It must not be obvious to both the speaker and the hearer that the hearer knows the proposition (or does not need reminding of the proposition). Also, a sincere assertion is made if the speaker `believes' the proposition and the assertion must count as an undertaking that the proposition represents an actual state of affairs. The notion of commitment in the communication suggests that the speaker is expected to be "willing and able to articulate why" the assertion is believed.

When the expert system makes an assertion about which the users are uncertain, there is an implied commitment on the part of the speaker to explain why the assertion is believed. The nature of computers means that this commitment to explain cannot be generated automatically. The users of the system, however, can make use of tools that allow them to determine the reasons why the assertion is believed.

In an expert system, the only evidence for the assertion being made is that a particular rule fired. The expert system only `believes' in the assertion to the extent that its working memory contains facts (entered by the user - i.e. indirectly believed) which were used to fire that rule. Thus using Winston's animal recognition knowledge base (1984) the only reason for asserting that "the animal is a tiger" is because the rule Identify-10 fired. This rule only fired because of the facts entered by the user which were added to the working memory.

```
RULE IDENTIFY-10
IF
animal is mammal
animal is carnivore
animal has tawny colour
animal has black stripes
THEN
animal is tiger
```

Figure 7.1 - Winston's rule for identifying a tiger (Winston 1984, p. 282)

In this case, the `belief' in the animal being a tiger follows directly from the facts that are known to be true and the rule that was fired. The rules in the knowledge base can therefore be seen as the means by which the experts and knowledge engineers convey their belief in certain assertions. It is possible, however, that the eventual rules used may misrepresent the intentions of the expert if there have been problems in the knowledge acquisition or implementation stage of the development process. Such cases of `unreasonable' rules leading to unjustifiable assertions would be observed and ignored by the users of the system who can then use the system to follow other reasoning paths (see below).

iii. Implementation of the support

An assertion is made when a rule with an inform marked then-clause is fired. The rule can only fire if all the if-clauses of the rule are known to be true (i.e. are found to be true in the working memory of the system). It is therefore advisable to display the entire rule for the users of the system indicating the if-clauses that are known to be true. In general, however, some if-clauses may be known to be false and others may be unknown and all three types should be displayed. In PESYS a ✓ is used to signify those clauses that are known to be true, **X** is used to signify clauses that are known to be false and **?** is used for clauses whose truth is unknown. Commands, which are "true" by default, are marked with **-** to show their special status. A typical rule is shown in **Figure 7.2**.

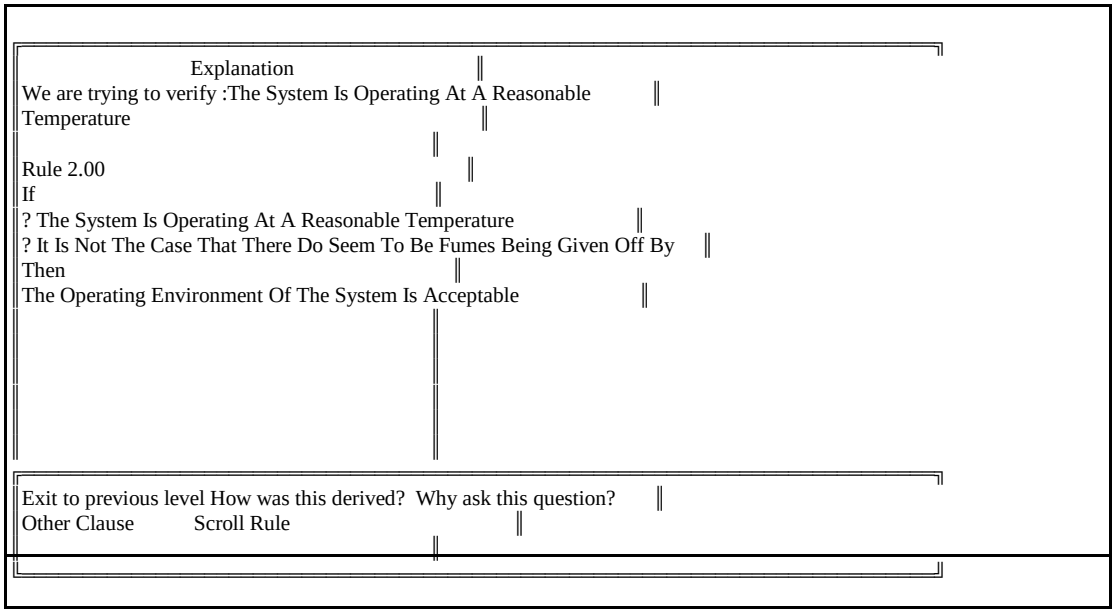


Figure 7.2 - An explanation in PESYS

When commands such as comparisons and variable manipulations are used it is useful to have the actual values displayed on the screen rather than simply stating that the commands were executed and the comparisons evaluated. Thus rather than just specifying that Number_of_components > 20 was true, the actual value of the number of components should also be shown. When commands such as Enter are used, it is possible to specify a large number of variables that are to be requested from the users and their actual values may be longer than one line. In such a case, the system only displays one screen line of values but allows the users, through the scroll rule option, to highlight the particular line and scroll through all the possible values. The scroll rule option is also used when the if-clauses and then-clauses in the rule cannot all be shown on the screen at the same time and an example of such a rule is shown in **Figure 7.4**, with the same rule in a scrolled form shown in **Figure 7.6**.

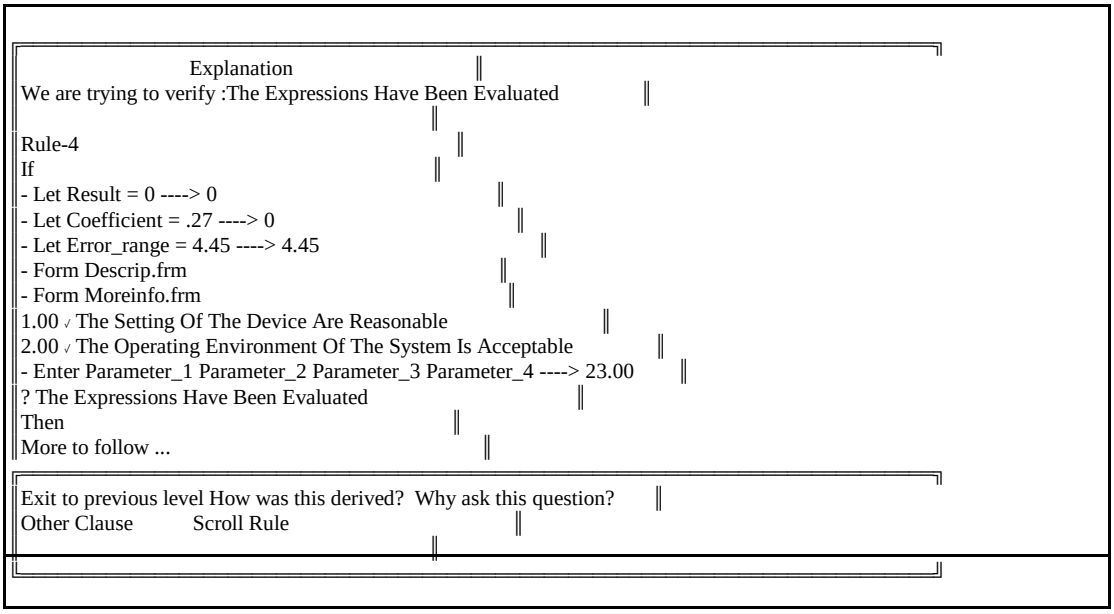


Figure 7.4 - An explanation based on a long rule

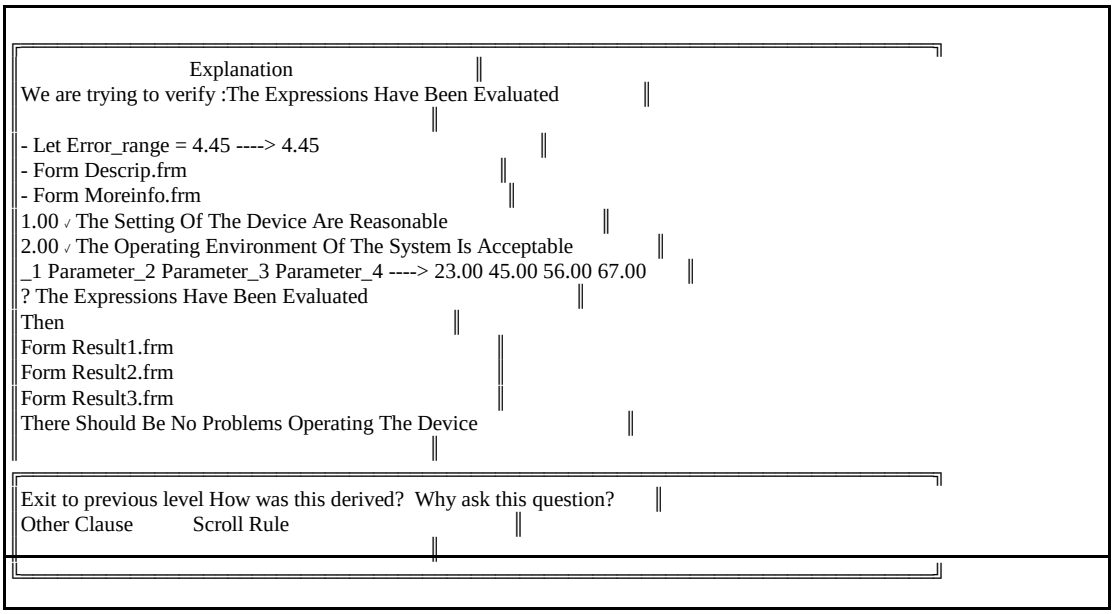


Figure 7.6 - The same rule after scrolling

iv. Understanding the justification for the clauses

In some cases, simply viewing the rule that was fired will be sufficient for the users to accept that the assertion made was both reasonable and appropriate. This is particularly likely to be the case when the users have forgotten or failed to consider a possible link between various conditions and the assertion made.

In other cases, however, simply displaying the rule that fired will not be sufficient to allow the users to resolve the confusion that has arisen. They may accept that if certain conditions are true then a certain assertion can be made in good faith but they disagree that the conditions for making the assertion have in fact been met. The users should, in these circumstances, be able to see the rationale behind the if-clauses in the particular rule to see how they have come to be known to the system, i.e. the justification for these if-clauses needs to be made apparent.

Selecting the if-clauses

Since most rules will have more than one if-clause it is necessary to select the particular if-clause that is being examined so that the justification or rationale for why that clause is known to be true can be examined. It is possible to examine the rationale behind all the known if-clauses, but each one needs to be examined separately. On machines that have high resolution screens and mouse type pointing devices, the most appropriate method of selecting a clause to be examined would be to move a pointer to the clause and pressing a button on the mouse. On a standard IBM PC, without a mouse type input device, a more feasible method would be to number the known if-clauses (i.e. those clauses that are not commands and whose truth is true or false but not unknown). To examine a particular clause, the users simply need to enter the number of that clause. Rules with only one known if-clause will cause that clause to be automatically selected and only valid choices will be accepted by the system. The clause chosen to be examined can be determined from its number.

Once the clause has been identified, the means by which it was added to the working memory needs to be found. The inference engine in PESYS only allows clauses to be added to the working memory in two ways. They can be arrived at as a result of a rule firing or they can be entered by the users in response to a question. It would be possible to store the rule used to arrive at a clause with the clause in working memory, however this method is rather wasteful of memory, particularly since many of the clauses may not need to be examined by the users.

An alternative method would involve examining all the rules that contain the clause in its then-clauses and seeing which of these rules fired. These rules could then be used to provide the explanation. At first sight this seems to be a computationally intensive task,

however it should be borne in mind that only those rules that have fired need be examined (since the clause can only be arrived at when a rule fires) and it is possible to mark those rules that have fired with a boolean flag. Moreover, in most applications the number of rules fired will only be a proportion of the total rules found in the knowledge base. If no fired rules could be found which contain the clause in their then-clauses then it follows that the clause *must* have been entered by the users.

Multiple rules that have fired

Some knowledge bases may have more than one rule which fires adding the particular clause to the working memory and in these cases the users are asked to select the rule which they wish to follow the reasoning through. Once the rule used has been selected, it can be displayed to the user in the same way as before.

In some cases, displaying the rule that asserted this fact may be sufficient for the users to accept the original assertion, in which case they can return to the previous level and continue with the inference process. In other cases, however, they may still want to examine the if-clauses of this rule further in which case the entire process is repeated again recursively.

The resolution of the confusion depends very much on the tacit skill of noticing problems which was discussed in Chapter 3. In addition to being a tacit skill, the noticing of confusion is highly influenced by situational factors which cannot be determined in advance by the designers of the expert system or the developers of the application and it is therefore impossible to plan for it arising. There should therefore be no limits on the amount of browsing in the knowledge base that the users can undertake and, indeed, they should be encouraged to perform this task until they are completely satisfied that the felicity conditions for the assertion have been satisfied.

In practice there is a slight memory limitation in the PESYS system as it stores the previous screen image as each rule explanation is displayed. This is only likely to cause a problem if a single line of reasoning is examined to a considerable depth. Differences in interpretation are likely to reveal themselves at a far earlier stage than this however and so problems are unlikely to arise in most applications.

v. Why-not justifications

The description that has been presented so far has concentrated on allowing the users of the system to examine how the felicity conditions associated with making an assertion are satisfied. The need to examine the felicity conditions of an assertion may also arise when a particular assertion is not made. The realisation that something `ought' to have been asserted can only come about when the system performs a particular act. From the discussion of confusion, the users of the system can only realise that an assertion ought to have been made when some other assertion is made instead (or when an unexpected question is asked or an unexpected request is made - see below). In the case of assertions, therefore, it is only when the system makes an assertion other than the one expected by the users that they will realise that confusion has occurred.

Felicity conditions can again be used to help determine why a particular assertion has not been made. For a rule to fire and make an assertion its if-clauses need to be known to be true. One possibility, therefore, is that the responses made by the users have caused some of the if-clauses of the particular rule to be entered as false when they would need to be entered as true for the rule to fire and the assertion to be made.

The second possibility is a direct consequence of the way in which the inference engine is implemented and used since it is possible that the required rule has not fired simply because the truth of some of its if-clauses are not known; they have not been considered yet. If the users are to examine the design rationale of the system, to follow the felicity conditions of the assertions, then these cases must also be considered.

Expected assertions are not made

When the system fails to present an expected assertion the users must be able to examine how this came about. Since the expected assertion has not been made, the rule that didn't fire is not immediately available to the system and, more particularly, the system has no way of determining which assertion was expected.

From the use of inform levels in PESYS knowledge bases two paths can be followed to select the rule that would have fired to make the expected assertion. The users can choose to examine a particular goal to see the reasoning that would need to be followed for that goal to be arrived at. Alternatively the users may choose to start the examination from a particular

inform 2 statement, particularly if the assertion was an intermediate statement. A list of rules which contain inform 1 then-clauses (goals) and a list of rules containing inform 2 statements are created when the knowledge base is loaded and they are used to allow the users to select the reasoning path they want to follow and the choices available are shown in **Figure 7.8**. Depending on the choice made, the appropriate list of clauses is shown and the users are asked to select the reasoning that is to be examined.

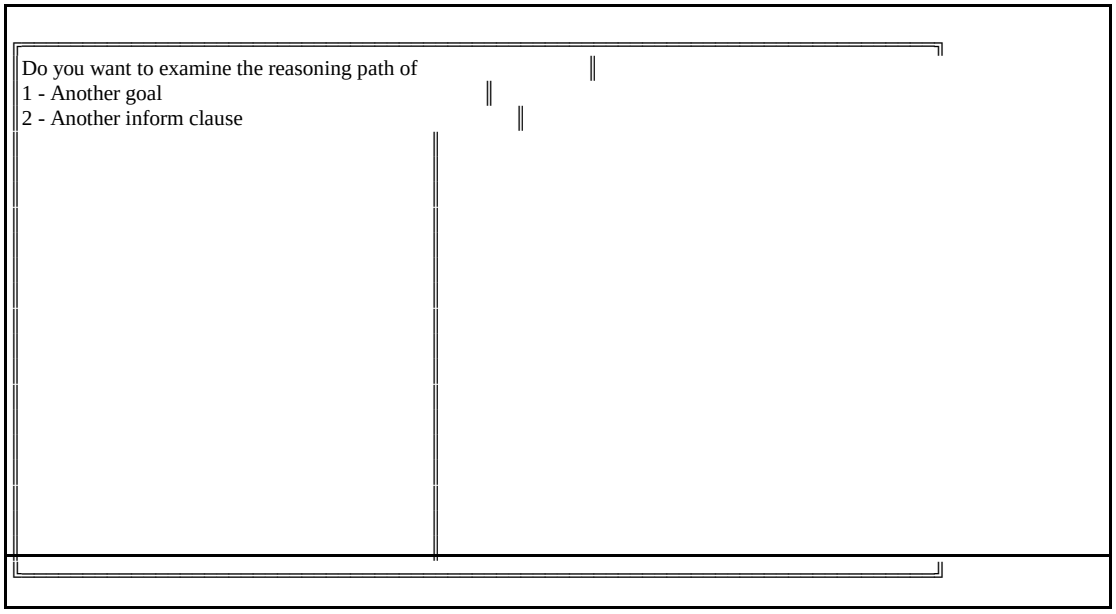


Figure 7.8 - The option to examine other goals or other clauses

The appropriate rule is then displayed in the same way as for assertions that have been made. Once the rule is displayed, a second problem arises, one which is particularly important for those occasions where a rule has not fired because the necessary information has simply not been obtained. Previously all the if-clauses were known and the means by which they were known could be examined. In the case of an assertion that has not been made, however, not all the if-clauses are known, or they are known to be false. Assertions that are known to be false can be tackled in the same way as those that are known to be true, but those that are unknown need extra support to deal with them. Whereas previously the system allowed the users to examine the reasoning through rules that fired, clauses that are unknown can only be examined through rules that have not fired. In PESYS this is performed by considering all the rules that could be used to assert the requested fact, not just the ones that have fired. Another consequence of this is that the choice of clauses to be examined must now include all the possible clauses rather than just those that are known to be true.

Rousset and Safar (1987) describe a system that provides negative and positive explanations in an expert system. They give no indications, however, as to why such explanations are required. The explanations generated are not particularly easy to understand either, as is shown in **Figure 7.10**.

```
Why-not bidding = diamonds?
        - possible bidding = diamonds
                - possible bidding = the longest
                - two long suits hand
                        - biddable spades
                        - biddable diamonds
                        length1 <> length2
                the longest = diamonds
        but
                bidding = the biddable major suit
                - preferring major suit
        We cannot then conclude on bidding = possible bidding
```

Figure 7.10 - A negative explanation from the system described by Rousset and Safar (1987)

vi. Questions and requests

In many respects the way that the design rationale of the system is made available when questions and requests cause the users to realise that confusion has arisen is very similar to that for assertions. Questions and requests do, however, differ in that they are made in order that a particular rule will fire rather than being a consequence of a rule having fired. As in the case for assertions however, the sincerity for the acts is simply a consequence of how the knowledge base was written and how the inference engine uses the knowledge base.

When a question or request is being performed, therefore, it is being performed to make a particular rule fire. This, again, is the extent of the sincerity behind asking the question or making the request. If the rule that is being examined is then displayed for the users it will provide information as to which clauses will be arrived at (and possibly asserted) if the question is answered appropriately or if the response to the request satisfies a certain condition.

In some cases, however, simply being told that a question is being asked to arrive at certain then-clauses is insufficient to overcome the confusion that has been revealed. In these cases the system must provide further information relating the clauses of the rule to other rules in the knowledge base. Since the inference engine is simply manipulating the knowledge base that was written by the knowledge base designers, the only reason for asking a question or making a request is that it will allow a rule to fire which adds then-clauses to the working memory of the system, which allows another rule to fire ... until a rule fires which adds a goal (inform 1 statement) to the working memory. This is the rationale behind performing the particular act when the inference engine is doing a backward chain and forms the basis for the explanation provided. When forward chaining is taking place no further explanation can be provided *because* no other rationale exists for the act to be performed.

The explanations for backward chaining can easily be generated from the backward chaining system. Whenever the backward chaining algorithm recursively considers a new rule, the current clause and the rule that it is found in are stored in a special stack. When the explanation of the request or question is being created, the necessary information can be found on this stack. This process can be repeated until the rule that will be fired to add a goal clause to the working memory is found and the stack is therefore empty.

As well as providing assistance with the questions and requests that are asked on some occasions it is beneficial to examine questions and requests that could have been made. The system may ask the users about topic A when they are expecting to be asked about topic B instead. In this case a why-not type explanation is required. However, the number of possible questions or requests made by the system is normally large. PESYS, therefore uses the previously described other clauses option instead and allows the users to examine other goals or other inform 2 clauses. This particular choice is made because the questions or requests are only considered to be significant in relation to the inform statements that they are used to arrive at.

How this method differs from more conventional approaches

The method of explanation described above differs from more conventional approaches in a number of respects which will now be summarised. One of the main arguments that has been presented in this thesis has been that the users of the expert system may form a different understanding of the system or may have different tacit knowledge to

that intended by the designers of a particular application. One important consequence of this is that the users of the system may realise that this difference of interpretation may have caused confusion to arise. The explanation facility is therefore needed to overcome the resulting confusion.

The system described above makes use of speech act theory and considers the three acts that are performed by the expert system, namely making an assertion, asking a question and making a request. It was argued that for each speech act, the users of the system expect certain felicity conditions to be met and that their confusion arises when these conditions are not met. The explanation facility must therefore attempt to show the users why the conditions were in fact met so that the confusion can be resolved.

In addition to the speech acts that have been performed, the noticing of confusion may also arise when an expected speech act is not performed. The system therefore allows the users to examine reasoning paths that could have been followed in addition to those that were followed. Again this feature is not normally present in conventional explanation facilities.

The limited communicative resources of computer based expert systems mean that it is not possible for the expert system itself to be aware of most cases of confusion that have arisen and it is therefore necessary for the users to initiate and control the use of the explanation facility. The examination of these communicative resources suggests that the conventional belief that the expert system can automatically generate the appropriate explanation for any situation is infeasible.

b. RULE IDENTIFIERS AND MULTIPLE GOALS

In the previous discussion it was implicitly assumed that each inform clause was only found in a single rule and that each known if-clause in a rule was only arrived at from one other rule. In practice, however, this is not the case and the system needs to be able to distinguish between the same clauses that were or could be arrived at using different rules.

In PESYS this is performed by using the rule identifier to allow the users to select which rule they want to examine. For example, when examining how an if-clause was arrived at there may be two rules which have (or could have) fired, adding the clause to the working memory of the system. The rule identifiers of the two rules are then presented to the users and they select the one which they want to use.

The rule identifiers that have been assumed so far are simple titles such as *rule-one* or *identify-10*. When such a list is presented to the users they will have little indication as to which rule they want to examine. They will either require a printed version of the knowledge base which they can use or they will have to examine all the rules individually. This is not an ideal situation.

Moreover, when a rule is presented to the users to enable them to examine the rationale of the system, it has been assumed that the users are able to determine the intended effect of the rule simply by examining the if-clauses and then-clauses of the rule. To some extent the use of natural language clauses in the rules simplifies this matter, but it is likely that the intended effect of many rules will not be readily conveyed simply by the clauses in that rule. A description of the intended effect of the rule, which plays no part in how the rule fires, would be sufficient to convey the intended effect.

Using rule identifiers

PESYS seeks to overcome these two problems by using the rule identifier as a description of the intended effect of the rule. The rule identifier is already displayed with the rule itself in the explanation screens and plays no part in the actual inference process yet is normally sufficient to convey the intended effect of the rule to the users, see **Figure 7.11**. The limit to the length of the rule description is 255 characters, which is the size limit for strings in the Turbo Pascal compiler used.

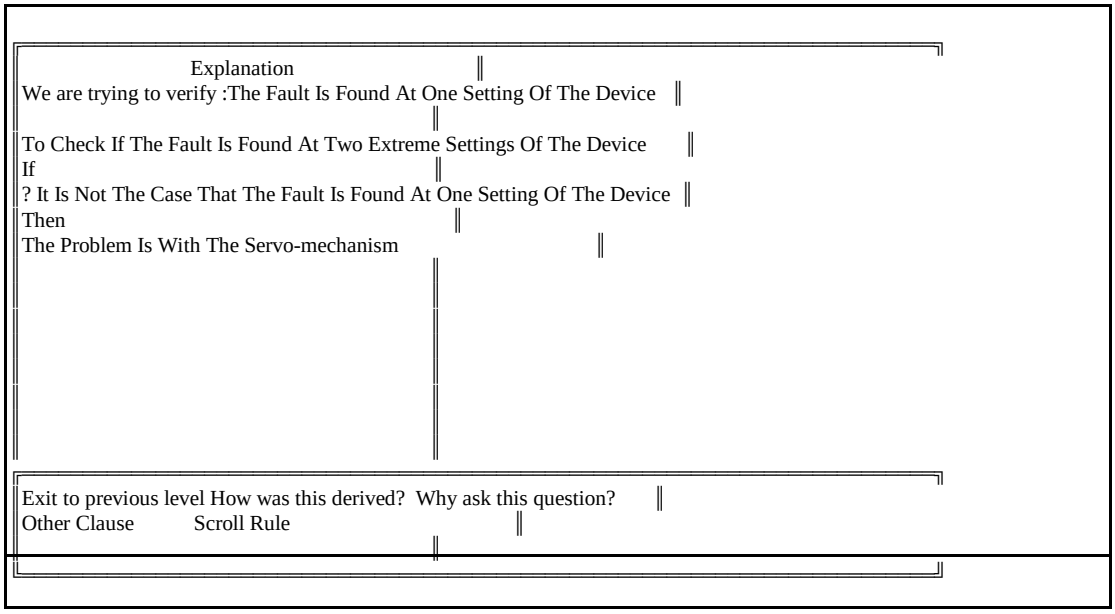


Figure 7.11 - Meaningful rule names assist in the explanation

The only obvious disadvantage associated with replacing the rule identifier with a free text description of the intended effect of the rule arises when the knowledge base is being debugged since there is no longer a direct way to identify particular rules. However, most editors now support sophisticated search facilities which enable particular rules to be identified using their descriptions. Moreover, if required the description of the intended effect of the rule can include the rule number as well, see **Figure 7.13**.

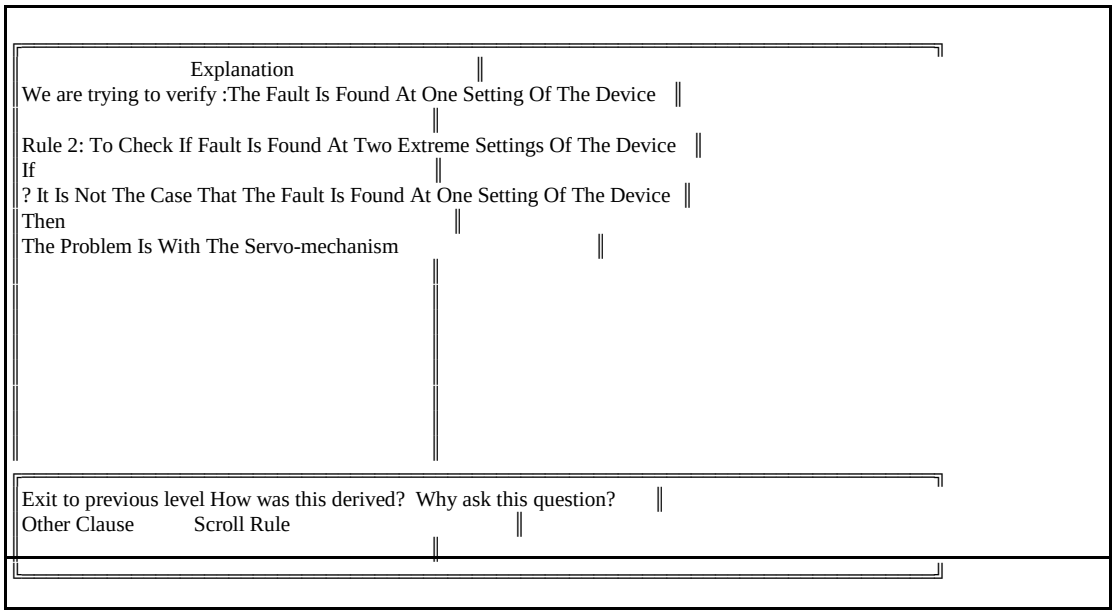


Figure 7.13 - It is possible to include a rule number in the rule description

i. Multiple goals

The standard form of inference in PESYS, described in Chapter 5, suggests that the system halts as soon as a goal clause has been arrived at. In some cases, however, it may be desirable for the system to find a number of different goals rather than halting when the first one is found. This means that multiple faults or hypothesis can be arrived at by the system and this may be more useful than simply reporting one. Allowing multiple goals differs from repeating the inference process a number of times since only those goals that are consistent with the existing known facts are considered.

Multiple goals are of particular use when the facts of the case are not known with any degree of certainty. Consider a simple expert system that has been developed to distinguish between different drinks. Two of the drinks considered might be coffee and cola. The rules associated with these drinks are given in **Figure 7.15**

```
this rule fires if the drink is cola
if
the glass and liquid are cool
and the liquid is a dark colour
and the liquid has bubbles
then
inform 1 the liquid is cola

this is a rule to identify coffee
if
the glass and liquid are hot
and the liquid is a dark colour
and the liquid has milk
then
inform 1 the liquid is coffee
```

Figure 7.15 - Some rules for multiple goals

Consider the case of users who are presented with a strange liquid which is neither hot nor cold, but which is warm. The liquid is certainly dark and has something in it, something which might be milk or which might be bubbles. By entering these facts and using the inference engine to find more than one goal, the users are able to use the system to examine the rationale behind the assertion that the drink is cola and the drink is coffee. The assertion that the drink is coffee can only be made if the warm liquid is a hot liquid that has

cooled down and the particles in the liquid are milk. Similarly the expert system can only assert that the liquid is cola if the liquid is a cool liquid that has warmed up and the particles are bubbles. By examining the reasons for making these different assertions, the users can decide which of the two assertions best fits the problem situation.

ii. Implementing multiple goals

Multiple goals can be implemented in PESYS simply by allowing the users to choose between pressing the **SPACE** bar when an inform 1 statement has been arrived at and pressing any other key. By pressing the **SPACE** bar, the users are indicating that the inference engine should continue and find any other goals that are consistent with the known facts.

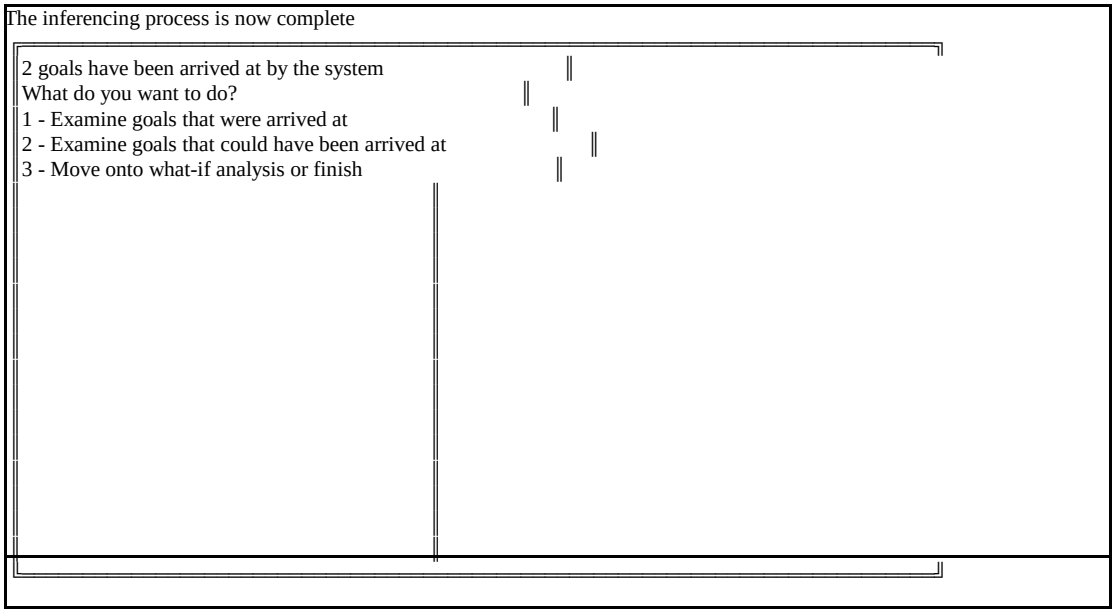


Figure 7.16 - The display for the completion of the inference process

When the inference process is complete, after the system has arrived at zero, one or more goals, the users are offered the choice of examining the reasoning of the goals that have been asserted or examining the reasoning of the goals that could have been asserted, see **Figure 7.16**. If more than one inform 1 clause was asserted, the system presents a list of the asserted statements and their associated rule descriptions and allows the users to select the one that they want to examine, see **Figure 7.18**.

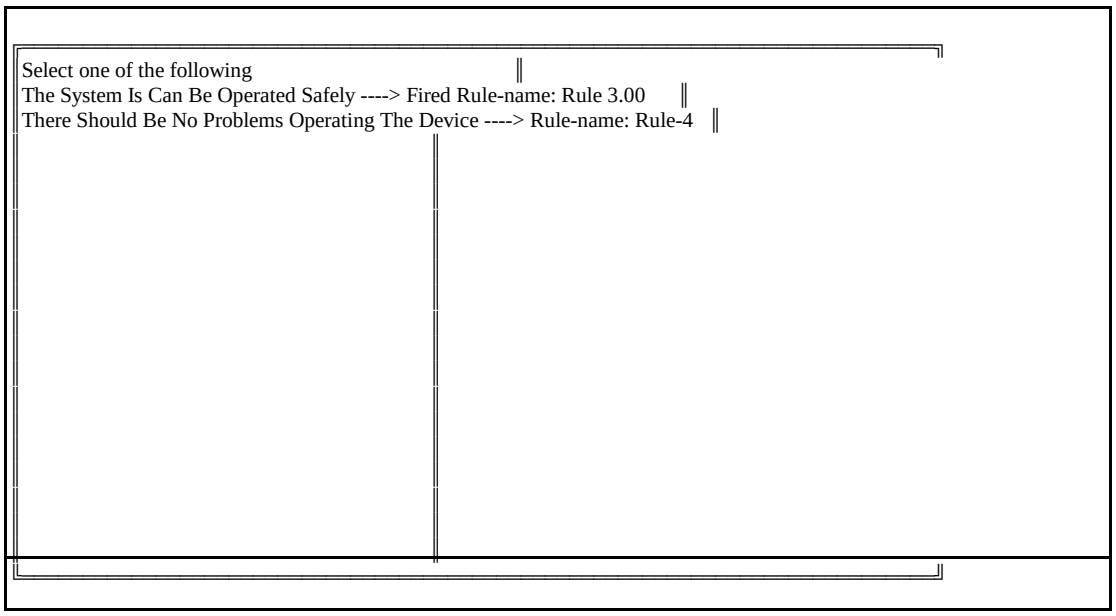


Figure 7.18 - The goals that could have been arrived at

Since the assertion of a goal is identical to the assertion of an intermediate inform statement, the system also allows the users to examine the reasoning of any of the possible goals. Although it would be possible to restrict the choice to all those rules that have not fired, a more general solution is to list all the possible goals and rules that could be arrived at and to mark the rules that have been arrived at with the keyword *Fired*. This allows the users to examine the goals that were not asserted as well as those that were.

CHAPTER 8 - CONCLUDING DISCUSSION

This chapter summarises the research presented in this thesis and discusses the main implications of the work. It also describes a number of further consequences of the practical work described in the previous chapters, linking them to the ideas developed by this research. The chapter describes the final version of PESYS and ends by discussing further areas for research.

a. EMBEDDING EXPERT SYSTEMS IN SEMI-FORMAL DOMAINS

i. The formal approach to expert system design

Chapter 2 of this thesis described two contrasting approaches to developing expert systems. The first of these takes a functionalist approach to design and assumes that expert systems operate within a regulated, clearly bounded area which is formally defined. Domains that are not formal are considered to be informal and are not suitable for expert systems development. Since formal domains consist of readily identifiable objects and concepts it is possible to model the domains in an expert system knowledge base using heuristics or rules of thumb to do this. The knowledge base is seen to lie outside the problem domain and is often assumed to contain all the knowledge necessary to solve problems in the domain.

Those problems that do arise are assumed to be a result of insufficient knowledge on the part of the users of the expert system. The expert system therefore takes control of the interaction, guiding the users of the system in their attempts to solve the problems that they face, providing them with the knowledge that they do not possess. In its extreme form, this means that the users are simply considered to be the hands and eyes of the expert system, performing those actions on the domain that are not possible by the expert system.

The expert system is used as a machine to solve problems and little consideration is given to the needs or capabilities of the users. In particular, they are not expected to make

use of their own intuitions or skills when using the system. This means that minimal support is provided for those users who, for example, may wish to examine other ways of solving the problems because other factors in the problem appear to be more important.

ii. The semi-formal domain approach to designing expert systems

The functionalist approach to designing expert systems emphasised the distinction between domains which are formally defined and are therefore suitable for expert systems developments and those which are informal and are unsuitable areas for using expert systems. By taking an information systems perspective on this distinction, which considers formal systems to be a part of informal environments, and computer systems as a suitable technology for part of the formal system, the concept of semi-formal domains as spanning the boundary can be seen. These are domains which combine formal and informal aspects in a single problem area. For example, there may be a set structure to performing a certain task, but the details of the individual elements in this structure are based on experience and other subjective factors which would normally be classified as being informal. The domain therefore comprises of both formal and informal aspects in a single area and therefore it can be classified as a semi-formal domain.

Expert systems are computer systems and cannot be developed for domains where there are no formalisable features whatsoever, i.e. for domains which are completely amorphous, yet there are many domains that combine some formal structure with informal elements and these are seen as suitable areas for expert systems development. The semi-formal approach to designing expert systems can be seen to provide an alternative to the functionalist method.

The functionalist approach to design considers the expert system to be distinct from the domain that it is concerned with. In particular this means that the knowledge is `extracted' from the domain (often using metaphors based on the extraction of mineral deposits) and `refined' for use in the knowledge base of the expert system. The semi-formal domain approach, in contrast, considers the expert system to be `embedded' in the domain, forming one part of the ensemble concerned with solving the particular problem being faced. Other important actors within this complex are the users of the system, the experts who

provided the original knowledge and the knowledge engineers who form it into structures that can be used by the computer system.

Embedding expert systems in semi-formal domains emphasises the role that the users of the system play. They are no longer considered to be simply individuals who do not know about the domain, but are now considered to be intelligent actors using the expert system to assist them in a task. The expert system is therefore used as a tool to help them solve problems, and by using this tool appropriately, they are able to enhance their own skills and capabilities. In particular, the system allows them to examine the consequences of their own opinions about relevant factors in the domain. In doing so, the users may become more confident both about using the system and accepting its results and they are more likely to accept the solutions provided by the system.

Consideration of the informal part of semi-formal domains means that the social processes involved in communication are taken into consideration. In particular this means that consideration is given to the users of the system in terms of their interpretations of output displayed by the expert system. Since this is an informal, social process the potential for problems of communication are considerable, especially if the users of the expert system have different beliefs and understandings of the problem domain from the designers of the system. Such problems are made worse because one of the participants in the interaction is a computer system. To date there has been little discussion of the prospects of programming computers to take an active part in the negotiation and compromise necessary to overcome problems of misinterpretation. The problems involved in communication between the expert system and its users mean that it is not appropriate to talk of the users not knowing about the domain, but that instead it is necessary to consider the different features of the two parties.

iii. Are formal domains really semi-formal?

Formal domains are made up of regulated objects and concepts and their recognition and identification is normally independent of experience or subjective considerations. In semi-formal domains, however, this agreement on terms is not necessarily present and different individuals may choose to use terms in different ways. The discussion about human communication and the disambiguation pseudo-problem presented in Chapter 3 suggested that in practice groups of individuals, through a process of negotiation and compromise, come to agree on certain usage of terms and concepts.

By using this analysis it is possible to see how `formal' domains are constructed. The domain that is being modelled is not necessarily based on a single, measurable reality which is originally identical for all possible actors, but is standardised by a group of individuals for a certain purpose.

The formality of a domain can also often arise as a by product of a purposeful activity. When, for example, an electromechanical device is designed it is done so to solve a particular problem and it is constructed in a purposeful manner. A consequence of this is that the domain has a formal basis that can be used for developing an expert system.

iv. The benefits of considering domains as being semi-formal

The considerations of semi-formal domains that have been described in this thesis highlight a number of important factors which need to be considered when designing expert systems, even if the domains themselves are not initially considered to be semi-formal.

Design based on semi-formal domains forces the designers of expert systems to consider the wider social environment which the expert system is to be a part of. It emphasizes the role of the users in interpreting and understanding displays provided by the system and shows that many possible interpretations of a domain can be formed. Expert systems need to be designed to take such factors into consideration and should offer support to the users whenever the possibilities of misinterpretation or the lack of tacit knowledge arise.

This thesis has also examined the problems that arise because a computer based system is interacting with a human user in detail. When problems arise in this communication, it is important to consider the functional capabilities of the expert system to detect and resolve these problems. In particular, a computer system has limited access to actions that are occurring in the rest of the domain, other than through the key presses made by the users. Current systems are unable to participate in a process of negotiation and compromise when differences arise between the model of the domain developed by the experts and knowledge engineers and those formed by the users of the system.

b. EXAMINING THE BOUNDARIES OF THE KNOWLEDGE BASE

i. The limits to what can be stored in the knowledge base

Chapter 3 examined a number of different examples of knowledge that cannot easily be formalised and represented within the knowledge base of an expert system. The cases presented suggest that there are important boundaries to the contents of knowledge bases.

One area of semi-formal domains that cannot easily be represented in a knowledge base is the use of descriptive definitions. These are descriptions of terms used in natural language that do not have necessary or sufficient conditions determining their use. Descriptively defined terms are determined through use rather than from predefined conditions. Since there is no effective procedure for determining the appropriate use of a descriptively defined term, it is possible that different groups of actors (for example, the experts and knowledge engineers and the users of the resulting expert system) will use the terms for different purposes. It is not possible, therefore, to formally specify the knowledge required for the domain so that it can be accurately represented within the knowledge base since other actors may use differing interpretations of the same term.

Subjective definitions are a form of descriptive definition that arise when a particular descriptive definition is used by a particular group of people, or when the term is used for a particular purpose. In such situations the possibility of problems arising is heightened since the intended use of the term is often not made explicit by the person using it. If the person receiving the utterance is not aware of this intended use, it is possible that problems will arise.

Tacit knowledge is knowledge that cannot be told, and by its very nature cannot be represented in a knowledge base. One example of tacit knowledge which was described in Chapter 3 is the knowledge required to notice problems before they can be specified. In many cases a problem is noticed before it can be formally stated and it is often noticed by someone who does not have the technical skills required to formulate the problem that exists. Tacit knowledge is also used when things are seen as other things.

A final limit to the knowledge that can be represented in a knowledge base arises because much knowledge about the domain is hidden when things are working as they should. While no problems arise, the knowledge remains ready-to-hand and only reveals itself when a problem arises, for example, if the object fails to act in the manner expected, when it is not present or when its use hinders the task being performed.

ii. Limits to the knowledge base arising from the use of computers

In addition to the problems of knowledge in semi-formal domains, there are a number of important consequences of expert systems being implemented on a computer system as described in Chapter 4.

Computer systems are very limited in the communicative resources that they have available for accessing the semi-formal domain of which they are a part of. An expert system normally only has access to the keys pressed by the users of the system and possibly sensors that are linked to parts of the domain. However, many of the problems described in the previous section arise because of situational factors, such as who the users are, what their assumed backgrounds consist of and whether they possess the appropriate tacit knowledge for the domain.

The role that these situational factors play in purposeful action in general, and in the interaction with expert systems in particular, mean that it is often not possible to plan for all the problems that may arise. At best, a plan can be used to point out certain factors that are likely to be significant.

An important consequence of the computer's role in the interaction arises when considering the notion of confusion. It is not normally possible to know of confusion at the time that it arises, rather confusion only reveals itself at some later stage when the interaction becomes unready-to-hand. In many situations, however, the computer will not be aware that the interaction has broken down due to its limited communicative resources. In these cases, therefore, it is up to the users to act when they become aware that confusion has arisen, although it may be possible for the expert system to provide tools that assist them in this process.

Confusion is only likely to be noticed when the interaction between the users and the expert system breaks down. It is, however, possible to devise a domain independent theory that suggests how the interaction is likely to fail. Expert systems perform three basic speech acts: they can make requests, they can ask questions and they can make assertions. Each of these acts has a number of associated conditions that determine whether or not the act has been performed appropriately. Confusion may therefore be potentially identified if the users feel that the acts have been performed inappropriately, that these conditions have not been met.

iii. Software tools to examine the boundaries of the knowledge base

Chapter 6 presented and evaluated a number of attempts to overcome the problems of knowledge that arise when an expert system is embedded in a semi-formal domain. The main problem faced by the developers of current expert systems arises from the use of natural language as a medium of communication between the experts, the knowledge engineers and the users of the resulting expert system.

The first solution presented involved attempting to represent a textual clause in terms of the underlying idea it was attempting to convey. This idea is described in terms of a relation, which describes the particular `action' in the clause, and associated parameters which differentiate between versions of the same relation. When this method was used in a practical case study, however, it was found that there were many problems associated with the determining the parameters for the relations. In many cases it was not possible to describe the parameters in any organised way and the result was an ad hoc process.

One possible solution to this problem, which could be the subject of further research, would be the creation of specialised form of natural language for specifying the clauses. This language could then be analyzed using a compiler, such as YACC (Yet Another Compiler Compiler), which would automatically generate the compiled (i.e. relation) form of the clauses. Since the language used would be artificially created, it may be possible to avoid some of the problems, problems which emphasize the extent to which the informal aspects of semi-formal domains are significant. The problems arise because the domain that was being considered had been created through use rather than being constructed for a particular purpose. There was therefore no underlying formality that could be made use of in the development process.

In view of the problems with this method, it was decided that the emphasis of the research should move away from attempting to represent knowledge in a different way and that it was more advisable to provide assistance to the users of the expert system as to which interpretations of the terms were intended by the developers of the application. This assistance was implemented through the use of text prompts that could be displayed on the screen if the users required assistance about the interpretation of a particular term.

When the prompts were evaluated using a case study that required many such clarifications it was noted that, although the use of such prompts was suitable for confirming

that the interpretation of the users was similar to that intended by the developers of the expert system, the prompts were of little use for those occasions where the interpretations of the users appeared to differ significantly from those of the developers. These differences may arise because the users' interpretations genuinely lie outside the boundaries of the knowledge base, but in other cases it might be the case that the users' interpretations are still indirectly within the scope of the knowledge base, but the users have no way of knowing this.

The final solution to these problems offered in this thesis is the use of non-linear documents. These are pieces of textual information that can be linked in many non-linear ways allowing readers to examine related pieces of text at will.

In addition to conveying the intended interpretations of various terms to the users of the expert system, non-linear documents also allow the users to examine related interpretations in an attempt to find an interpretation which matches their own. This interpretation may lie within the boundaries of the knowledge base, or it may lie outside. The use of non-linear documents allows the users to examine the various boundaries that exist to the knowledge base to determine whether or not their particular interpretation lies within it or not.

Non-linear documents can also be used to provide assistance with some problems involving tacit knowledge. For example, although the skill of seeing-as is based on experience, there are many occasions where assistance can be provided to help those users who do not possess this knowledge. In these cases, the use of a non-linear document, which allows the display of many related items, can be very useful.

iv. Examining the boundaries of a computer based knowledge base

After allowing users to examine the boundaries of a knowledge base that has been developed for a semi-formal domain, Chapter 7 describes and evaluates a number of computer based tools that can be used to extend and open the boundaries of the knowledge base.

The limited communicative resources of computer based systems mean that in most cases the computer system will not be aware that the users of the system are uncertain or confused. From the analysis of these communicative resources it is suggested that these problems can be overcome by having the users of the system examine the design rationale of

the system. Relating this to the speech acts that are performed by an expert system, it becomes apparent that the design rationale of these acts depends on the conditions that need to be satisfied for these acts to be performed appropriately.

When one of the speech acts performed by the system causes the users to realise that confusion may have arisen, it is because the conditions for the act may not have been satisfied. The design rationale that needs to be examined, therefore, relates to how the conditions for the act have been satisfied. In the case of making an assertion, for example, the important condition to be satisfied is that the assertion is `believed'. In an expert system, this belief is the result of a rule firing. Thus to examine the design rationale behind this particular act it is necessary to examine the rule that was fired, and possibly indirectly, the rationale behind the facts that are asserted in that rule. Similar conditions apply to the other two acts of requesting and asking questions.

In addition to acts that are performed, it is also necessary for the users to be able to examine those acts that are *not* performed. For example, when a particular assertion is not made, it is necessary to determine whether this is a consequence of the knowledge base (does the assertion lie outside the boundaries of what could be asserted given the known facts), or is it simply a consequence of the way the inference engine operates in that the particular assertion has not yet been considered.

When the rules that are used are examined by the users, the intended effect of the rule is not always apparent. Although the clauses of the rule may be written in a natural language form, the rationale behind the rule are not always easily identified. The thesis describes a simple but effective solution to this problem that makes use of the identifier associated with a rule. Instead of simply being a code number, the rule identifier is used to describe and convey the intended effect of the rule to the users. When the rule is displayed on the screen, the rule identifier is also displayed and this provides an indication as to what the rule is supposed to do.

The solutions presented can be seen as attempts to both avoid and recover from the problems of semi-formal domains and this might be considered to be inconsistent. Why is it necessary to provide means of overcoming problems when the thesis also describes methods that can minimise the possibility of confusion arising? To answer this question it is necessary to refer back to how these facilities are used. The users will only make use of the non-linear documents and prompts *if* they think there might be some problems with the use of a particular term. If they feel confident about the use of the term then they will not make use of any further assistance. If their use was, however, inappropriate for the expert system

application and they become aware of the confusion then they will have to recover from the situation. Thus it is necessary to provide both sets of tools since the users tend to assume that they know what they are doing until they realise they are confused, in which case they want to recover from the problems they face.

c. IMPLICATIONS FOR THE DEVELOPMENT AND USE OF EXPERT SYSTEMS

In addition to providing software tools that enable the users of expert systems to examine the boundaries of the knowledge base, a number of other factors were noted that have implications for the development and use of expert systems.

i. The conceptual view of the user system interface

One of the main points raised by the comparison of formal and semi-formal domains is the conceptual view of the user system interface. The formal view considers the expert system as taking the dominant role in the interaction, controlling the actions of the users in solving problems. The examination of semi-formal domains provided by this thesis indicates that there are many practical problems to this approach and proposes an alternative view of the system.

The interface with expert systems in semi-formal domains is considered to be one based on co-operation rather than control. This is necessary since the computer has limited access to the domain and must therefore make use of the responses of the users. The users, however, may form inappropriate interpretations of the domain, or may not possess certain tacit skills required to handle a particular situation.

The point that needs to be emphasized is that the users are encouraged to actively participate in the interaction and not to act simply as passive data gatherers and action implementers. They should be encouraged to make use of their tacit skills, such as noticing that a problem may have arisen with the interaction, in conjunction with the facilities offered by the expert system.

By encouraging the users to participate in the interaction it will be possible for the system to be of some assistance even if the particular situation cannot be resolved. By examining the boundaries of the knowledge base, the users will at least be able to determine precisely where their interpretation of the problem differs from that offered by the developers of the expert system. If users do not take this active role in the interaction, they will at best be able to determine that their situation is not covered by the knowledge base, without knowing precisely why.

ii. The `interpretation bottleneck'

Such problems of misinterpretation that can arise mean that considerable effort must be used in providing useful interpretations of the terms used in the knowledge base, even if the original domain is reasonably well structured. In many cases, providing suitable interpretations can become a major bottleneck in the expert system development process.

Domains that are based on some formal representation, such as legislation, may appear to avoid the problem of knowledge acquisition since the domain knowledge is already in a form that can be used to construct a knowledge base. However, such domains are precisely the ones where interpretation is likely to be a significant problem, and it is likely that the interpretation bottleneck will replace the knowledge acquisition bottleneck.

The need for appropriate interpretations is particularly relevant for areas such as legislation, since the systems are designed to provide assistance to users who are unlikely to be completely familiar with the terminology used. However, the interpretations must not become so over-simplified that they miss out key points of the law. Reconciling these two considerations is likely to be a time consuming process.

iii. Reasons for design

Many of the problems raised by this thesis, such as the possibility of misinterpretation or the limited communicative resources of the computer could, in principle, be overcome by using collaborative development techniques such as prototyping. Whenever any of the problems discussed in this thesis arose, the prototype might be modified to overcome the problem.

Indeed prototyping is very common in expert systems development because the requirements of an expert system application are rarely known in advance and are not normally formally specified. Prototyping therefore offers a suitable means of arriving at an operational set of specifications for the expert system. Expert systems can also be considered to be very high level programming languages and as such support the prototyping process well. Moreover the inference engine is kept separate from the knowledge base which further improves this process.

Whilst many problems can be overcome by prototyping the expert system with the users, the purpose of this thesis is to describe and anticipate these problems *before* they arise. If most of the problems can be avoided before the expert system is developed, it will not be necessary to involve the users in unnecessary prototyping of avoidable problems.

It is suggested that these problems are likely to arise in most applications and it is therefore advisable to consider them in the feasibility or design stage of the expert system rather than leaving them to be picked up in the implementation stage.

d. THE FINAL PESYS SYSTEM

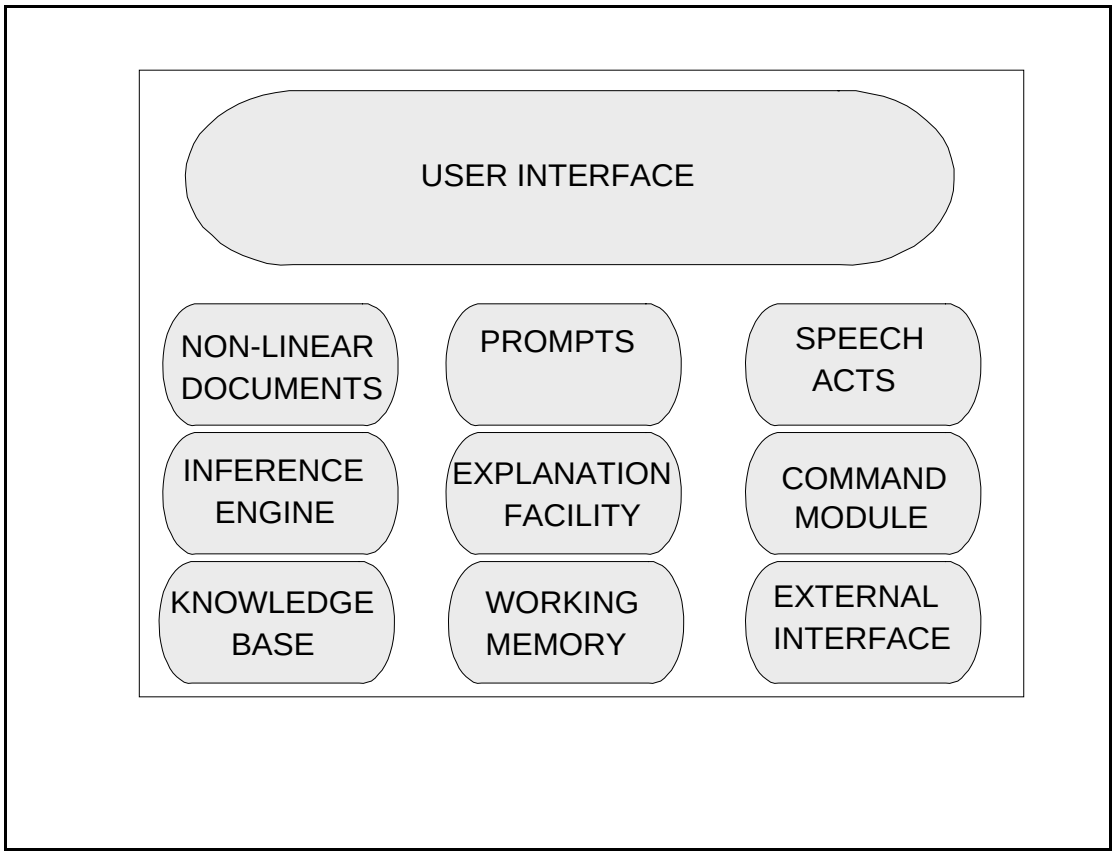


Figure 8.1 - The components of the final PESYS runtime system

The description of PESYS that was given in Chapter 5 was limited in that it did not include those components of the system that were specifically designed to overcome the problems of semi-formal domains which were described in Chapters 6 and 7. The final version of the PESYS system is described below.

i. The different programs in the system

Although the runtime component of PESYS has been extended considerably, it is still only a single program. However, it now contains a number of extra components in addition to those described in Chapter 5 and they are shown in **Figure 8.1**. In particular, the system now includes a component for making use of non-linear documents and the explanation facility makes use of the speech act module which is used to help the users of the system overcome any confusion that may have arisen. The system also provides prompts.

In the development phase, the functionality of the PREPARE program has been extended to take into consideration the use of prompts. The EXTRAS program now also asks for the text of any prompts that are to be displayed for the users.

Two extra programs have been added to the development phase, MAKEFORM and MAKE_NLD. The MAKEFORM program is used to create any forms that are to be used within an application and the MAKE_NLD program is used to create non-linear documents to provided assistance for particular applications. Full details of how to use these programs is provided in the user guide, see Appendix III.

ii. The different files used

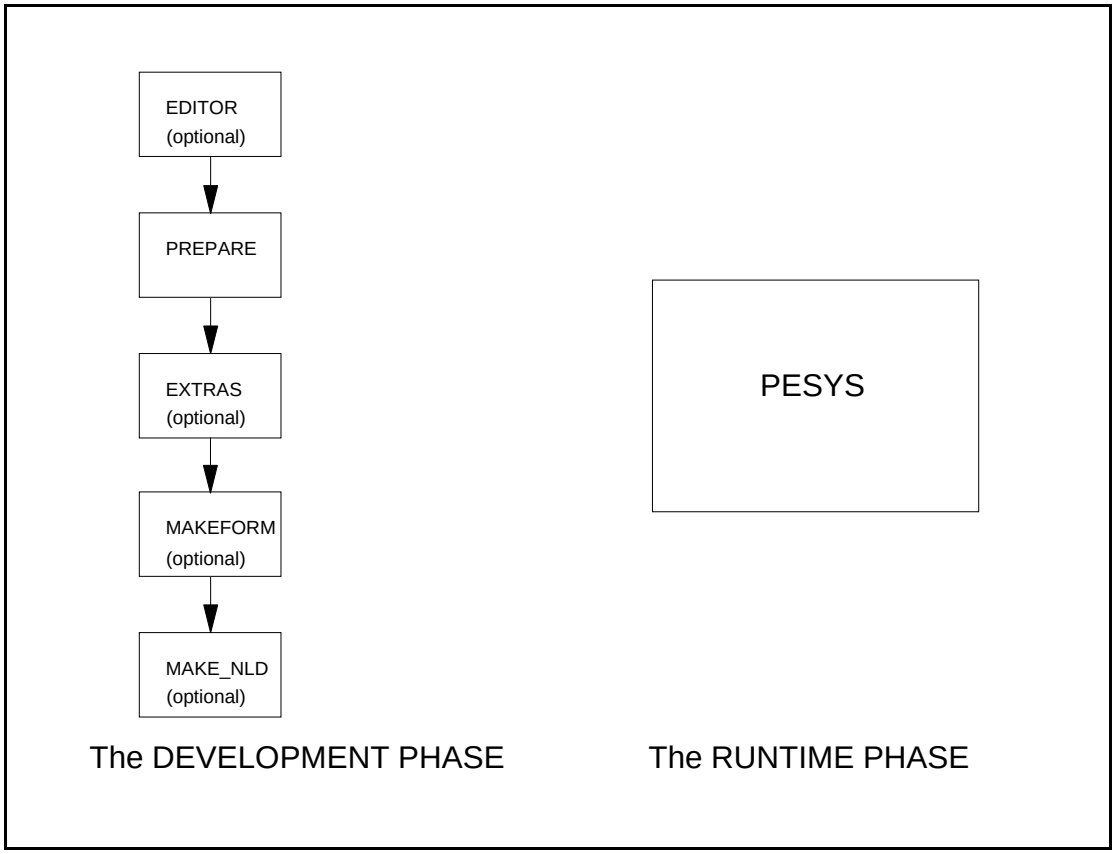


Figure 8.2 - The programs in the final PESYS system

The additional programs in the PESYS system mean that a number of new files are defined by the system. These files, and the programs that create them are shown in **Figure 8.3**. The new files are **.PRO** the prompts associated with a knowledge base, **.FRM** which contain the forms created by the MAKEFORM program. Non-linear documents make use of three new files and these are created by the program MAKE_NLD. The files are **.NLD** which contains the text screens for the non linear document. The file **.LNK** contains the information that links the different screens together and the **.NDX** file contains the index of screen names and their associated positions in the .NLD file.

iii. The important features of the PESYS expert system shell

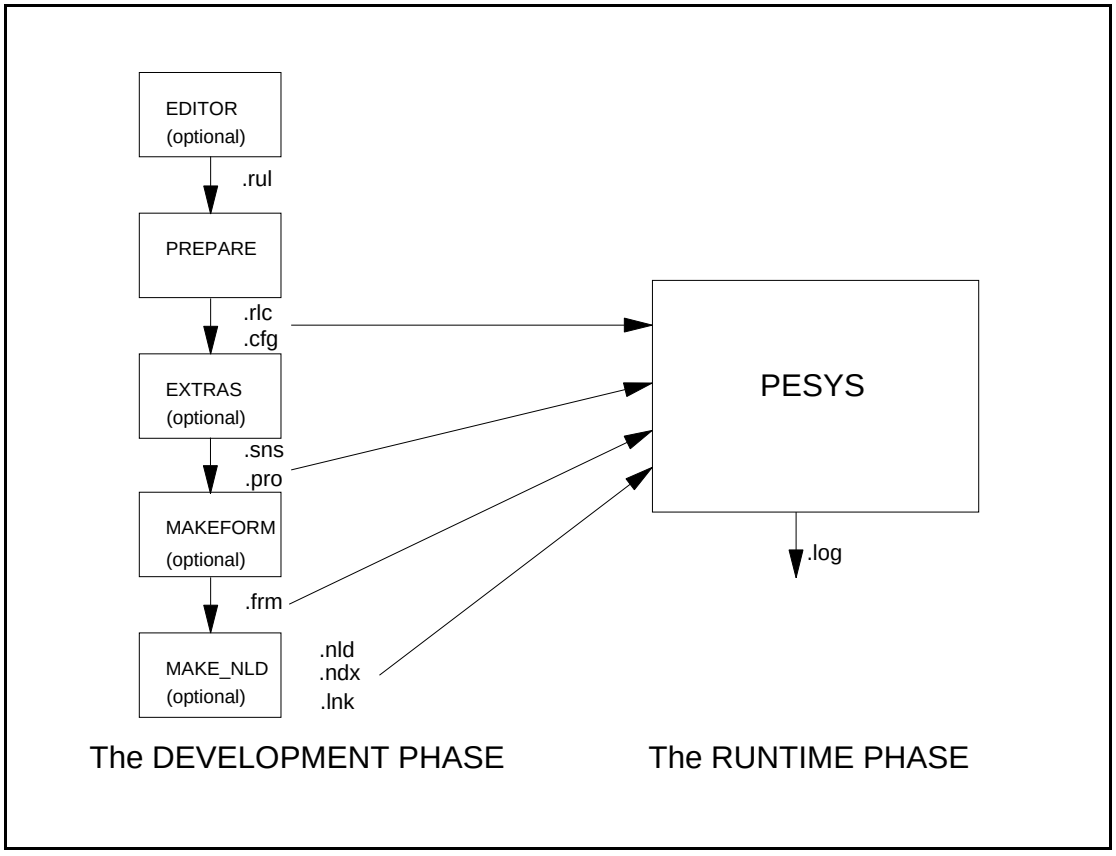


Figure 8.3 - The files created by the final PESYS system

The PESYS system provides a number of special features which can benefit the developers of expert systems. The developers of an expert system application have full control over the choice of inferencing techniques used by the expert system and the inference methods can vary between different knowledge bases. It is also possible to develop

applications that make use of a number of different sub-knowledge bases and each of these can make use of a different form of inferencing. Thus the main application may use backward chaining, whilst one sub-knowledge base may use forward chaining and another may use mixed chaining.

The inference engine provided by PESYS is very efficient in terms of speed and size and allows most of the available memory of the computer to be used to for the knowledge base and working memory of the system. This means that large applications can be developed with ease, especially if use is made of sub-knowledge bases. The performance of the system is such that it is possible to run practical applications using an IBM AT machine rather than the top-of-the-range machines required for many other expert system shells.

The user interface is consistent and simple to use, yet offers the functionality required for serious expert system applications. Any specialised input / output facilities can be provided by using external programs if necessary and the system provides a number of commands which can be used to improve the appearance of the system.

PESYS also offers a number of special features for knowledge base developers. The most important of these is the use of inform levels to mark both goals and general statements that will be of interest to users of the system. The use of inform levels means that problems of redundancy in the knowledge base, that can arise when the list of goals / hypothesis is kept separate from the rules used to arrive at them, are avoided in PESYS. This considerably aids the development of expert system applications.

A number of advanced structures, such as the use of or-clauses and atleast clauses, are provided within the knowledge base and these can also aid the writing of practical applications. In keeping with the aims of this thesis, the knowledge base is written in such a way that it can be easily understood and altered as necessary.

The special features of PESYS that relate to semi-formal domains have already been described in this chapter.

Applications developed using PESYS

The PESYS system has been successfully used by a number of researchers, other than the applications used to evaluate the solutions described in this thesis and these applications have all made use of various features of the PESYS shell.

One important application has been the use of PESYS to implement an effort estimation model for software development. By creating the model using PESYS it has been very easy to implement a version of the system which has been successfully demonstrated in a number of organisations. The application has also been used to analyze the results of a number of questionnaires that are based on this research.

PESYS has also been used to provide assistance with a piece of legislation, namely filing income tax returns. Many of the features of PESYS, such as the use of prompts and sub-knowledge bases where made use of in this application and the resulting system was far more effective than more conventional approaches to the particular problem.

Other applications of PESYS have investigated the ease with which it can be integrated with other software packages such as databases or simulation systems. These links have either been performed by using the expert system to drive the application or by having the application make use of the expert system. The open nature of PESYS has also been used to investigate its use for training and to examine the extent to which expert systems should be used to support rather than control users in particular domains.

e. FINAL SUMMARY AND FUTURE RESEARCH AREAS

The expert system described in this thesis has been developed to overcome a number of the most significant problems that can arise when expert systems are embedded in semi-formal domains. These solutions have arisen from an understanding of the nature of such domains that gives due consideration to the social factors that give rise to such domains.

In doing so, the research has emphasised the need to design for the likely problems of semi-formal domains, rather than simply responding to unexpected problems as and when they arise.

The thesis has shown the importance of encouraging the users to participate in the use of expert systems rather than accepting their output without question. One consequence of this approach, which would benefit from further research, is that the **process** of overcoming a problem situation is often more important than the **answer** that is obtained. By allowing the users of the expert system to examine the boundaries of the knowledge base, attempts have been made to encourage the users to understand the process involved in

arriving at an answer rather than simply accepting the answer provided. Moreover, users who examine this process, and combine it with their own considerations of the domain, are more likely to accept the results of the system.

In summary, therefore, this thesis has opened up the `black box' of expert systems in semi-formal domains by paying particular attention to the needs and skills of the users of such systems and allowing them to examine the boundaries of the knowledge base.

APPENDIX I - EXPERT SYSTEMS

I.1 WHAT IS AN EXPERT SYSTEM?

Doukidis and Whitley (1988) offer the following definition of an expert system:

> An expert system is a computer program that assists a user by providing information about a particular domain. It does this by manipulating information about the field that has been provided by a number of `experts' in the field. Another important feature of an expert system is that it has the facility to explain/justify the methods it used to provide that information.

I.2 WHAT DOES AN EXPERT SYSTEM DO?

In the course of everyday life we often come across problems. In some cases we have sufficient experience and understanding of these problems that we can solve them ourselves without any difficulty. Other problems, however, are beyond our capabilities and in these cases we turn to an `expert' who has experience in the problem field. For example, we may be able to cope with cuts and bruises but for more difficult problems such as serious illness we turn to the expert in this area, namely a doctor. Similarly we may be able to replace a tyre on a car but a faulty ignition will be repaired by a skilled car mechanic.

Expert systems are designed to assist non-experts with problems that arise in a particular domain when the expert is not available. They are also used to support experts by performing routine tests and informing the expert of any unusual circumstances. As well as trying to assist user with a particular problem an expert system will often try to solve the problem in a similar way to a human expert.

In order to solve these problems an expert system makes use of a formal representation of the knowledge of experts in the field. It uses this knowledge and combines it with particular details about the problem. These facts are normally entered by the user of the system. This combination of knowledge about the problem domain and the particular facts of the case allow the system to come up with a solution to the problem faced by the user. It can be seen that there are strong similarities between this approach and the human

situation it is trying to mimic, although some differences do exist mainly due to the formal nature of the computer based system.

I.3 WHERE DOES THE KNOWLEDGE FOR AN EXPERT SYSTEM COME FROM?

In order to solve problems for the user, an expert system must make use of knowledge about the problem domain. This knowledge is obtained through a process known as "knowledge acquisition" which is normally performed by a person known as a "knowledge engineer". There are a number of different ways in which the knowledge acquisition process can take place.

The most obvious method for doing this involves having the knowledge engineer ask the expert how she solves problems. This may be done through actively interviewing the expert. Alternatively it may be done by simply watching the expert as the problem is solved, perhaps asking the expert to say the things that she is thinking about at the time. Another technique that is used, especially when clarifying concepts in the domain, involves asking the expert to sort the concepts according to various criteria.

Another technique that can be used in the knowledge acquisition process comes from research into artificial intelligence and involves the use of induction. The expert is asked to list a large number of examples under a number of different headings and to classify them. For example, the expert may provide examples listed under weight, colour and taste and classify the examples as suitable for use or not suitable. The induction algorithm will then take all of these examples and use the principles of induction to find rules that will `explain' the decisions made in the examples. Whilst this technique suffers from the possibility of serious errors it often provides a good starting point for further discussion with the expert who is then often able to correct the rules and to provide other rules.

A final technique, that can be used in conjunction with the other two, is simply a good literature review. In many situations textbooks, guidebooks or company manuals will exist which can provide some of the knowledge used in the expert system.

I.4 FEATURES OF AN EXPERT SYSTEM

Figure 1.1 shows the typical architecture of an expert system. As can be seen from the diagram the expert system is made up of a number of distinct components, each of which contributes particular features to the functionality of the system.

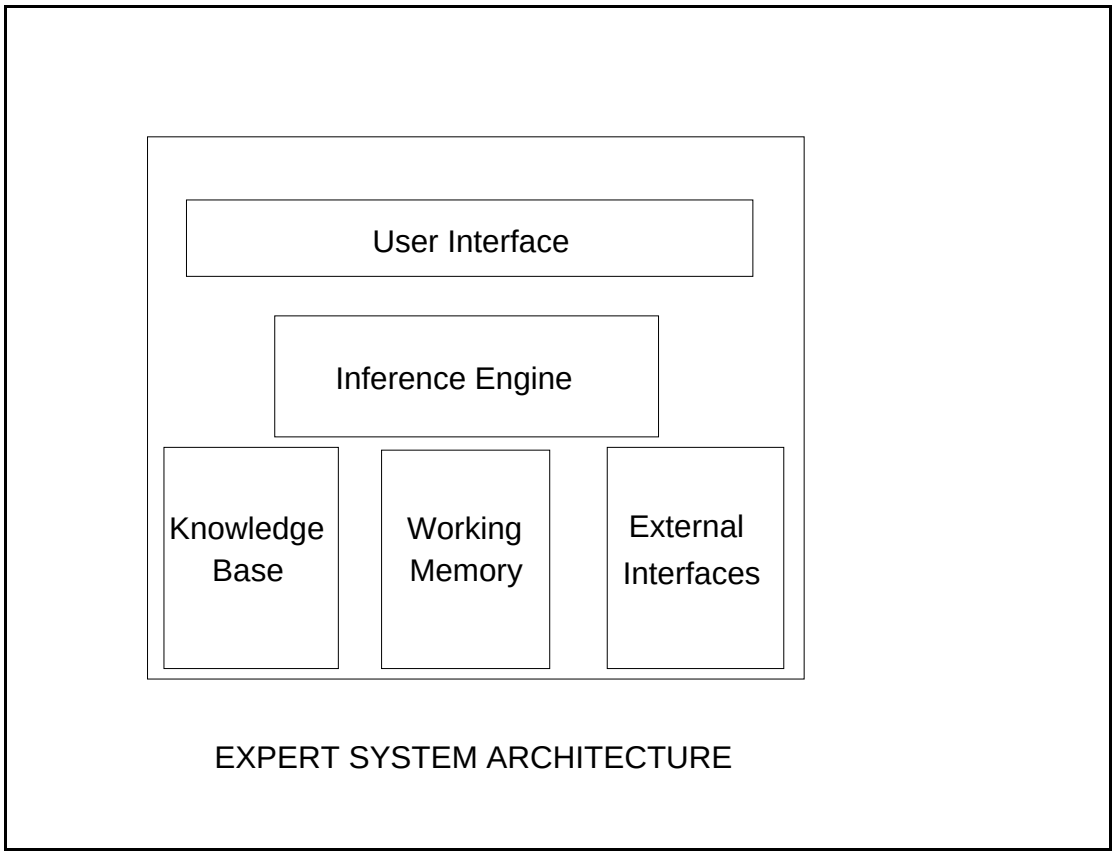


Figure 1.1 - The main components of an expert system

The dominant component of the expert system is the **knowledge base**. It is this part of the system that contains all the general knowledge about the problem area, the knowledge that was obtained through the process of knowledge acquisition described above. The knowledge base contains the formal representation of the knowledge of the expert, and the next section describes some of the common forms of knowledge representation.

The knowledge base, by itself, is as useful as a good book without a reader. Just as a book conveys no information until it is read, so a knowledge base is not able to solve any problems unless it is manipulated. The part of the expert system that manipulates the knowledge base is called the **inference engine**. The name derives from the logical inference techniques that are often used to manipulate knowledge representations. The inference engine makes use of the knowledge base in conjunction with particular facts about the problem. These facts cannot be stored in the knowledge base as they relate to one particular problem, rather the domain as a whole. A separate area of the expert system, known as the

working memory is set aside for this purpose. The working memory stores all the knowledge that has been arrived at in the course of a particular interaction. This includes the facts that are entered by the user as well as the facts that the inference engine arrives at.

Another significant component of the expert system is the **user interface**. This is the part of the system that the user comes into contact with. It is this component that asks the user questions about the domain, that presents the conclusions that the system arrives at and it is the part of the system that provides the explanation / justification of the results arrived at by the system.

In many applications the expert system must interact with the other software packages and even with sensors in the problem domain. To this end, the final component of the expert system comprises of external interfaces. These interfaces may be links to spreadsheets, databases or graphics packages. The system may make use of statistical or simulation packages or may take direct readings from the device it is monitoring.

I.5 HOW THE EXPERT SYSTEM WORKS

Production or if-then rules are the most common form of representing knowledge found in expert systems. These rules are made up of two parts. The first are the if-clauses which represent the conditions of the rule. The second part, the then-clauses, are the actions that are to take place if the conditions are met.

IF
the liquid is cool and dark
and the liquid is thick and frothy
THEN
it is probably dangerous to drink

The clauses in the rule may be simple `facts' that are either true or false, they may be statements that have a truth or `certainty factor' associated with them or they may even be `active' rules which contain pieces of code that are executed when a particular clause is examined. Thus a rule may have some facts in its if-clauses and if they are known to be true an active then-clause may take a certain course of action.

In addition to the rules in the knowledge base, it is common to find "goals" or "hypotheses" as well. These are certain facts that tell the system when to stop. As soon as

the system has determined that a goal is true, it does not need to perform any more actions - the inference process is complete. The use of goals enables two different forms of inferencing with the knowledge base to take place. One which starts from the data and sees which goals are arrived at, the other takes a particular goal and examines the facts needed to arrive at it.

Forward chaining, also known as data driven inferencing, takes a list of basic facts (which are stored in the working memory of the system) and then examines each rule. If all the if-clauses of the rule are known to be true (i.e. are found in the working memory) then the rule is said to `fire' and the then-clauses of the rule are either added to the working memory or are executed, depending on whether they are active or not. This process continues until either a rule is fired which adds a goal clause to the working memory, in which case the inference process can come to a halt, or until no more rules will fire, in which case the system must announce failure and halt.

The alternative strategy, known as backward chaining or goal-driven inferencing, takes a particular goal and sees which rules would have to fire for the goal to be added to the working memory. It then tries to fire this rule. This can only be done if all the if-clauses in the rule are known to be true. Three possible cases can arise, the if-clause is already known, the if-clause can be added by firing another rule, or no rule can be found that will add the rule to the working memory. In the first case the next if-clause is examined, in the second this clause becomes the new choice and the process is repeated with it. If, however, no rule could be found which would fire and add the clause to the working memory then the user must be asked about the clause directly. This process continues until a rule is fired which adds a goal to the working memory or until no goals can be inferred.

I.6 HOW AN EXPERT SYSTEM DIFFERS FROM OTHER COMPUTER SYSTEMS?

The most fundamental difference between expert systems and other computer systems lies in the fact that the inference engine, the part of the program that does the work, is kept entirely separate from the knowledge base, the data that it uses. This is in contrast to most conventional systems where the data and the programs that use them are inseparable. This means that the knowledge base can be rapidly altered without affecting the overall nature of the program. Indeed an inference engine may be used with a number of different

knowledge bases which could cover a diverse set of domains. The only limitation would be that the knowledge base conformed to the syntax expected by the inference engine.

Expert systems also differ from conventional programs because they operate on a symbolic rather than numeric level. The contents of the knowledge base are normally symbols - perhaps pieces of text or lines of code - rather than numbers. The problems that they tackle are often recursive or based on an indefinite number of elements. This is in contrast with most conventional data processing applications which manipulate pre-determined sets of numbers in a defined manner.

APPENDIX II - EXPERT SYSTEM DEVELOPMENT TOOLS

This appendix classifies and describes the advantages and disadvantages of a wide range of possible expert system development tools that can be used as the basis for the practical work described in the second part of the thesis. The reasons for the particular tool chosen for this thesis are also described. The main features of the chosen tool are described in Chapter 5.

II.1 A PROPOSED CLASSIFICATION OF EXPERT SYSTEM DEVELOPMENT TOOLS

There are many different kinds of development tool available for expert systems (Harmon *et al.* 1988). These range from highly specific development packages which only run on specialist hardware to conventional high and low level languages running on conventional hardware. In order to differentiate between these different kinds of tool it is common to use some form of comparative classification scheme. This section will discuss a number of such schemes, and will develop a scheme which diagrammatically depicts the tools available for the practical work undertaken in this thesis.

One popular form of classification is implicitly used when a new expert system development tool is reviewed and the product is compared with other packages that offer similar functionality or performance. This approach is used, for example, by Church (1989), Rajan (1988), Nuttall (1988) and Florentin (1987). A variation on this approach, which is commonly found in expert system journals involves a detailed comparison / evaluation of similar packages. In such a case the development tools are compared item for item, often by attempting to develop a simple application (Vedder 1989). Grouped evaluations make the classification of the tools chosen more explicit ("This article examines five different PC-based shells ..." (Vedder 1989, p. 28)). Comparative evaluations have been reported by Hinde *et al.* (1985), van Koppen and Philips (1986), Massotte *et al.* (1986), Puppe (1987) and Richer (1986) amongst others.

Although such evaluations offer useful insights into the functionality offered by individual and related software packages, they offer little indication as to how different

groups of tools are related. For example, how similar are artificial intelligence languages to expert system environments when it comes to developing actual expert system applications?

The Language-tool Continuum

Possibly the most common general classification scheme is the **Language-Tool Continuum** presented by Harmon and King (1985, p. 83). This classification scheme has high level languages which "are more flexible and more difficult to use to prototype a new system rapidly" at one extreme of a continuous scale and specialist development tools at the other. These tools are less flexible since many "knowledge engineering decisions have already been incorporated into the tools" (p. 83). Different expert system development tools are then classified by placing them on this continuum. **Figure II.1** shows the Language-Tool continuum.

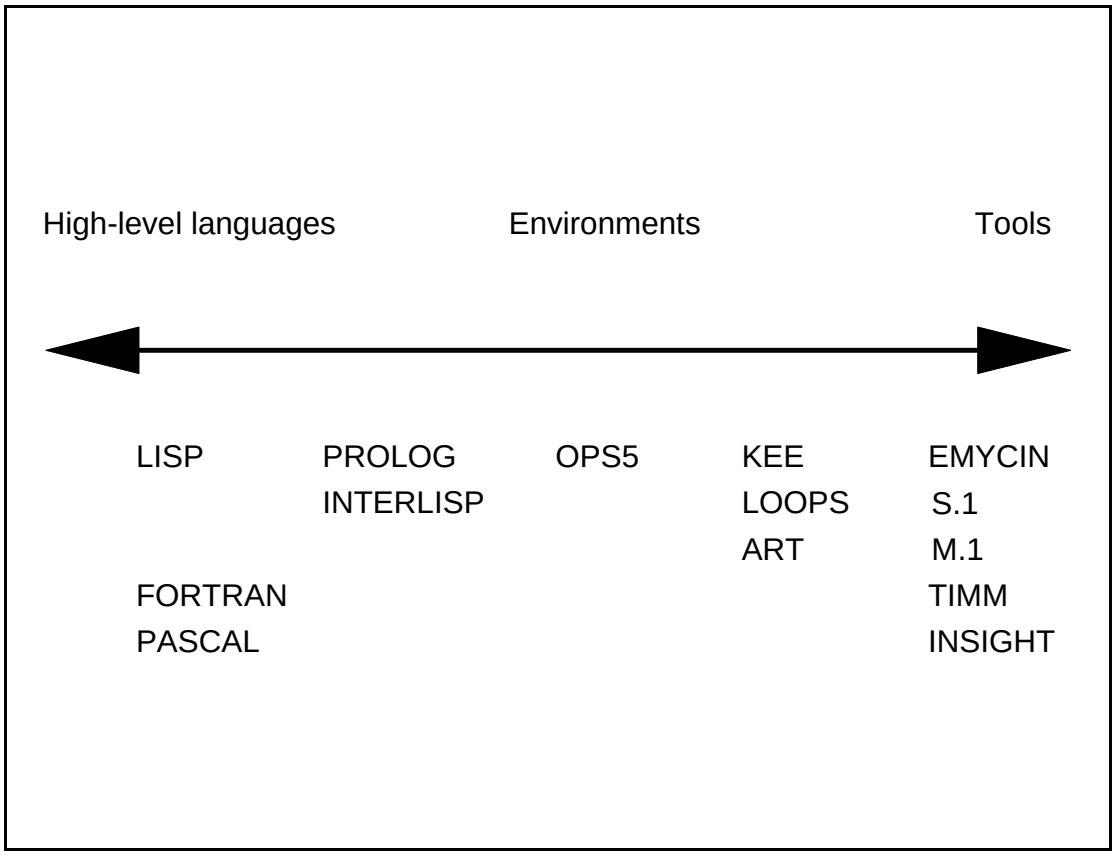


Figure II.1 - The language-tool continuum (Harmon and King 1985, p. 83)

A two dimensional classification

Rapid developments in the overall functionality of expert system development tools in the five years since the Harmon and King Continuum was introduced have severely limited the usefulness of that classification. Simply classifying tools according to whether they are languages or tools is no longer sufficiently discriminating to serve any useful analytic purpose. Instead a two dimensional classification is proposed, based on the two most important components of an expert system development tool. These are **control and inference strategies** and **data and knowledge representation**. It is proposed, therefore, that these factors are used to classify the various expert system development tools. Boley (1990) proposes a similar classification based on the expert system category (the control and inference strategies) and the domain dimension (data and knowledge representation). He suggests that these categories are interesting since they allow two complimentary ways of moving from individual expert systems to `meta systems' (see **Figure II.2**).

1) If the inference engine is kept fixed and the knowledge base is varied a meta system of expert systems for the same category in different domains can be constructed.
2) If the knowledge base is kept fixed and the inference engine is varied a meta system of expert systems in the same domain for different categories can be constructed.

Figure II.2 - The potential for meta-systems (Boley 1990, p.3)

Rather than have a continuum on each of the axes it is more appropriate to define three regions on each axis. These are **rapid development**, **easy to use** and **user-definable**. Thus, for example, a tool may have its own particular knowledge representation technique within the system which allows the rapid development of actual systems. The users simply provide the knowledge in this given formalism and the expert systems can be used almost immediately. Alternatively the tool may provide a number of different techniques which can be easily integrated in an application and hence the representation is easy to use. These techniques offer a number of different ways of representing knowledge and may therefore allow for a better model of the problem domain. The wider choice of representation techniques means that more attention must be given to deciding how to represent the knowledge and so system development is less rapid. Finally the system may offer user-definable representation techniques which allow the developers of the expert system to create representations that are closely related to the application in question. However, much

more effort is involved in creating new knowledge representation techniques and so system development will be much less rapid.

A similar classification exists on the control and inference strategy axis, whereby a system which provides a single inference strategy allows rapid development, whilst another system may offer a number of complimentary control and inference strategies which are easy to use for differing applications. Finally the tool may allow the developer to create control and inference strategies that accurately mimic the requirements of the domain.

Classifying the available tools in this manner offers a number of advantages. Firstly it shows the relationship between different kinds of tools, thus permitting alternative systems that offer similar degrees of functionality to be readily identified. Secondly, it can help guide the developers of applications to a particular part of the classification, based on the requirements of the chosen problem. For example, a project which must be implemented rapidly but which has fairly sophisticated knowledge representation requirements would benefit from using different tools from a problem which emphasises the need for accurate representation of domain knowledge over the rapid development of a prototype system.

II.2 ARTIFICIAL INTELLIGENCE LANGUAGES

Historically, the first expert systems were developed by researchers in artificial intelligence. Many artificial intelligence problems have a number of characteristics that are not found in conventional programming environments: they are based primarily on the symbolic representation of declarative knowledge and the manipulation of this knowledge rather than the storage and use of numeric data. Artificial intelligence problems are often recursive and make use of lists of indefinite length. These features are not found in many of the problems tackled by conventional languages and those languages offer very limited support for these features. Researchers in artificial intelligence therefore developed a number of languages that were specifically designed to support symbol manipulation, recursion and lists of indefinite length.

Two main approaches have been followed when developing artificial intelligence languages. Firstly there are those based on functional languages such as lambda calculus and, secondly, those based on first order logic. LISP is the prime example of the first approach and PROLOG is the most common example of the second.

II.2.1 LISP

LISP (LISt Processor) was one of the first artificial intelligence languages to be developed and is also one of the oldest programming languages still in use. It was developed by Professor John McCarthy and is based on the mathematical theory of **lambda calculus**. Lambda calculus describes the way in which input parameters to a function can be modified to provide a result (Ramsey 1988). This means that LISP is a functional language; every routine written in LISP is a function which returns a value.

There is only one data element in the LISP programming language and this is the **symbolic expression** (s-expression). An s-expression can either be an atom (a single element) or a list of atoms. An atom can either be a symbol or a number. A list of s-expressions may itself contain further lists of s-expressions. All the lists in LISP are enclosed within parenthesis (Winston and Horn 1984).

```
(THIS IS A LIST OF SYMBOLIC ATOMS)
(THIS IS A LIST (WHICH CONTAINS A LIST) AND OTHER ATOMS)
```

Figure II.3 - Examples of s-expressions in LISP

Since the only data element in LISP is the s-expression, this means that all LISP programs are also s-expressions; no distinction is made between programs and data. Programs can be manipulated as if they were data and vice versa. In particular this means that programs can create data structures that can be executed; effectively programs can automatically write other programs.

The LISP language is based on a few basic functions which perform tasks on s-expressions. The names of these functions are based on the first computer that the language was implemented on (Doukidis *et al.* 1988a) and are therefore not indicative of the tasks they perform. The function CAR takes the first element of a list. Note that due to the nature of s-expressions, the first element in a list may itself be a list.

```
(CAR '(A B C D))            returns A
(CAR '((A B) C D))    returns the list (A B)

(CDR '(A B C D))            returns the list (B C D)
(CDR '((A B) C D))    returns the list (C D)

(CONS A '(B C))             returns (A B C)
```

Figure II.4 - Basic functions in LISP

The function CDR takes the remainder of the list, i.e. everything except for the CAR of the list. The final function that forms the basis of LISP (apart from a few control functions that actually evaluate the other functions) is CONS. This function constructs new lists out of s-expressions.

In practice, however, most implementations of LISP have a far wider range of functions. These are built up in terms of the previously defined functions. One immediate effect of this small basic set of functions was a wide divergence in implementations of LISP. Many different versions of the language were available which provided different pre-defined functions. In many cases the functions were given different names or the ordering of the parameters differed between versions. To some extent this problem has been removed through the introduction of a standard - "Common Lisp". Many suppliers of LISP interpreters, however, still provide their own versions of the language and then also supply a library of functions that will allow their own versions to be compatible with Common Lisp.

DELETE is a typical LISP function, which may form part of a larger program, and it deletes the first occurrence of an item from a list.

```
(DEFUN DELETE (ITEM LST)
        (COND ((NULL LST) NIL)
        ((EQL (CAR LST) ITEM) (CDR LST))
        (T (CONS (CAR LST) (DELETE ITEM (CDR LST))))))
```

Figure II.5 - The LISP function DELETE

LISP supports the creation of many different forms of knowledge representation and allows the programming of many different inference and control methods. In many cases, they were originally developed in LISP.

II.2.2 PROLOG

LISP is very much the dominant programming language for artificial intelligence research in the United States. In contrast, PROLOG is very popular in the United Kingdom, the rest of Europe and Japan. It is based on a restricted form of first order predicate logic and makes use of the analytic power of logical inference. It is a joint French / British development with much of the initial research being undertaken at Edinburgh University although it has also been heavily promoted by Imperial College and Professor Bob Kowalski.

One of the simplest forms of logical reasoning is based on **propositional logic**. In this form of logic the complex sentences that are used in the reasoning process are broken down into propositions. Each proposition may either be true or false. The simple propositions can then be combined, using various logical operators, to arrive at the truth of the combination. The most common logical combinators are AND, OR, NOT and IMPLIES. Each logical operator has a **truth table** associated with it, showing how it combines the truth or falsity of its component parts to arrive at a final result. If a combined sentence is always true, no matter what the values of the constituent parts, then it is known as a **tautology**, whilst if it is always false, no matter what the truth of the basic elements is, then it is a **contradiction**. All other combinations are **contingent** on the truth or falsity of the basic values; in some cases the combined sentence is true, in others it is false.

All IBM computers have keyboards	(P)	
IBM-AT-1 is an IBM computer		(Q)
Therefore IBM-AT-1 has a keyboard	(R)	

Figure II.6 - A simple argument in propositional logic

Consider the argument shown in **Figure II.6**

If this is represented using propositional logic then the basic propositions are "All IBM computers have keyboards" (represented by P), "IBM-AT-1 is an IBM computer"

(represented by Q) and "Therefore IBM-AT-1 has a keyboard" (represented by R). The form of the argument is P and Q implies R. Intuitively the reasoning is valid, yet propositional logic is not able to demonstrate this.

First order predicate logic is, however, able to demonstrate this and makes use of the notion of **predicates**. Predicates in the previous example may include "Is_an_IBM_computer" and "Has_keyboard". It is possible to specify the clauses of the argument in this predicate form. For example, the first clause is effectively stating that "for all things x, if x is an IBM computer then this implies that x has a keyboard". Stated more formally this is:

∀x(is_an_IBM_computer(x) -> has_keyboard(x))
is_an_IBM_computer(IBM-AT-1)

Therefore has_keyboard(IBM-AT-1)

Figure II.7 - The same argument in predicate logic

Various rules of valid inference have been developed for first order logic that can be used to determine whether a conclusion can be validly inferred from a particular set of premises. The language PROLOG is based on a subset of first order logic and supports many of the features of logical reasoning.

PROLOG takes a different approach to programming to that found in most conventional languages. These languages are procedural in that the programmer specifies how the computer is to solve the task; the procedures that are to be followed are specified step-by-step. In contrast declarative languages, such as PROLOG, require the users to declare what the problem is and to specify constraints that must be satisfied. These are written using predicate logic clauses. The `execution' of the program is performed when the users enter a query. The programming language then tries to solve this query whilst satisfying the declared constraints. Any values which satisfy the query are then displayed as the results of the program (Black 1986).

For example, the following facts can be declared as the constraints of a problem:

```
Likes (John, Mary)
Likes (John, Jane)
Likes (Richard, Amanda)
Likes (Simon, Cathy)
Likes (Sarah, John)
```

Figure II.8 - Sample facts in PROLOG

and the query Likes (John, X) entered with the system returning all the values of X that satisfy the constraints of the program.

Mary

Jane

Further constraints can be added to the program, for example, it is possible to specify that if X likes Y then Y likes X:

```
Likes (X,Y) :- Likes (Y,X)
```

On querying the system again, the value Sarah would also be returned as a value for Likes (John, X).

In theory the entire programming process is performed through the writing of constraints with the programming language doing all the work. This naturally means that the performance of the system depends crucially on the efficiency of the implementation of the language and various techniques have been developed to improve this area (Ramsey 1988). In practice, however, even programmers using efficient PROLOG interpreters will often add pieces of procedural code to improve the performance of their programs (Black 1986).

A possible limitation of PROLOG arises from the form of logic upon which it is based. The facts stored in the PROLOG database are all assumed to be true and it is therefore rather difficult to store false facts in the system. Another implication of this occurs in the notion of "negation as failure". When the system is trying to prove a certain query it tries to prove the negation of the query. If the negation of the query is not proved then its opposite (i.e. the query) is assumed to be true. This process can only operate successfully if the "closed world assumption" holds. In a closed world, *everything* that needs to be known to

determine a query, is known. Therefore if the negation of the query could not be proved then this was not the result of insufficient information and its opposite *must* be true.

A standard PROLOG interpreter or compiler offers a number of knowledge representation techniques, based on predicate clauses and lists. The language, however, offers less flexibility in the control strategy it uses (although it is possible to overcome this difficulty).

II.3 LANGUAGES SPECIFICALLY DESIGNED TO DEVELOP RULE BASED SYSTEMS

One problem with using languages such as LISP or PROLOG to develop expert systems is that while they do offer a wide range of possible methods for implementing knowledge representation and control methods, none of these facilities are readily available in the language. Instead, it is often necessary to expend a considerable amount of effort in developing routines to implement them before any practical systems can be considered. As a result of this problem, programming languages have been developed that provide the basic features required to develop expert systems which often lie "on top of" versions of LISP or PROLOG.

A typical example of such a language is OPS5 ("Official Production System version 5'). OPS5 adds extra functionality to the LISP language, functionality which is directly related to the development of rule based systems. In particular OPS5 supports the creation and use of frame-like objects. An object can have many identified attributes, however since OPS5 is implemented in LISP there is no strict type checking on the contents of slots (Brownston *et al.* 1985). To define an object the command Literalize is used.

```
(Literalize Person Name Age Weight Hair_colour Hobbies)
```

To add values to this object, a prefix operator (^) is used which refers to the individual slots:

```
(Person ^Name John ^Hair_colour Brown)
```

To retrieve values from the list of facts, the prefix operator is again used and the system supports multiple objects that match a particular test. For example, it is possible to select all the people with brown hair using a single command.

(Person ^Hair_colour Brown)

The feature of OPS5 that makes it particularly suitable for developing expert systems is that this selection of multiple objects can be used as part of the conditions and actions of production rules. For example, it is possible to select all the objects whose size is greater than 50 and mark them as requiring special assistance:

IF (Object ^Size) > 50 THEN (Object ^Attention Special)

Some applications, however, do not require all the possible objects to be selected and altered and so OPS5 offers a number of useful conflict resolution strategies to choose between objects. For example, the objects may be selected according to the time since they were last used or the frequency of their use. Debugging facilities are provided to support the development of large scale systems.

OPS5 offers easy to use knowledge representation techniques since it is possible to create representations that match the particular problem being tackled. The choice of conflict resolution strategies provided by the system also suggest that the control and inference strategies are easy to use. Until recently OPS5 type languages have been restricted to applications developed on specialist hardware. However versions of the language, for use on IBM Personal Computers, have recently been introduced.

II.4 CONVENTIONAL PROGRAMMING LANGUAGES

The languages described above were designed with features that particularly match the requirements of problems covered by artificial intelligence research. This does not mean, however, that other programming languages cannot be used to tackle these same problems. In many cases, it is possible to provide the facilities offered by artificial intelligence languages using more conventional languages.

II.4.1 High level languages

An increasingly popular choice of development tool for the creation of expert systems are high level languages. These are languages such as PASCAL, C, FORTRAN, COBOL, PL/1 and BASIC which are commonly found in business environments. Santene (1989) notes that most financial applications of expert systems are implemented in the COBOL and BASIC programming languages. Perhaps most interestingly, expert systems concepts are extending the range of possible applications that companies can tackle, whilst still using their conventional hardware and programming languages. Expert systems techniques are allowing these companies to view old problems from a new perspective, a perspective which allows solutions to be developed. In a recent survey of the use of artificial intelligence techniques by Operational Research practitioners, Doukidis and Paul (1990a) found that conventional high level languages made up a quarter of the tools used to develop artificial intelligence and expert systems applications in OR departments.

The use of conventional languages offers many advantages to the developers of practical expert systems. High level languages have been available for a long time and their compilers or interpreters are now very well understood and tested. This means that the code they produce can be reasonably assumed to be `correct'. In addition, much work has been done to optimise the resulting code, either in terms of code size or in terms of runtime speed. Many compilers now support integrated development environments, combining editors, compilers and debuggers which can immediately highlight the source of any errors as they occur: at compile time (syntax or semantic errors) or at run time (where the errors are likely to be based on incorrect program logic) (Borland 1987).

Another feature for any serious, large scale applications arises from the fact that the company that has provided the high level language is likely to have been in existence for a long period of time and will have developed a good working relationship with the users of its products, a relationship which implies trust. Few, if any, data processing departments will be willing to trust a major business application to software that has been developed by a newly formed company with no track record of service and support (Butler *et al.* 1988).

High level languages run on conventional hardware platforms and often support full links with other software packages such as the main databases of the organisation. Whilst it is not impossible to link artificial intelligence languages to existing software packages it is often a rather difficult process. Artificial intelligence languages often require specialist

hardware in order to run effectively and this may again cause problems for conventional data processing departments as this will be an extra and different piece of hardware to maintain.

Most organisations will also have made a considerable investment in traditional hardware and software, and will have used many resources in training their staff to be familiar with this hardware and software. This is yet another reason for the popularity of conventional high level languages for the development of expert systems (Butler *et al.* 1988).

High level languages do not have the basic symbol and list manipulating routines that are commonly required for expert system applications and this means that routines must be devised to implement the required functionality before actual expert systems can be developed and hence high level languages are found in the user-definable regions of both axes.

II.4.2 High level languages with pre-written modules

In a similar manner to OPS5 type systems, versions of high level languages have been developed which offer pre-written modules that implement the low-level artificial intelligence functionality that is not already part of the languages. These may be extended to general libraries of routines which can simply be included and used in expert system programs. These libraries contain routines that implement various knowledge representation techniques and different inference methods.

An example of this approach is ASPES (A Skeletal Pascal Expert System) described by Doukidis and Paul (1987). As its name suggests, ASPES provides libraries of routines that can be used to create full scale expert system applications. Many expert system development tools are based on existing expert systems and ASPES is no exception. It differs from many other tools since it is not simply an existing expert system without a knowledge base. Instead ASPES provides libraries of routines that support the various tasks of an expert system such as knowledge representation, inference and explanation. Since these tasks are implemented in libraries they can be used or replaced as necessary.

ASPES was developed at the London School of Economics and Political Science for use in teaching and research. The choice of programming language was therefore determined by the predominant language used for teaching in the school, namely PASCAL. Specialist artificial intelligence languages such as LISP or PROLOG are not as well supported as the more commonly used high level programming languages, they are expensive to purchase and

new researchers often need to learn a new programming language in order to use them (Doukidis and Paul 1987).

In order to provide the functionality required for programming expert systems, such as the manipulation of lists of objects, a number of routines must be developed and these are provided in one of the ASPES libraries. The use of the predefined modules is not compulsory and APSES can be used for domains other than those explicitly supported by the provided routines by making use of other libraries.

ASPES has been widely used for teaching and research at the school and has been used as the basis for a number of existing expert systems. One such system is the PASCAL DEBUGGING AID described by Doukidis *et al.* (1988b). This expert system attempts to help students who have problems learning to program in PASCAL. The expert system incorporates some ideas from **intelligent tutoring systems** which were easily incorporated into the expert system developed by ASPES.

A second application developed using ASPES is SIPDES, a SImulation Programming Debugging Expert System, which provides advice on program errors that arise when using a simulation program generator (Doukidis and Paul 1990b). Students create models based on the three phase simulation structure (Crookes *et al.* 1986) and use an interactive simulation program generator to generate a PASCAL program based on their model. Errors may be introduced into the program at this time and these may occur at run time when, for example, an attempt is made to move an entity from an empty queue, or an entity may disappear completely over time. The basis for the SIPDES program was again ASPES and the structure of ASPES meant that it was possible to incorporate further routines into the expert system that provide extra assistance in a format suitable for simulation modellers.

II.5 OBJECT ORIENTED LANGUAGES

Object oriented languages have become very popular for use in artificial intelligence applications in recent years. These languages, such as SMALLTALK and C++, take a different perspective on the programming task. Conventional programming languages are based primarily on procedures through which data objects pass. This normally means that separate routines must be written for different data types: it is not easy to re-use pieces of program code. Object oriented languages, in contrast, consider the data objects to be the primary part of the program and the program itself consists of messages being passed between these objects.

When a particular object receives a message it checks to see whether it has a procedure for undertaking the action described in the message. If it does, then the action is performed, if it does not then the message is passed up to a more general type of object to see if this object can perform the action. This process continues until the message is acted upon.

This can be demonstrated by comparing how the statement **3+4** is handled in a conventional and object oriented language. In a conventional high level language the expression is evaluated by a special part of the program that contains routines for calculating the values of expressions. In object oriented programming, however, the message **+4** is sent to the object **3**. This object takes the first part of the message (the plus sign) and sees if it can perform the requested task. In this case it cannot, so the message is passed to the more generic object, number. This object contains routines for dealing with the message **+** and uses the rest of the message to perform this task.

Object oriented techniques have been widely used in graphical user interfaces, such as that found on the Apple Macintosh, since messages about whether the mouse button has been pressed can be passed to a particular icon which will act on it. For example, a menu item or data file will know what actions are to be performed when a mouse button pressed message is passed to it, such as selecting the highlighted item or running a particular application.

In expert systems such techniques may be useful when developing causal networks made up of many components which are interconnected and send messages to one another. Object oriented techniques are also available in some advanced versions of Common Lisp.

II.6 EXPERT SYSTEM SHELLS

Expert system shells are pre-written expert systems that do not have any particular knowledge base, instead the developers of an expert system can use any knowledge base with the shell provided it follows the syntax used by the shell. This means that it is possible to develop applications very rapidly. In addition, most shells provide a number of useful features for developing applications such as editors, knowledge base checkers and debugging facilities.

One interesting feature about the use of expert system shells, at least in the United Kingdom, is the predominance of `home produced' systems (D'Agapeyeff and Hawkins 1987). This apparent `patriotism' is heavily influenced by the low cost of British expert system shells which are designed to run on conventional hardware systems, often IBM

personal computers. Their low cost in comparison to American workstation based products means that they are more acceptable to risk-averse UK management than their competitors (D'Agapeyeff and Hawkins 1987). Having localised user-support is another influential factor (Bodkin and Graham 1989).

II.6.1 XI+

One popular expert system shell is Xi+, developed by Expertech Ltd., a British firm based in Slough. Expertech were one of the first companies to provide pre-written knowledge bases for use with runtime versions of their shell. These knowledge bases, which cover areas such as dismissal law and maternity leave, were designed to be used "off-the-shelf" by companies who wanted advice in these areas.

Xi+ operates by trying to satisfy a query presented by the users. This is done by using a number of knowledge bases which contain rules or the "know how" entered by an expert. Xi+ allows knowledge bases to be structured so that, for example, one knowledge base can call up another one, encouraging top down, structured programming. The system allows the creation of help screens, reports and forms to aid the users of the final application. It is also possible to interface Xi+ with other applications such as databases and spreadsheets (Forsyth 1987).

The system operates by asking the users questions about the domain. These can be questions that have YES/NO answers and also questions that allow the users to select values from a list of possible values. Some of these questions may allow only one value, whilst others allow the users to enter a number of different values. Once the query has been satisfied the users are able to perform a what-if analysis, perhaps by volunteering extra information.

Xi+ was previously implemented using a version of MICRO-PROLOG and this caused the system's performance to be rather slow. The current version (release 3) is implemented directly in the C programming language and so operates rather more quickly (Church 1989).

II.6.2 LEONARDO

Leonardo, produced by Creative Logic, a British company located in the Brunel Science Park, is another expert system shell. It differs from Xi+ in that it offers a particular form of knowledge representation based on frames in addition to the production rule formalism found in Xi+. There are three versions of Leonardo which offer differing amounts of functionality in knowledge representation. The second level of the product offers better control of inheritance and level three offers sophisticated routines for the use of certainty factors using techniques based on fuzzy logic.

Everything that is mentioned in a Leonardo knowledge base has an associated frame. These frames contain slots which may contain actual values, default values, inheritance links or even program code as Leonardo has a small procedural language which is used for those parts of an application that cannot be supported directly using inference techniques (Forsyth 1988).

Leonardo also has a sophisticated forms facility that is directly related to the frame based representation of knowledge. Thus it is possible to have pieces of code attached to the slots of a frame which determine how the users are to interact with the form.

The system is implemented in FORTRAN and is available on a wide range of hardware platforms ranging from IBM personal computers to mini-computers. An application created on one platform can easily be re-implemented on other platforms that support the product (Roth 1988).

II.6.3 CRYSTAL

A third expert system shell is CRYSTAL, developed by Intelligence Environments based in London. In contrast to the functionality provided by shells like Xi+ and LEONARDO, CRYSTAL has a very limited number of options available, for example, it only supports backward chaining (Wallsgrove 1988). However, it makes up for this by providing a very fast inference engine.

CRYSTAL includes a number of facilities for improving the screen display of the system including the use of graphics, forms and questions. It is possible for CRYSTAL to access data from other programs such as dBase. The program can also read text and numbers that are stored in ASCII format. Full calculations, including financial and boolean analysis are provided by the system.

The rules in CRYSTAL are written in a slightly unusual format: "These conclusions are true IF these conditions are true". The if statements may be combined using AND and OR. Rule statements may contain system commands and it is possible to interface the system to other packages by making use of the C programming language (Linderholm 1987).

The CRYSTAL expert system shell (version II) was used to develop the expert system for the Latent Damage Act described by Capper and Susskind (1988). They chose the shell because the performance of the system was not adversely affected by the number of rules that was entered into the knowledge base, because the system supported the rapid development of prototypes and because the software was available in a runtime only version, allowing them to widely distribute their knowledge base at minimal cost.

II.7 EXPERT SYSTEM TOOLKITS

Expert system toolkits were conventionally implemented on specialist hardware, normally LISP machines. They make use of high resolution graphics screens and sophisticated windowing techniques. These toolkits are normally written in LISP and offer many different knowledge representation techniques. The high cost of the underlying hardware originally needed to run expert system toolkits means that expert systems were often developed using such tools, making use of the sophisticated editing, tracing and debugging facilities, and were then re-implemented on a more conventional platform.

II.7.1 ART

Art, developed by the American company Inference Corporation, is a rule based system that is based on the OPS5 language. It offers the knowledge engineer frames, logic programming and LISP for the development of applications. It also supports assumption based reasoning, i.e. non-monotonic logic and time dependent reasoning. These logical dependencies allow the system to handle constructs like:

While A is true, B is true

It is also possible to create different viewpoints and competing worlds. The system supports the creation of different models of the world depending on which assumptions are

used. This allows for comparison between the implications of different views (Grégoire 1988).

The frame based representation supports multiple inheritance but does not support active values and constraints. The performance of the system is improved by pre-compiling the knowledge base and hence more advanced frame operations are not supported. A distinction is made between inference rules - which add facts and objects to the working memory of the system - and production rules - which alter the values of objects in a similar manner to OPS5.

The system is written in C and is very fast in operation, aided by the knowledge base compilation. The limited support for maintenance and control suggests, however, that there may be problems developing large scale applications using Art.

II.7.2 KEE

The KEE toolkit is developed by the American company Intellicorp. It also uses frames for storing information about object taxonomies, but frames are implemented more effectively in KEE. For example, it is possible to include active values and methods (procedural code) in the frames. It is even possible to represent rules within the slots of a frame (Grégoire 1988).

The system is written in Common Lisp and is therefore slightly slower than Art although this allows the knowledge base designer to make direct calls to the underlying LISP language and user-defined conflict resolution strategies are permitted (Florentin 1987).

KEE uses object oriented techniques within the knowledge base and these are also used to drive the screen displays. Thus it is possible to represent a dial on the screen and as the setting on the dial is altered, the associated value in the frame is changed accordingly.

II.8 THE CLASSIFICATION OF EXPERT SYSTEM DEVELOPMENT TOOLS

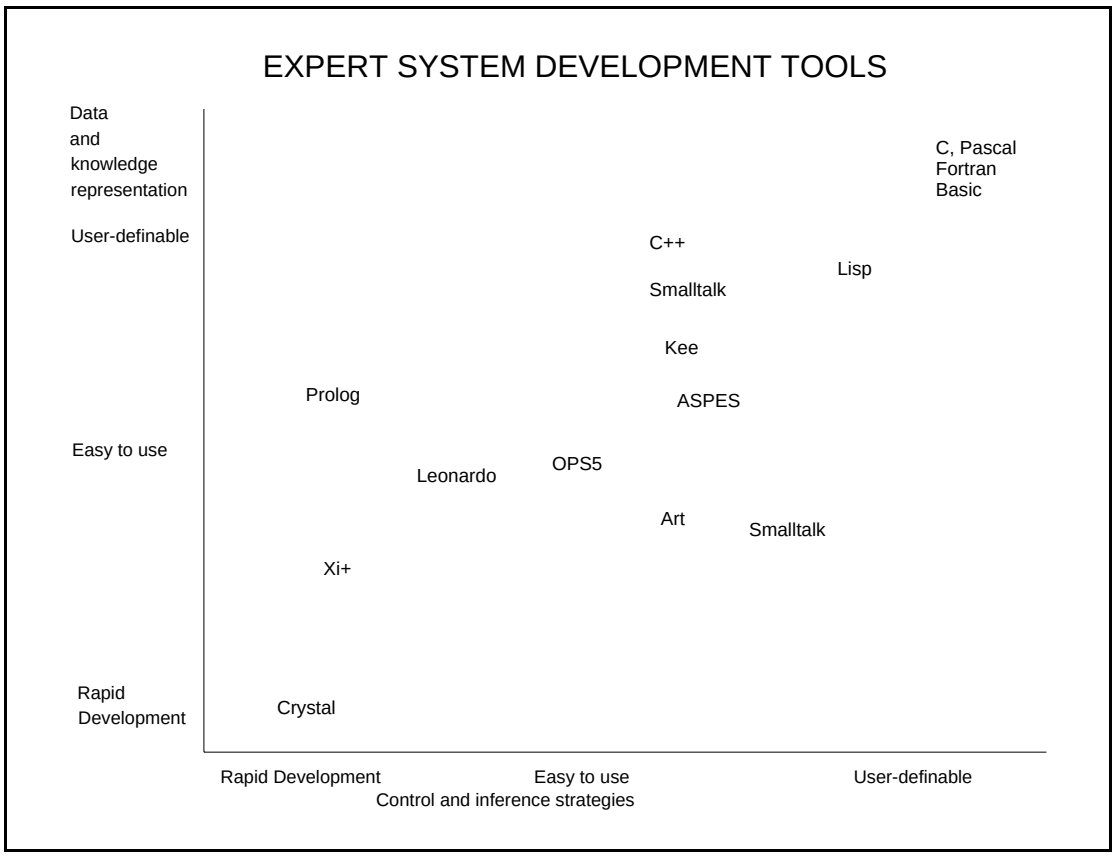


Figure II.9 - A classification of expert system development tools

Figure II.9 uses the classification of expert system development tools introduced in this appendix to illustrate the various tools described above.

II.9 THE LIKELY FUTURE DIRECTION OF EXPERT SYSTEM TOOLS

A number of trends in expert system development tools have been predicted (see, for example, Land *et al.* 1988). These include the creation of generic systems and the incorporation of intelligent elements in conventional data processing systems. Generic systems are designed to tackle problems in a "domain of general interest". The most common areas of such general interest are legislation and financial management and Expertech have marketed runtime versions of the expert system shell Xi+ with pre-written knowledge bases on areas such as maternity leave and employment law.

Another significant trend is the move to combine artificial intelligence functionality with conventional data processing systems. Waller (1989) suggests that most current expert

system development tools have either a high artificial intelligence content or a high data processing content. Those with a high AI content may support production rules, inheritance and sophisticated frame based representations. They may also make use of object oriented programming. In contrast tools with a high DP content are able to access other high level languages and existing applications such as databases and report generators. Waller predicts that in the future these two components will tend to become equally important.

One of the main reasons for this integration of the two components is the potential size of the data processing market. The personal computer market for expert system development tools is stagnating and hence developers are looking for new areas where their products can be sold. The data processing community is such a market (Waller 1989). In particular, data processing departments normally have large budgets and plan over longer time periods than many user-departments. Data processing departments are now required to tackle more complex problems, problems which may well be suitable for applying expert systems techniques and they also need to be more flexible in their approach to changing circumstances. In both these areas the successful use of expert systems tools can prove to be beneficial (Land *et al.* 1988).

II.10 THE CHOICE OF HARDWARE

The choice of which development tool is to be used depends on the choice of hardware platform that it is to be implemented on. Broadly speaking there are two main types of hardware available for expert system development: specialist hardware which has been developed to specifically run artificial intelligence languages and conventional hardware platforms. The advantages and disadvantages of these two categories will now be discussed.

II.10.1 Specialist hardware

Early research in artificial intelligence was often hampered because the hardware being used to implement versions of LISP and PROLOG was not particularly suited to many of the basic tasks required by these languages such as garbage collection and resolution. This meant that the languages were relatively slow in operation. As a result of this, research was undertaken to develop specialist hardware which directly supported AI languages.

The first commercially available specialist hardware platforms were **Lisp machines**, so called because they were based on central processors which had been specifically designed and optimised for implementing the LISP programming language. Since they were designed for running LISP programs, the specialist hardware platforms (such as the Xerox 1100, Symbolics 3600 and Texas Instrument Explorer workstations) offered considerable increases in the run time efficiency of the programs. Lisp machines normally make use of high resolution graphics screens and windowing interfaces. They also have tracing and debugging facilities that simplify and support the process of program development. The early arrival of Lisp machines is, in part, due to the dominance of LISP as the programming language for artificial intelligence research in the United States. The alternative European language, PROLOG, is gradually following the same path with ICL, for example, launching a **Prolog machine** based on a central processor specially designed to support the PROLOG programming language.

The sophistication of Lisp and Prolog machines, both in terms of their specialist processing chips and the software environments they use, together with the limited market for such machines, means that they are rather expensive. This has a number of important implications. Firstly, it means that many universities do not have access to such machines (in particular the London School of Economics and Political Science does not have such a machine) although it is often possible to share resources with other universities or colleges. Secondly, it means that few outside organisations have access to such hardware either. This suggests that even if development work was undertaken on such machines, when the applications developed are used in real life situations they will need to be converted to different hardware platforms. It is interesting to note that many of the software packages that were originally developed to run on Lisp machines are now being converted to run on more conventional hardware such as Sun workstations and top-of-the-range IBM personal computers (Waller 1989). Specialist hardware such as Lisp machines may also prove difficult to interface with conventional hardware systems. Thus before the software can be tested with the existing data processing facilities in an organisation it is necessary to convert it into a form that can be executed by this hardware. If this is anything other than an automatic process, considerable constraints will be imposed on the use of prototyping in the development process.

II.10.2 Conventional hardware

The LSE does not have specialist hardware available for the development of expert systems and therefore the development work would have to be done using conventional hardware. Four possible platforms were available for this research work and they will now be described.

The LSE has two **mini-computers** available for use by staff and students. These are both Digital Equipment Corporation Vax 11/780 mini-computers operating under VMS. Access to these machines is through terminals available around the school. The programming languages available include PASCAL, C and FORTRAN. Artificial intelligence languages such as LISP and PROLOG are not available. This central computing facility has been optimised to support statistical packages analyzing large data sets performing computationally intensive tasks such as regression analysis. This results in a rather poor performance for programming tasks, both in compilation and execution. It was expected that there would be many problems of portability if the system had to be demonstrated at other locations. The Vax systems were upgraded and replaced by a single Vax 6330 in the summer of 1989.

The Information Systems department has a Sun 3/160 **workstation** for use by research students. It runs the UNIX operating system and despite being a multi-user system only two dedicated terminals are available. At the current time the system only supports the (pre ANSI standard) C programming language. Access to the machine is limited by the number of terminals available and there may again be problems associated with demonstrating any software developments in other organisations.

The LSE has decided on two standards for **microcomputers**. The first standard is the IBM PC running the MS-DOS operating system. The school currently has three public access rooms containing, in total, 80 IBM personal computers. Additionally the Information Systems department has a number of machines available for use by research students. Versions of the LISP and PROLOG artificial intelligence languages are available, in addition to the PASCAL, C and MODULA 2 programming languages.

As well as providing a number of programming languages for use in the development of expert systems, the IBM PC has become one of the de facto hardware standards for microcomputers and this means that it is the most common platform for the developers of expert system shells. Most of the available shells have, as a minimum, MS-DOS versions of their software. When taken in combination with the large numbers of DOS machines on the market this means that the unit price of much of this software is far lower than software for

more specialised hardware. The relatively low price of such software meant that the department was able to purchase copies of some of the more popular expert system development tools available.

The second microcomputer standard adopted by the LSE is the Apple Macintosh. When this research began there were no Macintosh computers available for student use. However one year into the research approximately thirty machines were made available for students to use. Although programming languages are available for the Macintosh, none are available on the public access machines (except for Hypercard). The popularity of the machines, especially when running software that takes advantage of the graphics facilities offered, such as word processing and desk top publishing, means that access to the machines available is often limited. Later in the second year of this research a number of Macintosh computers were obtained by the Information Systems department for use by research students and the PASCAL programming language was made available on them.

Figure II.10 summarises the hardware platforms that were considered.

Lisp Machines
Prolog Machines
Mini-computers (Vax 11/780)
Workstations (Sun 3/160)
Microcomputers (IBM PC / Apple Macintosh)

Figure II.10 - Hardware platforms for expert systems development tools

II.10.3 The choice made

Very early on in the research a choice was made to use the IBM PC hardware environment as the basis for the development work. The reasons for this choice will now be described.

The **availability of machines** was an overriding factor in the choice of hardware environment. The school and department have a large number of IBM PCs available for use and it was possible to gain access to a machine as and when necessary. Although the Apple Macintoshes and Sun workstation have better graphics and a far simpler user interface, access to these machines is severely limited, posing a major constraint on the research being undertaken. Although access to the Vax computers was also widely available, the relatively

poor performance when running non statistical packages would again influence the research undertaken.

There is also a greater **availability of software** on the IBM PC environment as it supports reasonably priced expert systems software in addition to a wide range of programming languages. Both artificial intelligence languages (such as LISP and PROLOG) and more conventional programming languages (such as PASCAL and C) are available on IBM personal computers. Furthermore, the large number of machines (and compatibles) means that considerable effort has been undertaken to write efficient compilers and interpreters for these languages and they are generally available at a significantly lower cost than software developed for specialist hardware.

From the outset of the research it was hoped that various versions of the software could be shown to various other organisations. For example, the software may need to be demonstrated at conferences or used in seminars and industrial presentations. Therefore the **widespread use of the hardware in other organisations** was another important factor influencing the choice made. Since IBM computers are widely used all that is often required is the provision of suitably formatted floppy diskettes containing the appropriate software.

On those occasions where no machine was available for use it would still be possible to use a portable IBM compatible machine to achieve the same results. Portable Apple Macintoshes are also available and these would need to be used more often since fewer organisations make use of Apple computers.

Mini-computers and workstations are not widely used and therefore the opportunities for demonstrating the system would be severely limited. On those occasions when comparable hardware would be available, there would still be significant problems associated with transferring programs and data.

In addition to software that can be used to develop expert systems, IBM personal computers offer a wide range of other software packages including databases, spreadsheets, graphics packages and wordprocessors. Wordprocessing packages and graphics packages are used in the preparation of articles and reports and this results in a **familiarity with the operating environment**. As a result of this there is considerable inertia to be overcome before a new operating environment is considered. For example, the UNIX operating system (as is found on the Sun workstation) is generally considered to offer many useful facilities to the users of the system. However the commands used within UNIX are very different from those found under DOS and a significant learning curve would need to be overcome before it could be used successfully. A similar argument applies to the Vax VMS operating system.

The Apple Macintosh has a very attractive and simple to use graphical user interface, however the operating system is based entirely on Object Oriented Programming techniques and this determines the style of programs written on the Macintosh. Therefore any programs that make use of the operating system to display characters or read the mouse would need to be written in an Object Oriented style which again has a learning curve associated with it.

Finally, there is considerable **user support** available for the IBM PC since it is one of the standard hardware platforms adopted by the school. This user support covers both hardware and software and thus any problems arising in these areas could be addressed by the microcomputer support staff who are able to provide assistance as necessary. Also, since many members of staff and other students also used the IBM PC standard, other sources of support were also available.

Availability of machines
Availability of software
Widespread use of the hardware in other organisations
Familiarity with the operating environment
User support

Figure II.11 - Reasons for the choice of hardware platform

II.11 THE CHOICE OF DEVELOPMENT TOOL

Given the choice of development hardware three generic development tools were available for consideration (the fourth, expert system toolkits, were not available on the chosen hardware platform, and have only recently been introduced to top-of-the-range IBM personal computers). These are the use of artificial intelligence languages, the use of a conventional high level languages (with or without predefined modules) or the use of an existing, commercially available, expert system shell.

The option chosen was to develop an expert system shell using a high level language. The reasons for this approach and the choice of the high level language used (PASCAL) will now be described.

II.11.1 Existing work

One of the problems associated with using a conventional programming language is that the basic routines involved in implementing an expert system, such as symbol manipulation and list handling, are not found in the standard version of the language. This means that a considerable amount of work must be undertaken before the actual system can be implemented. LSE, however, ran a course on symbolic computing which included simple LISP programming and also the development of an expert system shell in the PASCAL programming language following the high level language with prewritten modules approach exemplified in ASPES (Doukidis and Paul 1987). PASCAL is the standard language for teaching programming in the school and was therefore the natural choice for creating the expert system shell. This choice was supported by the fact that LISP was not available on the Vax mini-computers on which most programming was undertaken at that time.

As a consequence of this work considerable experience was gained in using the PASCAL programming language, both in terms of general proficiency and also in terms of the techniques involved in developing an expert system. Other high level languages, such as C, were considered. However they were not readily available on the chosen hardware platform and, since they were not used to teach programming, they were not well supported by the school.

II.11.2 The choice of high level language

The main alternative to the PASCAL programming language for use in developing an expert system shell is the C programming language. It has been claimed that C has a number of advantages over PASCAL for writing computer programs (Crookes 1989).

C has provisions for writing programs in a modular manner, thus aiding debugging and the reuse of code. In PASCAL this support is not a standard feature and the support provided varies between compilers. C provides some data typing but this can be legally overcome if required. Data types in PASCAL are much stronger and it is not possible to (elegantly) overcome them. Most C compilers provide warnings about the use of variables (especially pointers) before they are assigned values.

The C language supports very efficient movement through arrays and also supports macros which can be used in conjunction with a precompiler to improve the runtime

efficiency of the program. It is also available on a wide range of hardware platforms and is the language used to write the UNIX operating system.

Many of the advantages that C offers over PASCAL were insignificant for the design of the expert system shell. The particular Pascal compiler used (see below) provides full support for the modular design of programs and also includes a smart linker that only links in those routines that are used in the final program, thus keeping the size of the compiled program to a minimum. The problems of data typing were not important in the shell (except for the storage of strings in the vocabulary tree, see the documentation provided in Appendix III).

The expert system shell makes heavy use of linked lists and does not use arrays to store any significant information. This means that the efficiency of C in dealing with arrays was not missed. The linked lists were kept simple and a modular approach was taken to ensure that every element in the list was properly initialised before use and therefore the warnings about using variables before they were initialised were not so important.

In summary, therefore, the main advantages of C over PASCAL were not used in the expert system shell and there was no incentive to explore a new programming language for developing the software. With hindsight, the advantage of the wide range of hardware platforms that support standard C might well have justified the use of C instead of PASCAL. The program design, however, does not make considerable use of special features of PASCAL and could be quite easily converted to this or any other high level language.

II.11.3 The PASCAL compiler

The first versions of the expert system shell were developed using Borland's Turbo Pascal Compiler Version 3. This only produced COMmand files which had a 64K size limit for programs and data. As the number of facilities offered by the system increased this size limit became a serious constraint and an attempt was made to use the Microsoft Pascal Compiler (Version 3.32). The documentation for this package was not particularly easy to read and lacked the ease of use found with the Borland package. More importantly Microsoft Pascal provided little support for the control of textual output on the screen. In contrast, the Borland compiler offered routines to define screen windows, to position text at any point on the screen and use whatever colours the programmer chose.

After two months working with the Microsoft Pascal compiler, Borland introduced their Pascal Compiler Version 4. This was a considerably more advanced compiler than Version 3 and offered a number of very useful facilities. These included a development environment using pull down menus, on-line help facilities and special `hot keys' to speed up the use of the menus.

Another very useful feature of the compiler is its use of **units**. These are effectively program modules that allow programmers to practice structured programming techniques and top down design. The program can be split up into a number of separate modules which can be used selectively. For example, one unit may simply contain the type definitions used within the program, another may contain the basic routines that deal with list manipulation, a third may hold the routines that deal with knowledge representation techniques and other modules may be assigned for different inferencing techniques. This approach means that it is possible to write general purpose routines, for example concerned with screen handling and windowing, which can be used in many different programs. Naturally the benefits of structured design in the location of errors and maintenance are also realised.

The use of units also speeds up the compilation process as each unit is compiled only once, with the compiled form used afterwards. It is only when the original pascal unit is altered (i.e. the compiled version is no longer an accurate representation of the source code) or if the programmer explicitly tells the compiler to re-compile all the units that a unit needs to be recompiled. Furthermore, the compiler has a `smart linker' that only links in those routines that are actually used by the program. This means that each unit can contain many house keeping and debugging routines which if they are not used in the final program will not be included in the final program. The use of units therefore generally improves the compilation time of the program in addition to keeping the actual program size to a minimum.

The Borland compiler provides a number of predefined Units which contain routines for manipulating text as it is displayed on the screen, routines that implement a number of high resolution graphics facilities and routines that provide direct access to the operating system.

II.11.4 Pascal versus AI languages

Many AI languages now offer similar development environments to those provided by high level language compilers such as Turbo Pascal. Therefore PASCAL must be shown to have advantages over artificial intelligence languages that do not relate to the development environment available.

The PASCAL programmer has far better control over how the basic facilities found in the expert system, such as knowledge representation, are implemented. This means that it is possible for a proficient programmer to implement a number of techniques which can improve the efficiency of the system both in terms of speed and size. One simple example will demonstrate this point. The clauses in an expert system rule are conventionally based on a series of words which are either natural language clauses or more formal representations. These clauses often need to be compared by the inference engine, for example, when searching for rules that contain a particular clause amongst their then-clauses. This comparison is normally performed on a character by character basis, with care being taken when considering the comparison between upper and lower case versions of the same letter. Further problems arise when dealing with the white spaces between "words".

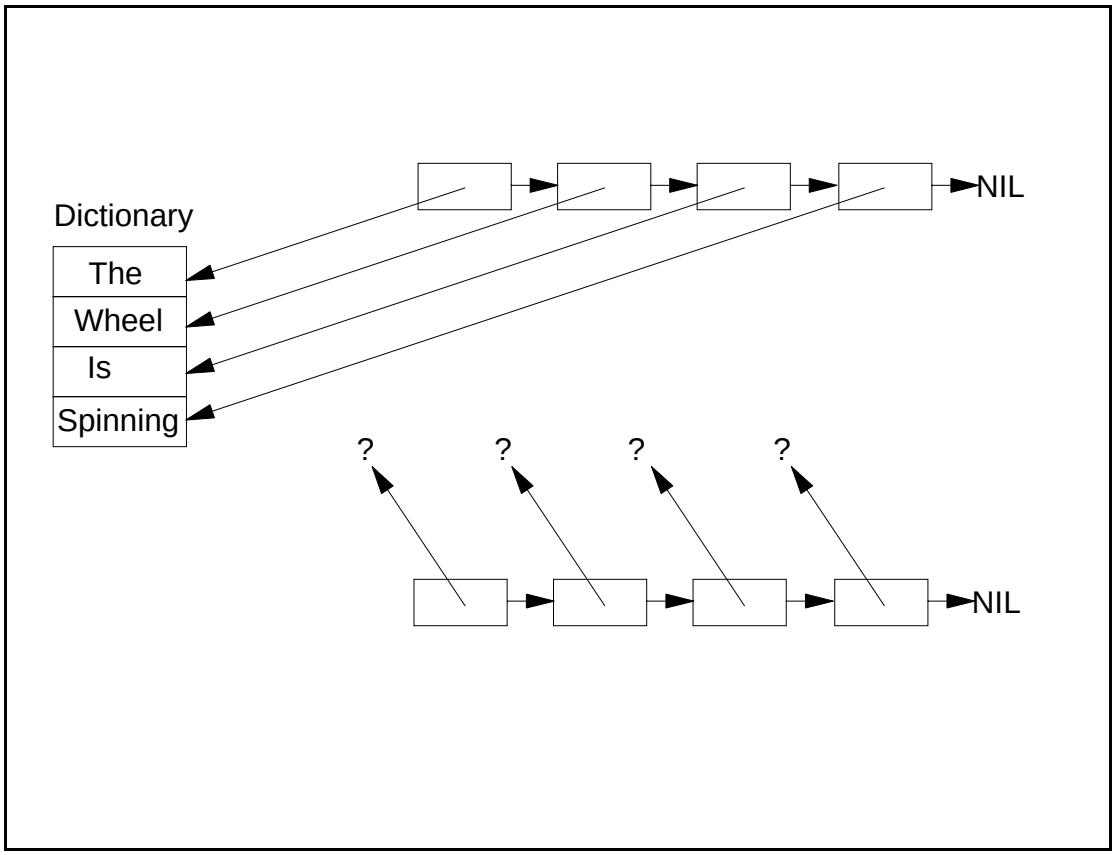


Figure II.12 - A more efficient comparison method implemented using pointers in PASCAL

A more efficient approach, see **Figure II.12**, is adopted in the PASCAL shell developed for the thesis. Every word used by the system is stored in a "dictionary" and before a new word is added it is converted to a standard form (in this case the first letter is capitalized, all the remaining ones are converted to lower case). The clauses in the rules then contain *pointers to* words in the dictionary rather than the words themselves. Comparison of the clauses then involves comparing two pointers - if they point to the same place then they represent the same word - which is normally a very simple operation by the microprocessor. Thus the comparisons are performed far more rapidly and the clauses also take up less memory as each word is only stored once in full. The operation of the system can be made even more efficient through the use of the dictionary and this is described in more detail in Appendix III.

II.11.5 Pascal versus commercial expert system shells

Commercially available expert system shells offer many facilities that would require considerable programming effort to implement in a high level language. The potential benefits of writing an expert system shell in PASCAL rather than making use of the facilities of an existing shell must therefore be discussed.

The advantages of `hand crafting' certain features in a high level language were discussed above in relation to artificial intelligence languages. This is also an important factor when considering expert system shells as many of these products are developed "on top of" artificial intelligence languages. Early versions of Xi+ were implemented on top of Micro-prolog (Forsyth 1987) and this considerably retarded the system's performance.

A second important factor relates to the ease with which additional functionality could be incorporated into the basic shell. With commercial shells this is not particularly easy, the additions would normally have to be separate programs that would be executed from the knowledge base, rather than being available for use whenever necessary. Leonardo attempts to minimise this problem by including a procedural language within the system, however even this is limited in functionality.

External programs that are added to the expert system shell will generally not be able to access the internal knowledge representation schemes used by the shell. Most commercially available expert systems shells do not provide `hooks' for programmers to access the knowledge representations found in the shells. Similarly various knowledge

manipulation routines are not easily accessed by external programs. It is not suggested that such links are impossible, but only that the task of implementing them will be rather difficult.

In comparison, the creation of an expert system shell using a high level language means that additional features can be added directly to the source code of the program, which can then be recompiled. Also, since these features are part of the source code, they will be able to directly make use of the knowledge representation techniques contained elsewhere in the source code for the program.

APPENDIX III - THE PESYS EXPERT SYSTEM SHELL USER GUIDE

III.1 THE PESYS EXPERT SYSTEM SHELL USER GUIDE

The following document is the user guide for the PESYS (Pascal Expert SYstem Shell) expert system shell. The program runs on an IBM Personal Computer and any compatible computer. The system makes use of standard memory and text graphics, however computers with larger memory (up to 640K) can run larger knowledge bases.

The documentation comes in three main parts. The first describes how to consult an existing application using the program PESYS. All the features of the system are described. The second part of the user guide describes the programs that are used in the development process and the final part provides technical details on how to write a knowledge base in the form of a simple tutorial on the development process.

Keys that are to be pressed are marked in **bold**, whilst text shown on the screen will be shown in *italic*. Finally any commands that you must enter are shown in underline.

III.2 GETTING STARTED

To start consulting the expert system, you must have a copy of the program together with the knowledge base you want to use. The program may be provided on a disk or may be found on a network. Particular details of how to obtain a copy of the program are provided separately. Suppose that program and knowledge base has been installed on drive C:. When faced with the DOS prompt *C:>* type the name of the program (PESYS), a space and the name of the knowledge base that you want to use, for example RICH, and then press **RETURN**.

```
C:>PESYS RICH
```

If you do not specify the name of the knowledge base, the system will display a prompt asking you to enter it at that time. The system will then display a title screen and wait for you to press any key.

Once the knowledge base has been specified the system will start to load the files associated with it. As this happens, a window is opened on the screen and the name of each file is displayed on the screen as it is read in. After the files have been read in the screen clears and the first selection menu in the system is displayed.

III.3 USING THE SELECTION MENUS

The user interface for PESYS is designed to be easy and consistent in use. In order to do this the system uses selection menus for all interactions where the possible options can be specified in advance. For example, when you have the choice between consulting the system again and finishing the interaction, the set of possible choices is known in advance and a selection menu will be available. In other instances, however, such as when the system asks for your name, it is not possible to limit the available choices and so a selection menu is not used.

III.3.1 THE TWO PARTS OF A SELECTION MENU

A selection menu consists of two main parts. Firstly there is the list of possible options, these may be listed vertically or horizontally, or a combination of the two. The second item is a highlight. The highlighted item is shown on screen in reversed colours. To move the highlight around the available choices press the cursor (arrow) keys. If you keep pressing the same key the highlight will loop round all the possible choices. To actually select an item from the list, move the highlight to the item required and press **RETURN**. In most of the selection menus, it is possible to press the **ESCAPE (Esc)** key and have a configuration menu. This is itself a set of selection menus and allows you to alter the configuration of the expert system inference engine. It is described in more detail below.

At any time when using a selection menu (and also whenever a window is open and a key must be pressed) pressing the function key **F4** will allow you to move the current window around the screen. Whenever the key is pressed a cursor appears on the screen. This cursor marks the top left corner of a window and can be moved around the screen to reposition the window by using the cursor keys. The cursor is only allowed to move to valid positions, i.e. the window will always remain fully displayed on the screen. Once the new position of the window has been selected, pressing **RETURN** will move the window to this new place. This feature is particularly useful if the default window covers some important information that is displayed on the screen.

III.4 THE CONFIGURATION MENU

The configuration menu has four items available for selection by the user of the expert system. These items allow the user to check and alter the settings of the inference engine, to save and load data that is stored in the working memory of the system and to view and change the basic parameters of the system.

Return to System Inference Engine File Operations Other Facilities

The first option on the list should be selected to continue with the use of the expert system. The second element allows you to change the way that the inference engine operates with the knowledge base. For example, you may want to use a different inference technique with the knowledge base. On selecting this option, a further list of options is displayed, allowing you to alter these values.

The third element on the menu allows you to load data from a file and store it in the working memory of the system or to save the data that has been arrived at during an interaction with the system.

Other facilities covers all the other options that can be set up for the system. This includes whether or not a log is kept of the interactions, the accuracy used for mathematical comparisons etc. A full description of all these items is given in the section on developing a knowledge base.

III.5 SELECTING ITEMS BEFORE THE INFERENCE PROCESS BEGINS

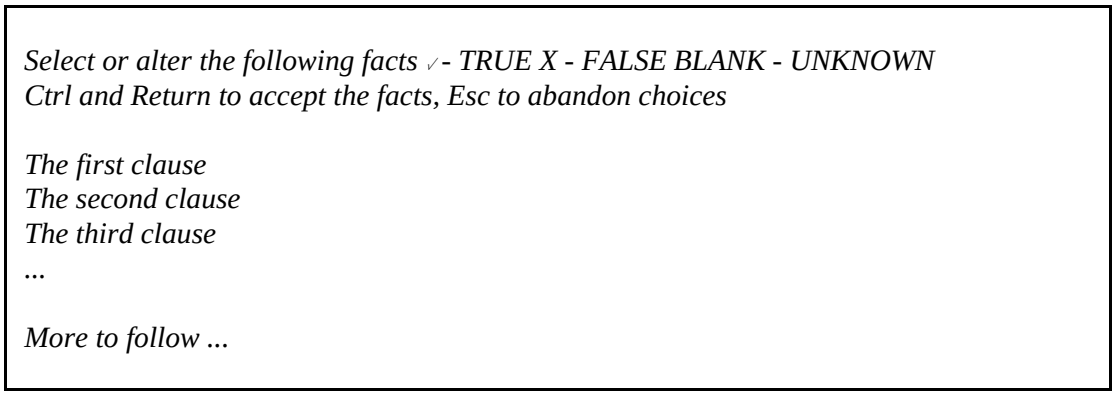
Ordinarily a forward chaining inference engine will produce no conclusions unless some information is provided for the system. In some applications this information may come from external sensors or from some external file that specifies the problem. However, in the sort of domains where PESYS would be best suited for, it is unlikely that either of these will be the case. In order to allow for applications that work with only forward chaining it is possible to configure the system so that it will provide a list of possible items that can be chosen before the inference process begins.

This list consists of all those clauses that cannot be arrived at using any of the rules. Effectively these are all the clauses that you could possibly be asked in order for the system to arrive at a conclusion.

For example,

IF there are no clouds THEN the sky is blue
IF the sky is blue THEN the weather is good

In this case there are three possible clauses that could possibly be used - there are no clouds, the sky is blue and the weather is good. However the last two of these can both be arrived at using the rules given, thus in this case you would be given a list containing only the first clause: "There are no clouds".



When the list is presented you have a number of options.

III.5.1 SELECTING THE ITEMS

If you want to select or deselect the available clauses you can move the highlight to the appropriate clause. Pressing **RETURN** once will select the item (this is shown by placing a tick ✓ in front of the clause) - a selected item is considered to be known to be true, pressing **RETURN** again will cause it to be deselected - known to be false (shown with a cross *X* in front of the clause). Pressing **RETURN** a third time will unselect the clause - suggesting that the truth or falsity of the clause is not known (this is shown by not having anything in front of the clause).

The highlight, that shows which clause is being considered can be moved up and down the screen using the cursor keys. If the display shows *More to follow ...* at the bottom of the page then this means that there are more clauses than can be shown on one page. As you scroll through the available options the remainder of the list will be displayed.

III.5.2 FINISHING THE SELECTION

Once the selection has been made you have two possible options, you can either confirm your choice and add the items to the working memory of the system as you have specified or you can abandon your choices.

III.5.3 TO CONFIRM YOUR CHOICE AND ADD THE ITEMS TO THE WORKING MEMORY

To add the items press both the **Control (CTRL)** and **RETURN** keys at the same time. There will be a slight pause as the system adds the clauses to the working memory and the system will then proceed to continue with the inference process.

III.5.4 TO ABANDON YOUR CHOICE

If you decide that you do not want to keep the selection that you have made (or if you do not want to select any of the items) then pressing the **ESCAPE (Esc)** key will cause the

system to abandon all the previous choices you have made and continue with the inference process.

III.6 ANSWERING QUESTIONS IN PESYS

In order for the system to arrive at a conclusion, it will, in most cases, be necessary for the system to ask you questions that relate to the domain.

IT IS IMPORTANT TO REMEMBER THAT THE SYSTEM DOES NOT UNDERSTAND THE QUESTIONS THAT IT ASKS, THESE QUESTIONS ARE BASED ON THE KNOWLEDGE BASE THAT YOU ARE USING AND HENCE ARE THE RESPONSIBILITY OF THE PERSON WHO WROTE THE KNOWLEDGE BASE. IF THE QUESTIONS SEEM IRRELEVANT OR IRRESPONSIBLE BLAME THE PERSON WHO CREATED THE KNOWLEDGE BASE NOT THE SYSTEM AND IF NECESSARY IGNORE THE QUESTIONS.

III.7 ANSWERING YES/NO QUESTIONS

The simplest form of question that the system can ask it the YES/NO question. In this case the system will open a window on the screen and ask

Is this true

This clause is true

Yes *No* *Unknown Explain Prompt*

Where "This clause is true" is the clause that the system is trying to arrive at, at that time. You have four or five possible options. Not all questions have prompts available and those that don't will not have the option prompt will not be shown, giving only four options. Using the selection menus, you move the highlight to the answer you wish to choose.

III.7.1 YES - THE CLAUSE IS TRUE

If you have decided that the clause displayed is true then select this option from the list. Once selected, the clause is added to the working memory as a true fact.

III.7.2 NO - THE CLAUSE IS FALSE

If the clause is known to be false then select this option from the list. Once selected, the clause is added to the working memory as a false fact.

III.7.3 UNKNOWN - NO INFORMATION IS KNOWN ABOUT THE CLAUSE

If you do not have enough information to decide whether the clause is true or not, then it will be necessary to select the unknown option from the list. If this is chosen, the clause is not added to the working memory and when the system tries to use the knowledge of the clause to fire a rule, it will not be able to and will therefore have to try and use a different rule or fail.

III.7.4 EXPLAIN - TO PROVIDE A JUSTIFICATION FOR THE QUESTION BEING ASKED

This option allows you full access to the explanation facilities provided by PESYS. These are described in full below. Once you have used the explanation facilities, you are returned to the main menu and you can make a decision about the clause displayed.

III.7.5 PROMPT - TO DISPLAY EXTRA INFORMATION WHERE AVAILABLE

In many cases the knowledge base will contain extra information that will assist you in deciding about the truth of a particular clause. This extra information may take the form of either a piece of text, for example it may contain the actual rules that the question is based on, or it may execute a command for more advanced facilities. Once this option has been selected, the appropriate information is shown on the screen and you must press any key. Pressing **F4** allows you to move the prompt window around the screen. This then returns you to the main menu, and based on this extra information you can make a more informed decision about the clause.

III.8 NON-LINEAR DOCUMENTS

One of the special features of PESYS is its use of non-linear documents. These are simply documents that allow you to examine a number of related concepts as well as the one presented by the designers of the system. When a non-linear document is displayed on the screen you will be shown a screen of text relating to a particular topic. Within this screen certain words will be highlighted. The highlight can be moved between these words by pressing **TAB**. To select a particular keyword press **RETURN**. On doing this the displayed screen will change to one related to the keyword that you have selected. Some screens take up more space than is available on the screen. In such cases, you will be presented with the option to move up or down the screen. Pressing **PageUp (PgUp)** or **PageDown (PgDn)** will then move you through these various screens. Pressing **Ctrl** and **F1** takes you back to the previous screen, whilst pressing **ESCAPE (Esc)** takes you out of the non-linear document. Finally pressing **F1** from within the document will take you to the main index / help screen.

III.8.1 HELP ABOUT THE SYSTEM

Non-linear documents are used to provide help about the PESYS system and can be obtained by pressing the **F1** function key at any time.

III.9 VALUE QUESTIONS

As well as asking you whether a fact is true or not, a particular knowledge base may also request values for certain variables. PESYS supports two forms of variables, text (strings) and numbers (reals). Whenever the system requests a value it will display a question that explains what is required and then waits for you to enter a value. For example, a knowledge base may need to know your age and might give a question like:

```
Please enter your age (in years)
```

You are then able to enter an appropriate value at the keyboard and pressing **RETURN**. Up to 255 characters may be used to enter values and the input line will scroll as necessary. In some cases, however, you may require a clarification of the question before entering a value. You may want an explanation or a prompt. These can be obtained by pressing the **ESCAPE (Esc)** key at any time whilst entering a value (before pressing **RETURN**). When this is done, the following selection menu is shown

```
Continue  Restart  Explain  Prompt
```

As was the case with YES/NO questions, those questions that do not have an associated prompt will not have the prompt option displayed. Selecting Continue will allow you to continue editing the same line as before, Restart abandons the line and lets you type in a new answer. Prompt and Explain operate as before.

III.10 RANGE QUESTIONS

In some cases, you may be asked to specify a range of values which are valid. For example, you may be estimating the wear on a device and may enter the minimum possible value, the maximum value and the most likely value. PESYS supports this form of input. The system begins by asking you for the minimum value for the variable, and then the maximum value.

```
Please enter the min value for The Wear On The Device?
0
Please enter the max value for The Wear On The Device?
10
```

The text of the question depends on the variable being input. After obtaining the minimum and maximum values for the range, the system displays a sliding scale for the possible range. You can move the slider (*X*) along the scale using the cursor keys until it marks the most likely value.

```
0 |----+----X----+----| 10
        ^           5
To find the most likely position, move the cursor "X" to the
appropriate part of the scale and press Return
```

III.11 FORMS

Forms are used when sets of related information need to be entered. Forms are made up of three main sections. Firstly, there is the background to the form. This is the text that appears on the form. This might be instructions, labels or a description of the information to be input. The second part of the form, which is optional, is made up of expressions that are evaluated and displayed on top of the background. Finally the form may have various slots that can have values entered into them. Again these slots are optional in a form. If a form does contain slots, however, the first of these will be displayed on the screen by the symbol "Δ". If you want to enter a value for this slot type in the value that you want. The process of doing this is the same as value and range questions described earlier. Once the value has been entered it is displayed on the form and the marker moves onto the next slot. If you want to skip a particular slot, you can simply press the **TAB** key and the marker will automatically move onto the next slot.

After the final slot has been reached, the system will then display "▉" at the bottom right corner of the form. This means that the form is "complete" and, if selected by pressing **RETURN**, will cause the system to continue with the inference process. Alternatively, if you press the **TAB** key again you will move to the first slot. This means that you can enter new values for any of the values for the slots time and time again until you are completely satisfied with them all. You can then press **RETURN** when the "▉" element is highlighted.

The line editor used within PESYS means that you can alter or replace the previous values of any slots. If you decide to press **RETURN** when the system is highlighting a slot that already has a value, a window appears with the value displayed. You are then able to edit it or replace it. If the first key you press represents a character which could be part of the value then the old value is replaced and you can continue to type the new value in. Alternatively, typing a cursor control key allows you to alter the current value. Access to the explanation facility and prompts is available as described above, when entering values for the slots.

III.12 THE EXPLANATION FACILITY

If an expert system is to be used in an organization, then the most important factor affecting its use (after the quality of the user interface - a program that cannot be used easily is unlikely to be used at all) is the explanation facility provided. The explanation facility of PESYS can be used in one of two ways.

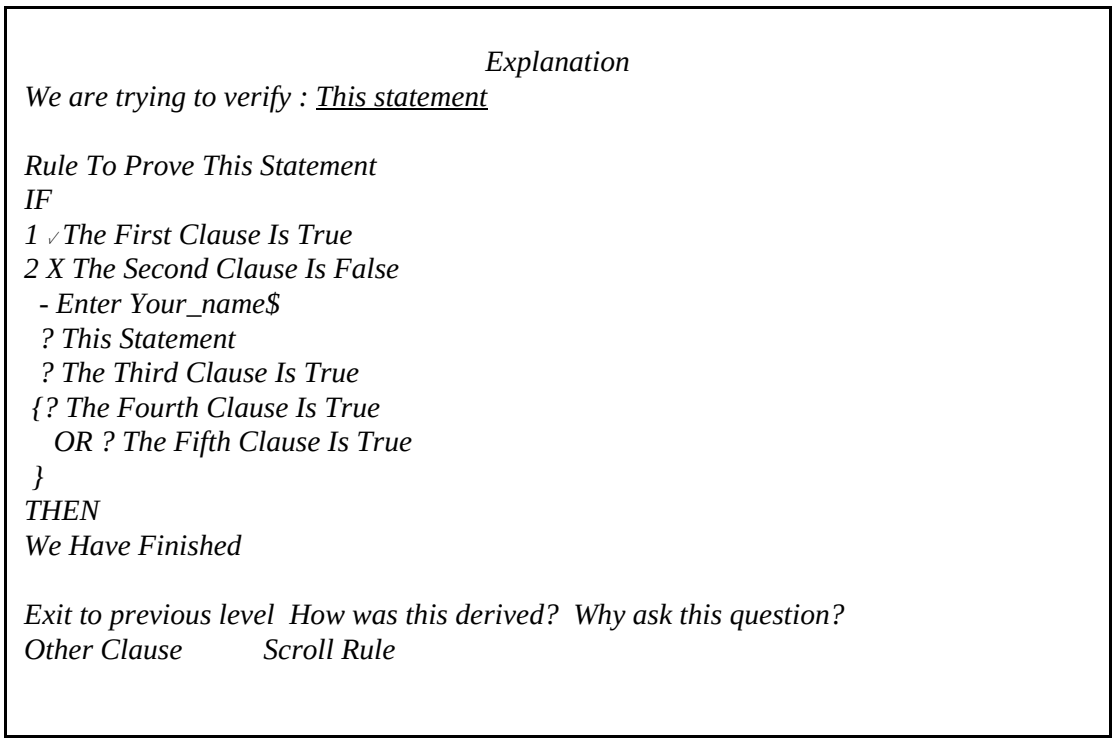
III.12.1 DURING AN INTERACTION

When the system asks you a question you may want to have some explanation of its line of reasoning, you may also want to check how it has arrived at certain statements.

III.12.2 ONCE THE INTERACTION IS COMPLETE

When the interaction has been completed and the system has arrived at a conclusion, it will be useful, in many cases, to examine the reasoning that the system has followed to arrive at that particular answer.

III.13 HOW TO USE THE EXPLANATION FACILITY



The explanation screen shows a lot of information. The first line tells you what clause is currently being used. This is followed by the rule that is being considered. The following indicators are used to show the truth of the clauses:

✓ This means that the clause is known to be true

X This means that the clause is known to be false

? This means that the clause is unknown

- This marks the clauses that are commands

{ and *}* mark the beginning and end of sets of related clauses that are either connected with OR or ATLEAST. Details of both these constructs are given in the section on writing applications.

After the rule has been displayed you are offered four possible choices, to exit from this level of explanation, to explain how a statement was arrived at, to ask why the clause needs to be arrived at or to examine other clauses. You are also able to scroll through the rule, which is particularly useful if the rule is larger than the available screen. Note that if you are using the explanation facility after the inference engine has finished working, the WHY option will not be available. Also pressing **ESCAPE (Esc)** will give you access to the

configuration menu. Selecting the first option will take you back through the explanations that you have made.

Selecting the How option causes the system to ask you which (numbered) clause you want to examine. If no clauses in the current rule have been arrived at, then obviously you cannot choose any of them and the system will inform you of this fact. Similarly if there is only one clause that has been arrived at in the current rule, the system will automatically select this one. If more than one clause has been arrived at in the current rule, you are asked to enter the number corresponding to the clause you wish to examine. Thus using the example above, if you wanted to know how *This statement is true* was arrived at, you would select 1.

III.13.1 OTHER CLAUSES

The other clauses facility is used to examine the reasoning of clauses that have not yet been arrived at. The available clauses are either goals or inform 2 clauses and you are asked which you wish to examine. On choosing the appropriate list you are given the opportunity to examine the available clauses and the rules that they are found in, plus an indication as to whether the rule has fired. Select the rule that you wish to examine and it will be displayed on the screen in the standard way. This allows you to examine an alternative reasoning path and see why a particular goal was not arrived at.

III.14 HOW THE CLAUSE YOU CHOSE WILL BE EXPLAINED

Once you have chosen the clause to be explained, there are a number of possible paths. If the clause was arrived at using a rule, then this rule will be displayed in a similar fashion to the first rule and the process will continue allowing you to explain the clauses in that rule etc.

If the clause you chose was a comparison, then the system will inform you that the comparison was made using known variables and the values of these variables will be shown on the screen as necessary.

A third possible way for the system to have arrived at a particular clause, would be if you told it that it was true directly. In this case the system would tell you.

III.14.1 THE WHY EXPLANATION

If you are using the explanation facility within an interaction, then provided that the system is using backward chaining, then it will be possible to ask Why are you asking me this question. Whilst the How facility examined those rules that had been used to arrive at a clause, the Why facility allows you to examine those clauses that could be used to arrive at a clause. The Why facility allows you to return all the way to the actual clause that is the overall goal for the system.

III.15 THE END OF THE INTERACTION

Once an interaction has been completed, one of two things will have occurred. Either the system has arrived at a goal, or no goals could be arrived at.

III.15.1 THE SYSTEM ARRIVED AT A GOAL

When the system has arrived at a goal, you will be shown something like:

These conclusions have been arrived at:

The system is ready

This conclusion has been arrived at using BACKWARD Chaining

Press <SPACE> to try and find another conclusion
Else press any key

In some circumstances, there may be a number of possible goals that would apply to the problem situation you are examining and pressing **SPACE** allows you to examine them all. This process continues using the runtime working memory that has already been arrived at, thus the system will only try and arrive at conclusions that are still consistent with the

current one. When you have finished using the inference process the system will tell you to press any key to continue. This is done to allow you to check any displays before the final steps are taken.

You are then asked to choose between:

1 goal has been arrived at by the system
What do you want to do?

1 - Examine the goals that were arrived at
2 - Examine the goals that could have been arrived at
3 - Move onto the what-if analysis or finish

Selecting the first option allows you to examine the reasoning of one of the goals that was fired to arrive at a goal. The second option, however, allows you to examine the reasoning behind any of the goal clauses, whether they were in a fired rule or not. The third option allows you to move onto the what-if analysis described below.

If you choose one of the first two options, then you will either be presented with a single goal (if there is only one) or a list of possible goals together with the rules that they are associated with:

Select a possible goal whose reasoning you would like to examine

Select one of the following

This is a goal - Fired Rule-name: This rule will have fired
This is another goal - Rule-name: Note that this rule will not have fired

By selecting a particular goal it is possible to examine the actual or possible reasoning behind that goal. Once the various reasoning paths have been examined, you can move onto the what-if analysis.

What IF Print Log Run Again Finish

III.16 OTHER OPTIONS

Selecting the What-if facility allows you the opportunity to alter values and clauses that have been arrived at previously and then to see what effect the changes have on the conclusions arrived at by the system. The What-if facility is described in more detail below.

III.16.1 PRINT LOG

The print log option allows you to print the log file automatically. Alternatively you can copy the file to the printer when the interaction is complete.

III.16.2 RUN AGAIN

The run again facility simply allows you to run the knowledge base again, without having to load it from disk.

III.16.3 FINISH

Finish clears out the memory of the program and returns you to the operating system.

III.17 THE WHAT-IF FACILITY

The What-if facility is fairly sophisticated in that it will allow you to alter any values that you have entered and see what the effects of this are. When you enter the what-if facility you have four options.

III.17.1 ALTERING THE EXISTING VALUES

The first option you have is to alter any existing values. In particular, those clauses that cannot be arrived at using other rules are displayed as well as any variables that you have used.

You have the facility to alter the truth of any of the clauses, whilst for variables you can type in new values.

III.17.2 ADD LOW LEVEL DATA

This option allows you to add the low level clauses in a similar manner to that described above.

III.17.3 SEE EFFECTS OF CHANGES

By selecting this option the system performs a forward chain with all the new data and any results are displayed. Note that the system does not perform a backward chain as this is likely to involve asking you questions again.

III.17.4 EXIT

Selecting this option leaves the What-if facility

III.18 THE DEVELOPMENT ENVIRONMENT

The development of an expert system application/knowledge base passes through a number of different programs. These are shown in **Figure III.1**. This part of the user guide describes these different programs.

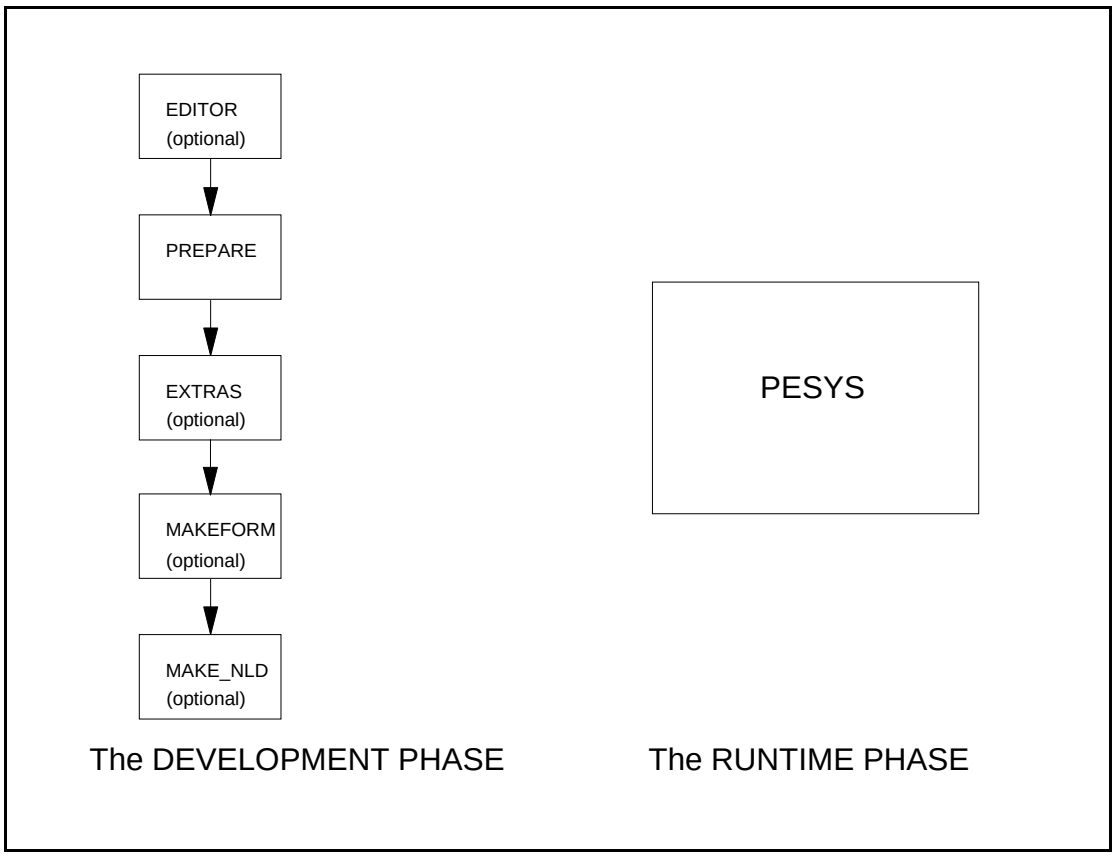


Figure III.1 - The stages of the development process

III.19 THE FILENAME STRUCTURE USED

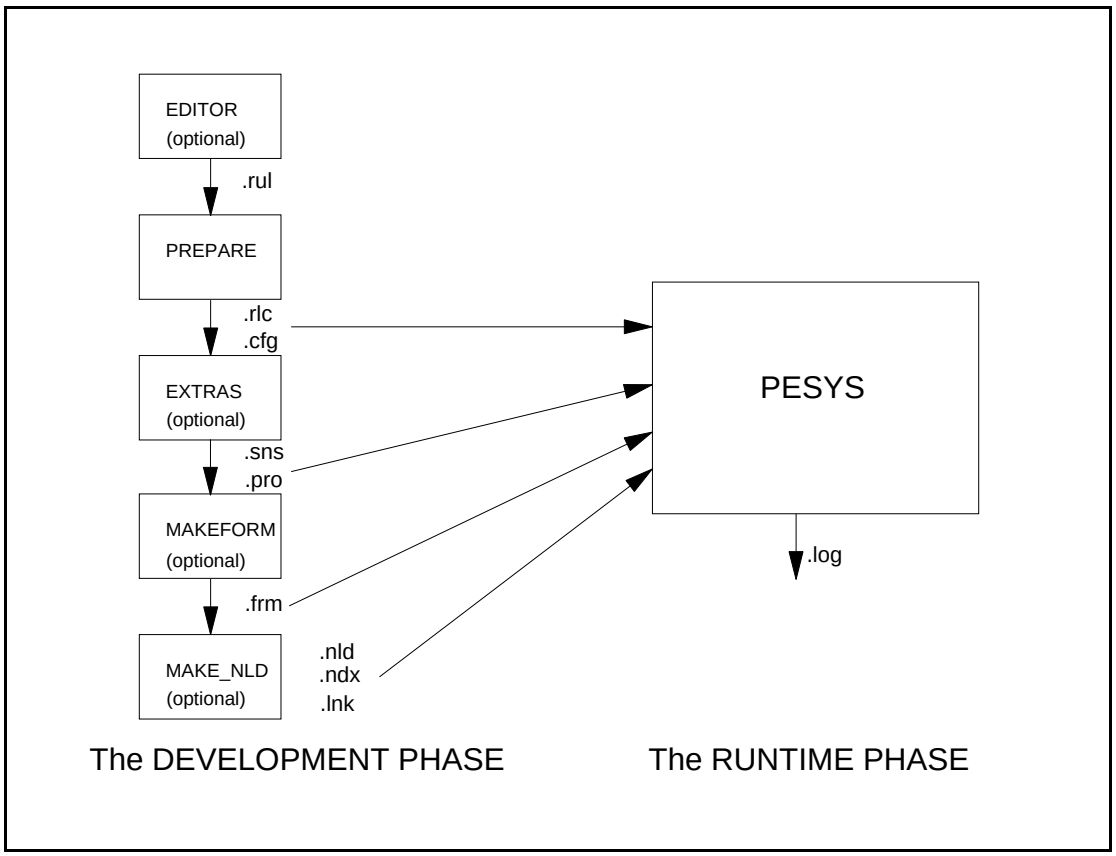


Figure III.2 - The files in PESYS

All the files in a knowledge base share the same main FILENAME. Each of the different files has its own extension to the main filename to identify the form that it takes. The different programs in the development process create the different files, as is shown in **Figure III.2**.

The extensions for the files are:

.RUL stands for the original rule file. This is the rule file that was entered by the knowledge engineer / expert when the knowledge base was created.

.RLC is the coded form of the original rule file. This file is created by the PREPARE program. All the development and runtime programs use the coded form of the rules rather than the original .RUL form.

.SNS contains the common sense information that is needed by the system. For the current version of PESYS this is primarily information about the variables used in the program - in particular the questions that are to be asked when variables are input, which files the values are to be found in, the valid ranges that may apply to numeric variables etc.

.PRO contains the text of any prompts that are presented to you when you require extra information.

.CFG is the file for the configuration information about the knowledge base. It contains information about the type of inferencing methods used in the knowledge base, what information is logged to disk and other system features.

.LOG is created whenever a knowledge base is consulted. It keeps a record of the information that has been derived by the system. According to the configuration the information is either brief or detailed.

.NLD contains the screens for the non-linear documents.

.LNK contains the information linking the various screens in the non-linear document.

.NDX is the index of screen names that is used to access the individual screens in a non-linear document by name rather than by number.

III.20 USING EDITOR

The PESYS development environment includes a full screen text editor (called EDITOR) which can be used to write knowledge base files (.RUL). Also, since all the files associated with a knowledge base are in plain ASCII format, it can also be used to edit any of the files, for example to correct any typing errors. If you want to edit a new file then simply run the program, whilst to edit an existing file give its *full* name when loading the program.

The editor supports a full range of editing features such as search, search and replace, go to the top of the file, go to the bottom of the file etc. It also allows you to define and alter preset keywords. This facility can be used if you are going to use a certain term many times in the knowledge base.

III.20.1 FACILITIES WHEN EDITING A PARTICULAR LINE

The facilities offered by the editor can be split into two main areas, those that refer to each individual line of the file and those that refer to the file as a whole.

START OF LINE / END OF LINE

To automatically jump to the start of the line, press **HOME**. Similarly to move to the end of the line press **END**.

CURSOR LEFT/CURSOR RIGHT

To move the cursor left and right use the cursor keys. Also **CTRL S** and **CTRL D** will move the cursor one character to the left and right respectively. Note that if you move beyond the end of the line or before the start of the line you will be moved down or up a line as necessary, if that is possible. **CTRL A** and **CTRL Cursor-left** will move the cursor one word to the left, whilst **CTRL F** and **CTRL Cursor-right** do the same to the right.

DELETING

To delete the last character entered, press **BACKSPACE** whilst to delete the next character in the line press **DEL**. Again wrapping occurs if you reach the end of the line. To delete a word press **CTRL T**, and to delete the current line press **CTRL Y**.

INSERT/OVERWRITE

To change between insert mode and overwrite mode, press **INS**.

PREDEFINED TEXT

The editor allows you to define up to 26 different sets of predefined text. These can be inserted anywhere in the file by pressing an associated key. The clauses are found by pressing **ALT** and one of the 26 alphabetic keys. Each key has its own phrase.

CHANGING THE PREDEFINED TEXT

To define your own predefined text, press **CTRL F10** followed by the key that you want to redefine. The previous value is shown, allowing you to alter it if necessary. Once the key has been redefined, all further uses of it will result in the new text being used.

III.20.2 FACILITIES WHEN EDITING WITHIN THE FILE

NEW LINE

Pressing **RETURN** creates a new line in the file.

MOVING BETWEEN LINES

The **Cursor-up** and **Cursor-down** keys move you between the different lines of the file. Similarly **PageUp (PgUp)** and **PageDown (PgDn)** moves you up or down a screen, if that is possible.

MOVING TO THE TOP/BOTTOM OF THE FILE

To move to the top of the file press **CTRL Q** followed by **R** and to move to the bottom of the file type **CTRL Q** followed by **C**.

FINDING TEXT

To find a piece of text press **CTRL Q** followed by **F**. You will then be asked to enter the text to be found and then press **RETURN**. The system will then try and find the required piece of text. If found, the cursor will be placed on it, otherwise you will be told that it cannot be found. To find the next occurrence of the text press **CTRL L**.

REPLACING TEXT

It is also possible to replace a piece of text. This is done by pressing **CTRL Q** followed by **A**. You are then asked for the text and its replacement. You are queried before the replacement is made. Otherwise the operation is as for FINDING TEXT.

SAVING A FILE AND EXITING

To save the file press the **ESCAPE (Esc)** key. You are then asked to enter the name of the file. If you are editing an existing file, the existing file's name is used as a default. Press **ESCAPE (Esc)** if you do not want to save the file. You are then asked if you have finished editing. Pressing **Y** will cause you to leave the editor, any other key returns you to the file.

III.21 USING PREPARE

Once you have written the rules, using the editor, it is necessary to convert them into a specially coded form for use by the main program. This process is undertaken automatically by the program PREPARE. However, before this is done it is necessary to configure the system for the particular knowledge base. This configuration can be altered during the execution of the shell PESYS and so it will be described in detail here.

III.21.1 CONFIGURATION

The configuration process encompasses two areas - the configuration of the inference engine and the setting up of other system values. To configure the inference engine you switch on or off various inferencing strategies.

FORWARD CHAIN

When this is selected the system will perform a forward chain when the knowledge base is loaded. Since forward chaining only works when known clauses match if-clauses of rules and a newly loaded knowledge base will not contain any known clauses, the only rules that will fire are those that have no if-clauses. By having such "empty" rules and selecting forward chaining it is possible to add default clauses to the working memory.

BACKWARD CHAIN

Selecting backward chain makes the system try and arrive at conclusions using the backward chaining mechanism. This method is described in more detail in the section on inferencing methods in the tutorial.

MIXED CHAIN

Sometimes it may be necessary to allow rules to fire whenever their if-clauses are true, regardless of whether they are currently being examined. This feature is often known as "demons" and is implemented by selecting mixed chaining. This means that whenever a clauses has been found using backward chaining the system performs a forward chain to see if this clause allows new rules (demons) to fire. Obviously backward chaining must be on before mixed chaining can have any effect.

PURE FORWARD CHAIN

This is an advanced form of forward chaining in that it will examine the if-clauses of rules and if they are commands it will try and execute them.

LOG INFORMS ONLY

The system will automatically create a trace of the actions undertaken by the system (such as what questions were asked, what answers were given, etc.). However by selecting LOG INFORMS ONLY, it is possible to restrict this trace to only inform statements. By deselecting this option all information will be logged.

SEND INFORMATION TO DISK

For some applications it may be desirable to prevent the logging of actions from taking place. In this case SEND INFORMATION TO DISK is deselected.

RETURN EXPRESSION RESULTS

Whenever an expression is evaluated, it is possible to display the results (for debugging purposes) by selecting this option.

SELECT FOR FORWARD CHAIN

By selecting this option, the user is presented with a list of all possible clauses that cannot be arrived at by using other rules before inferencing begins. It is normally used with forward chaining.

COMPARISON ACCURACY

Numeric comparisons are only available to a certain degree of accuracy. The value of the accuracy (default 1e-6) specifies how close numbers must be for comparisons to be valid.

DECIMAL PLACES

This option specifies the number of decimal places to be used when the system displays any numbers. This is the default value used if a variable does not have an alternative value of its own.

NORMAL COLOUR, REVERSE COLOUR

Alter the values for these options if different screen colours are required, for example for use with projection equipment etc.

III.21.2 CONVERTING THE RULES

Once the configuration has been made, the system automatically converts all the rules. A count is kept of the total number of rules converted.

III.22 USING EXTRAS

The program EXTRAS is a utility for getting the information about variables, prompts and forms for display in the runtime unit PESYS. Variables must have a specification for where they are to be obtained from and what prompts are to be displayed. The variables used in any forms must also have this facility. Also the text of any prompts must be found.

III.22.1 VARIABLES

The system performs this task by examining each rule. If there is a command to enter variables, use the range_of command or load a form, the system will find the variables used and will check to see if the variables already have information associated with them. If no information exists for the variables you will be asked:

Where from *From Keyboard* *From File* *From Range*

Select where you want the variable to be read from. Once this has been done, you must either type the prompt that is to be displayed (in the case of from keyboard or from range) or you must enter the filename where the variable can be found.

If the variable is numeric, you will then be given a number of extra options

Finish *Decimal Places* *Valid Range* *0 to infinity* *1 to infinity*

FINISH

Finish allows you to exit from this option

DECIMAL PLACES

This allows you to specify the number of decimal places to be used when the variable is displayed.

VALID RANGE

In some applications it may be necessary to specify a certain range for the valid values of a variable. For example, if asking for the month, you only want values from 1 to 12. Selecting this option allows you to enter the minimum and maximum valid values.

0 TO INFINITY

In some cases the valid range is from 0 upwards. In this case, rather than simply selecting the VALID RANGE and entering the values, use 0 to infinity to perform the task automatically.

1 TO INFINITY

This option performs a similar task to the one above, except that it starts from 1.

III.22.2 PROMPTS

If the system comes across any prompts that have not yet been defined it will ask you for details of the actual prompt.

III.23 USING MAKEFORM

The program MAKEFORM creates forms for the user interface of the runtime program. Each form is a screen. These screens can have any of three components - a background text, expressions to be displayed and variables to be entered. The position of the variables / expressions and the number of characters to be displayed are specified using MAKEFORM.

III.23.1 CREATING A FORM

Specifying the name of the form when loading the system allows you to alter an existing form. The extension .FRM is used to signify that the file is a form. Note that when you save the form you are always asked for a filename and therefore you will not overwrite the file that you read in unless you save it under the same name. This means that, for example, you can create a standard background form (Standard.frm) and use it to create a number of different forms.

MAKEFORM standard.frm
(make alterations to form)
Save Screen1.frm

III.23.2 THE OPTIONS AVAILABLE

Once you have entered the program, you are shown the following menu:

Add text *Variables* *Expression* *Edit*
Delete *Redraw* *Exit*

EDIT BACKGROUND

Selecting this option places you in the main screen. Your position is indicated by the cursor. Any text you type will be placed on the screen. Using the cursor keys you can move around the screen to position text. To erase text either overwrite it with spaces or use the **BACKSPACE** key.

For the system to work will all displays, there is no facility for altering the colour of the text. To finish adding text press **ESCAPE (Esc)**. This returns you to the main menu. You can add more text simply by selecting this option from the list.

ADD VARIABLES

This option allows you to add variables to the display which the user may enter. Once the option has been selected move the cursor to the position where the variable will be displayed once it has been entered (it is advisable to leave one space after any text introduction as the system displays a prompt for the user) and press **RETURN**. You will now be asked to enter the name of the variable to be entered and the number of characters of the users response that are to be displayed. Note that this is not the number of characters used to store the response, only the number of characters shown. Once you have specified this the screen will show the area and you can add more variables. Once again, pressing **ESCAPE (Esc)** takes you back to the main menu. Selecting this option again allows you to add more variables.

ADD EXPRESSIONS

This option is identical to the one above, except that you can specify expressions to be displayed rather than variables to be entered.

EDIT

In order to edit something you must first specify if you want to edit variables or expressions (alternatively you can abandon this option). Once you have selected one of the two lists a marker is placed by the first item in the list chosen. (If there are no items in the list you are returned to the main menu). This marker is moved through the list by pressing **TAB**. Once you have found the item you want to edit, press **RETURN**. **ESCAPE (Esc)** leaves this option.

The details of the variable/expression chosen are then shown:

Expression
X Value
Y Value
Display Length
Finish

Selecting one of the values allows you to enter a new value. Selecting finish lets you edit another item in the list. It is important to note that any changes you make will not be shown immediately, you must select the redraw option to see any changes.

DELETE

To delete an item you must first select the list from which it is to be deleted. Then, as above, you select the item to be deleted. Pressing **RETURN** then deletes the item. Again no changes to the display will become apparent until you select the redraw option. Since the delete option cannot be undone, once an item has been deleted you must select the delete option again to delete another item.

REDRAW

The redraw option redisplays all the background text and variables/expressions to be displayed, showing any changes made by the edit and delete options.

FINISH

When you have finished adding text and variables and expressions to the display, selecting Finish will allow you to save the form. The system asks you for a filename for the data. Use the extension .FRM to specify that the file is a form. The form is then saved in the named file. A data compression routine is used to minimise storage requirements. If you do not specify a file name the form and all the alterations are abandoned.

III.24 USING MAKE_NLD

The program MAKE_NLD creates a set of files for a non-linear document. It is only necessary to specify the text of the document and the links between different screens. All the house keeping tasks are performed automatically by the system.

Non-linear documents are made up of two main components. The first of these are the screens in the document. These are simply pieces of text that are displayed on the screen. Each screen is given a unique identifying name so that it can be referred to by other screens. The other component of a non-linear document is the link that exists between different screens. Each link

is associated with a piece of text which is displayed on the screen and refers to a separate screen. It is through these links that users are able to move through a non-linear document.

To create a non-linear document you must first specify its name. It is advisable to use a name that is related to the knowledge base being designed. If the name is new you are presented with a completely blank document, however if the non-linear document already exists then you can edit it. The first element in a non-linear document is used as a help facility or central index and so it is advisable that the first screen entered is used for this purpose. When you are editing a non-linear document the following options are available to you:

CREATE A NEW SCREEN

This process allows you to create a new screen and also to link it with other screens. It is effectively a combination of some of the other options.

SELECT A SCREEN

The remaining options operate on screens and links that have already been created. Choosing this option, therefore, allows you to select an existing screen to modify. The system presents you with a list of all the available screens and you can choose the one you want to alter. Once the screen has been selected it is displayed in the main editing area.

EDIT SCREEN

This option is used to alter the text in an existing screen. A full on screen editor is available for this purpose. To exit the editor, press **ESCAPE (Esc)**.

EDIT LINK

This option allows you to edit an existing link, either by changing its text, its position or the screen it points to.

ADD LINK

Instead of editing an existing link, this option allows you to add a new link to a particular screen.

DELETE SCREEN

This option allows you to delete individual screens and their links to other screens.

QUIT

This option allows you to leave the application. When this is done, all the necessary updating of files is performed automatically.

III.25 WRITING A KNOWLEDGE BASE IN PESYS

PESYS is a rule based expert system shell. This means that the representation of knowledge in the system is in the form of if-then "production rules". Production rules are the most common knowledge representation technique used in expert system shells. Some shells add the facility to describe objects using techniques such as first order logic or frames. In many cases, however, this is done by "forcing" the knowledge into unnatural formalisations which can be both inappropriate and inefficient.

The rules in PESYS are designed to be as easy to understand as possible - this is done by allowing them to be written in natural language (English) with only one or two minor restrictions. A typical (simple rule) looks like this:

This rule considers the case of normal applications
if
the student is an ordinary applicant
then
we have found the status of the applicant

III.25.1 PARTS OF THE RULE

RULE NAME

The first line of the rule is the rule name. This can be anything (up to 255 characters). One possibility is simply to number the rules, however this is not particularly useful. A far more useful form is to use a descriptive rule name. A rule name that fully describes the actions of and reasons for a rule will not only aid in debugging the knowledge base, it will also be particularly useful in the explanation facility since it can convey the intentionality behind the rule.

IF

After the rule name is a line containing the keyword IF. Note that PESYS is completely case independent - the word IF can be in upper or lower case, or a combination of the two. This case independence occurs throughout the system.

IF-CLAUSES

After the IF line come the if-clauses. The if-clauses are a list of statements (zero or more) that define the conditions necessary before the rule can be used. If-clauses can either be clauses of text or they can be commands. The text clauses can be of any form. For example:

The ball is red
The animal has long hair
The person has attended the course for at least sixty days in the last year
We have got the basic information about the user
etc.

Negations are easily defined by adding the word NOT to the sentence. Thus the above clauses could have negated forms of:

The ball is not red
The animal has not long hair
The person has not attended the course for at least sixty days in the last year
We have not got the basic information about the user

When the knowledge base is PREPAREd all the NOTs in the clauses are removed and counted. The number determines whether the clause is negated or not. When you are asked about the clause you are always asked to confirm a positive statement.

i.e. Is this true
You Have Done All The Work
rather than

Is this true
You Have Not Done All The Work

which has possible responses of
Yes - I have not done all the work or
No - I have done all the work?

This method can deal with double (or more negations). For example - "is it true that it is not the case that you did not go to the concert" means the same as "Is it true that it is the case that you did go to the concert". The only limitation is that sometimes the addition of the word will not be natural. In these cases it is best to simply add the word NOT to the start of the clause.

THEN

The conditions of the rule (the if-clauses) are separated from the actions part of the rule by a line containing the keyword THEN.

THEN-CLAUSES

The last part of the rule are the then-clauses. These have the same form as the if-clauses. There can be zero or more then-clauses.

III.25.2 FIRING A RULE

A rule is said to "fire" when all its if-clauses are "true". In the case of commands, these are "true" when they are executed. Comparisons between variables (both string and numeric) are "true" only when the comparison is true (to the degree of accuracy specified). Finally text clauses are "true" only when the working memory contains clauses whose known truth or falsity matches that of the clause. When all the if-clauses are "true" (logically they are combined with the AND connective) the rule "fires". The text clauses in the then part of the rule are then added to the working memory and any commands are executed.

III.26 HOW THE SYSTEM FIRES RULES

There are two basic ways in which the system can fire rules: It can take whatever data is available and see what "falls out", or alternatively it can try a particular path and see whether the data supports it. Technically these two strategies are known as Forward chaining and Backward chaining. The easiest way to understand these methods is to see how the inference engine undertakes each strategy.

III.26.1 FORWARD CHAINING

The inference engine starts at the top of the rules. Consider the rules as being in a long line attached with string. It then looks at each rule to see whether it has been fired already. If the rule has not been fired the inference engine looks at all the if-clauses. If they are all true the rule fires and the then-clauses are added to the working memory and the system continues by examining the next rule. Once all the rules have been examined the system starts at the top of the list of rules again. This process continues until no more rules can be fired or a goal has been arrived at.

III.26.2 INFORM LEVELS AND GOALS

Every time a rule fires, it adds then-clauses to the working memory. Each then-clause is potentially useful information for the user. In order to differentiate between clauses that the user needs to know when they have been arrived at, and temporary clauses that are simply used

in the inference process, PESYS allows you to mark those clauses that the user needs to know about with the keyword INFORM. Thus if the user needs to know that the animal is mammal, the clause will have the keyword inform added to it. For example,

Inform 2 The animal is a mammal

Note that the inform keyword has a "level" associated with it - in this case level 2. Any clauses which do not have the keyword inform are considered to be of level 0 and the user is not informed when the clause is added to the working memory. Inform levels of 2 (and above) are considered to be useful information that the user should be informed about, however they are not the final result and once the user has seen the clause (and pressed a key) the system continues to arrive at conclusions.

Inform clauses with a level of 1 are considered to be goals and whenever they are arrived at the system stops inferencing. The main advantage of this approach is that it minimises the possibility of data redundancy and the problems associated with this For example, the normal way of specifying the goals that the system is trying to arrive at is to have a list of goals at, say, the start of the knowledge base. However when you add or remove rules from the main knowledge base there is no guarantee that you will remember to add or remove goals from the list of goals. If, for example, a goal is not added when new rules are added the system will never consider the rules for the (missing) goal and the system will be incomplete. By attaching the "goal status" of a clause to the rule itself, this problem of redundancy will never arise. This direct attachment also means that whenever a rule with inform 1 then-clauses is fired the system will recognise that it has reached the end of the inference process and stop.

III.26.3 BACKWARD CHAINING

Backward chaining works in the opposite direction to forward chaining. Instead of taking the data and seeing what comes out, backward chaining tries to fire a rule directly.

The first step is to list all the rules that have inform 1 then-clauses. The system will try and fire each of these. To do this, it takes the first rule in this list. In order for this rule to fire, it will need to check that all the if-clauses are in the working memory of the system. Assume that there is no information in the working memory. This means that none of the if-clauses are

known. The system must therefore try and arrive at the if-clauses indirectly. The if-clause becomes the new goal that the system is trying to verify.

Firstly, if the clause is a command the system executes it immediately and moves onto the next if-clause. Similarly, if the clause is a comparison the system tries to evaluate it. If the comparison is valid, the system moves onto the next if-clause.

Finally, if the clause is textual, the system checks to see whether it is in the working memory (or if its negation is in the working memory if the clause is negated). If it is not, the system must try and arrive at the answer. The first way to do this is to arrive at it, directly or indirectly, from other rules and so the system makes a list of all the rules that contain the clause in their then-clauses (i.e. rules that, if fired, would add the clause to the working memory). If this list of rules is empty the clause cannot be arrived at by using rules and so the system must ask the user about the clause.

If there are any rules that could arrive at the clause, the system sees if it can fire any using the data that already exists in the working memory. If the rule fires (and the clause is arrived at) the system moves onto the next if-clause. If none of the rules can be arrived at, the system tries each rule indirectly, that is, each if-clause in this rule is checked using the same method as described above. This process continues until either a goal is arrived at, or there are no more rules to be tried.

III.27 A SIMPLE EXAMPLE

To demonstrate the two techniques we will consider a very simple example. The knowledge base will try to decide which of three drinks you have before you. The three drinks are milk, lemonade and coffee.

In order to identify each of these, a rule is needed for each and since our goal is to identify the drink in question, the then-clauses in each rule will be marked with inform level 1. First consider milk

```
This rule fires if the drink is milk - Note the meaningful rule name
if
the glass and liquid are cool to the touch
and the liquid is a white colour
then
inform 1 the drink that you have before you is milk
```

Next consider lemonade

this rule fires if the drink is lemonade
if
the glass and liquid are cool to the touch
and the liquid is clear
and the liquid is carbonated
then
inform 1 the drink you have before you is lemonade

finally for coffee

this rule is for coffee
if
the glass and liquid are hot
and the liquid is a dark colour
then
inform 1 you have a glass of coffee

Since the user may not understand the term carbonated, we can add the following rules:

carbonated drinks taste fizzy
if
the drink has a fizzy taste
then
the liquid is carbonated

carbonated liquids have bubbles
if
the drink has bubbles in it
then
the liquid is carbonated

III.27.1 A FORWARD CHAINING EXAMPLE

If we PREPARE this file with the following configuration (see the section on using PREPARE).

Forward Chain True

Backward Chain *False*

Mixed Chain *False*

Pure Forward *False*

...

Select Items *True*

When you run the program you will first be shown a screen

The drink has a "fizzy" taste

The drink has bubbles in it

The glass an liquid are cool to the touch

The glass and liquid are hot

The liquid is a dark colour

The liquid is a white colour

The liquid is clear

Select the following items

The drink has a "fizzy" taste

✓*The drink has bubbles in it*

✓*The glass an liquid are cool to the touch*

The glass and liquid are hot

The liquid is a dark colour

The liquid is a white colour

✓*The liquid is clear*

Press **CTRL** and **RETURN** to start the inference process.

First cycle. The forward chain mechanism examines the first rule - the first clause matches, the second doesn't. Second rule, the first and second clauses match, the third doesn't. Third rule, the first clause doesn't match. Fourth rule, the first clause doesn't match. Fifth rule, the first (and only) clause matches, the rules fires. "The liquid is carbonated" is added to the working memory. End of first cycle.

Second cycle. First rule, first clause matches, second clause doesn't. Second rule, all three clauses match and the rule fires. Then-clauses contain inform 1 clause and so inference process stops. End of second cycle.

System completes forward chaining by informing you that the drink you have before you is lemonade.

III.27.2 A BACKWARD CHAINING EXAMPLE

To see how the system works using backward chaining, PREPARE the rules with the following configuration.

Forward Chain False
Backward Chain *True*
Mixed Chain *False*
Pure Forward *False*
...
Select items *False*

Firstly the system makes a list of all the rules which have inform 1 then-clauses.

This rule fires if the drink is milk
This rule fires if the drink is lemonade
This rule is for coffee

It considers the first rule in this list. In order to fire this it needs to know if the if-clauses are true. First if-clause: "The glass and liquid are cool to the touch". This is not in the working memory and so it must be verified. The system first finds any rules that contain the clause in their then-clauses - there aren't any, so the user must be asked directly. Suppose the answer is yes, the system then considers the second if-clause "The liquid is a white colour". Since this is not in the working memory, it must again be verified. Once again there are no rules to arrive at this so the user is asked directly. This time the answer is No. This means that the clause is not true and so the rule cannot fire.

The system then considers the second Inform 1 rule. The first if-clause ("The glass and liquid are cool to the touch") is already in the working memory and so the system can move onto the next if-clause. Since this is not known and cannot be arrived at using any rules, the user is again asked and answers Yes. The system then moves onto the third clause - "The liquid is carbonated". This is not in working memory and must therefore be verified. The clause can be arrived at by firing two rules :- "Carbonated liquids taste fizzy" and "carbonated liquids have bubbles". The system will first see whether these rules can be fired using data in the working memory - they can't. It then tries to fire them indirectly.

First rule: In order to fire the system needs to consider the clause "The drink has a fizzy taste". This is not in the working memory and so it must be verified. There are no rules to arrive

at it so the user is asked. The answer is No, and this means that the rule cannot fire. The system then moves onto the next rule. Again the if-clause must be verified by asking the user directly. The answer this time is Yes and the rule fires, adding the clause "The liquid is carbonated" to the working memory. The addition of this clause means that the rule "this rule fires if the drink is lemonade" also fires, adding an inform 1 clause to the working memory. This means that the system halts and informs the user that a goal has been arrived at.

III.28 ADVANCED FEATURES IN THE KNOWLEDGE BASE

A second knowledge base, Rich.rul, illustrates a number of more advanced features.

III.28.1 META RULES

A meta rule can be described as one which describes how other rules are to be used. This is a useful facility when you do not want rules to fire at random, rather you want to control the actions of the system. Effectively this means that meta rules can be used to "program" the actions of the knowledge base.

For example, in the Rich knowledge base we are trying to determine what the likelihood is of the user becoming rich. In order to do this we want the person to provide some personal information before we can determine the possibility of becoming rich. The if-clauses for this rule are therefore a statement checking that we have obtained the basic information and a statement asking if we have considered the ways of becoming rich. Once these steps have been taken, we can finish and inform (1) the user that the system has done its work

```
this rule controls the main actions
if
we have got the basic information about the user
and we have considered ways of getting rich
then
inform 1 the system has assessed your richness potential
```

III.28.2 TOP DOWN DESIGN

One of the most effective software design methods for minimising maintenance problems is top down design. This principle can easily be applied to the writing of knowledge bases. If the previous rule is considered as the top of the design, then by a top down breakdown, the next rule to be considered will be one to get the basic information from the user.

III.28.3 USING FORMS TO ENTER DATA

One frequent difficulty with expert systems is the fact that once you have entered data, because the system is often using a backward chaining technique, there is no way to go back and alter values. PESYS partially overcomes this problem by allowing the designer of the knowledge base to group a series of questions together in one form. The assumption being that if you want to change a value, you will do so with recent answers, rather than earlier answers. A form for entry allows you to alter the values until you select the end item.

III.28.4 COMMENTS

Comments may be inserted into the knowledge base by adding lines beginning with an asterisk (*).

III.28.5 OR CLAUSES

The ability to combine clauses logically using OR as well as AND is also available in PESYS. The if-clauses that we have seen so far can be considered as sets of OR clauses where there is only one alternative - this OR nothing.

A simple rule with ORs and ANDs
If

A
Or B
Or C
And D

is represented by the system as

A simple rule with ORs and ANDs
If
A or B or C
D
then

Only one item from each line needs to be true for the rule to fire. See how this corresponds to the simple if-clauses we saw earlier.

A simple rule with ANDs only
If
A
B
C
then

which is represented as

A simple rule with ANDs only
if
A
B
C
then

Where again each line needs only one true clause for the rule to fire.

Every if-clause is actually a list of or-clauses.
If
{
this is true - an or clause
or that is true - an or clause
}

Where the contents of the curly brackets is one if-clause. If the first or-clause is true the whole if-clause is true. If it isn't the second clause is considered. This process continues until one of the or-clauses is true, or there are no more or-clauses to be considered.

III.28.6 VARIABLES

PESYS allows you to use variables as part of the knowledge base. Both string and numeric variables are supported with expression evaluation available for numeric variables. Variable names can be any alphanumeric symbols and underline ("_"), provided that the first character is alphabetic. String variables are marked by adding $ to the end of the variable name. Meaningful variable names considerably aid the readability of knowledge bases.

First_name$ - a string variable age - a numeric variable

All numeric variables are reals.

III.28.7 COMPARISON OF VARIABLES

Numeric variables can be compared with one another or expressions. If the comparison (to the defined level of accuracy) is valid, the clause returns true.

age > 21 returns true if the current value of age is greater than 21.
min_of_these <= a+(6*b) returns true if the comparison holds.

Comparison of string variables is also possible:

first_name$ = Edgar

Note that no quotation marks are required around string constants.

III.28.8 ATLEAST CLAUSES

A final control structure is the ATLEAST construct. The ATLEAST command is an advanced form of the OR clauses described above. When the system was evaluating a list of or-clauses, it moved onto the next if-clause as soon as one was found to be true. However in the ATLEAST command, every clause is examined and a count is taken of the number of true ones. If this count is more than or equal to the limit in the command the if-clause is set to true. In addition a variable, how_many, is set to the number of true clauses.

III.28.9 PROMPTS

The user may be given useful information, in the form of prompts by adding the word PROMPT to the end of the clause and then specifying the prompt name on the next line.

For example:
You keep your eyes open all the time prompt
Never_know

When you run the program EXTRAS, you will be asked for the text of the prompt associated with the question.

III.28.10 COMMANDS IN PESYS

The following commands are available in PESYS.

LET This command is used to assign values to numeric and string variables. For numeric variables a simple expression evaluator is available that allows the use of variables within expressions.

DOS, **DOS_E** The DOS and DOS_E commands are used to execute external programs from PESYS. DOS_E should be used if the parameters to be passed to the programs are to be evaluated using variables known to the system, otherwise DOS should be used.

PRINT This command is used to output information onto the screen. Anything within quotation marks is taken to be a constant and is displayed immediately, anything else is evaluated and the result is displayed. Note that there are no separators between the different expressions to be evaluated.

ENTER **RE_ENTER** The enter and re_enter commands are used to input values from the keyboard. The commands differ in that enter only obtains a value if none exists previously. Re_enter, on the other hand, always obtains a value. The values entered can either

be obtained from the keyboard, from a file or by use of a slider marking the minimum, maximum and most likely values of the variable.

OPEN_IN **OPEN_OUT** PESYS is able to communicate with other programs through the use of ASCII data files. The commands open_in and open_out are used to open files for inputting data and for outputting data respectively and take the name of the file to be opened as a parameter.

APPEND The append command is used to open an existing file so that values are written at the end of it.

READLN **READ** The readln and read commands are used to obtain values from a file that has been previously opened for input. The commands differ in whether the items are expected to be one per line or not.

WRITELN **WRITE** Writeln and write are used to output data to the opened output file. writeln adds an end of line character to each line of output.

CLOSE_IN **CLOSE_OUT** Close_in and close_out perform the opposite actions to open_in and open_out.

CLS The cls command is used to clear the display screen.

PRESS_KEY When executing this command, the system waits for the users to press a key before continuing.

RANGE_OF This can be used instead of enter or re_enter to force the users to enter a minimum, maximum and most likely values of a variable.

USE The use command is used when a sub-knowledge base is to be used in the inference process. The specified knowledge base is loaded and executed and once it is complete, control returns to the previous knowledge base. The available memory of the system is the only restriction on the depth of nesting of knowledge bases.

FORM The form command is used to display a particular form on the screen.

NLD This command is used to access the non-linear documents. Two parameters are required namely the name of the file being used and either the location of the required screen or its name.

APPENDIX IV - THE MAIN DATA STRUCTURES USED BY PESYS

The PESYS system makes use of three basic data structures. These are binary trees, linked lists and a special form of binary tree that is used to store facts about the domain - the working memory of the system.

IV.1 BINARY TREES

Simple binary trees are used to sort data in the expert system. For example, when presenting the users with a list of known facts to be altered in the what-if facility, they are added to a binary tree and so are presented in alphabetical order.

A binary tree is also used to store all the words known in the system, effectively the structure acts as a dictionary for the system. Words are represented in the rest of the expert system as pointers to the actual word in the tree. Comparison of words, therefore, involves comparing pointers to nodes in the tree. If two pointers point to the same node then, by definition, they point to the same word. Comparison of words is then efficiently implemented as a comparison of pointers rather than a lengthy character by character comparison.

The only problem with this tree arises when storing the words in the system using Turbo Pascal strings. Each word in the tree is converted into a standard form (with the first character in capitals and the remaining letters in lower case) and could be between 1 and 255 characters in length. Allocating the maximum length (256 bytes) for every word is very wasteful of memory and so a rather clumsy mechanism is introduced which makes use of a number of different string types and pointers to them. This problem is a direct consequence of the way that PASCAL handles strings and would be better implemented in C.

IV.2 LINKED LISTS

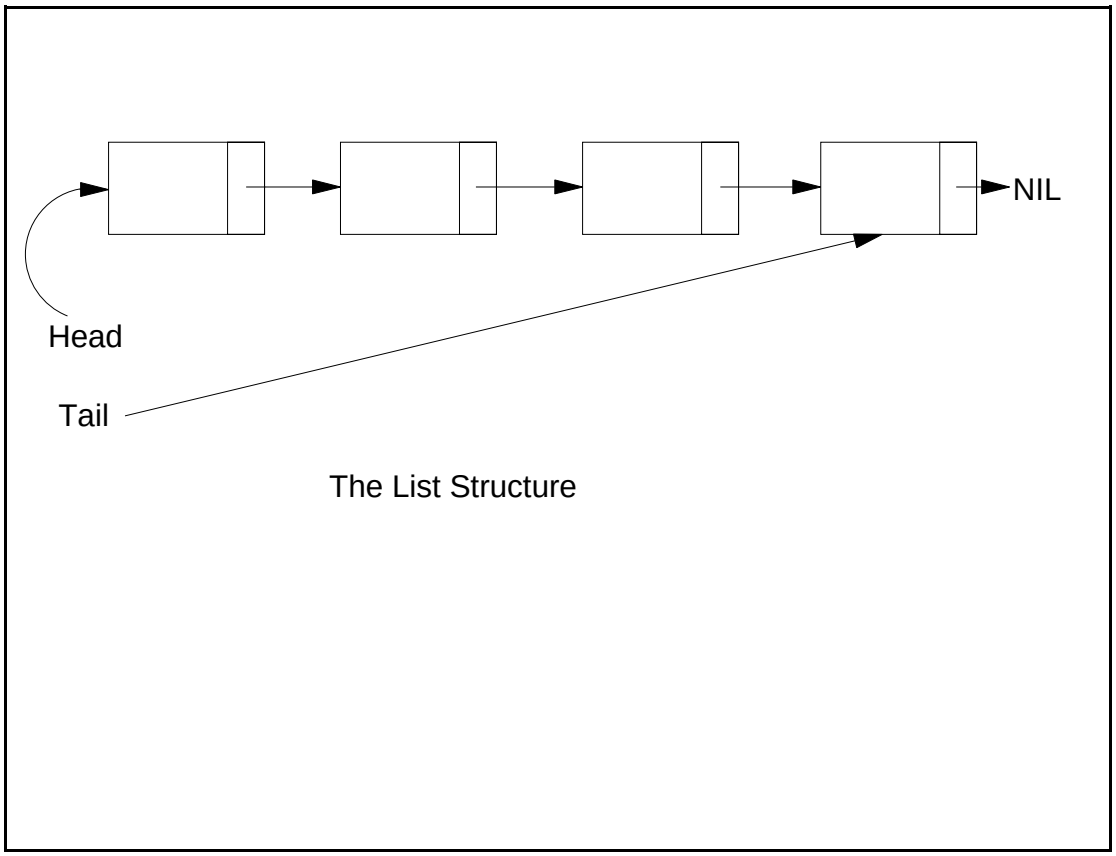


Figure IV.1 - Linked lists in PESYS

PESYS uses linked lists to store most of the data in the system. Since the lists are used without any particular ordering they are implemented using a very simple but effective mechanism, see **Figure IV.1**.

When a list is being created two pointers are used, one which points to the head (first element) and one which points to the tail (last element). For a new list, therefore, both head and tail point to NIL.

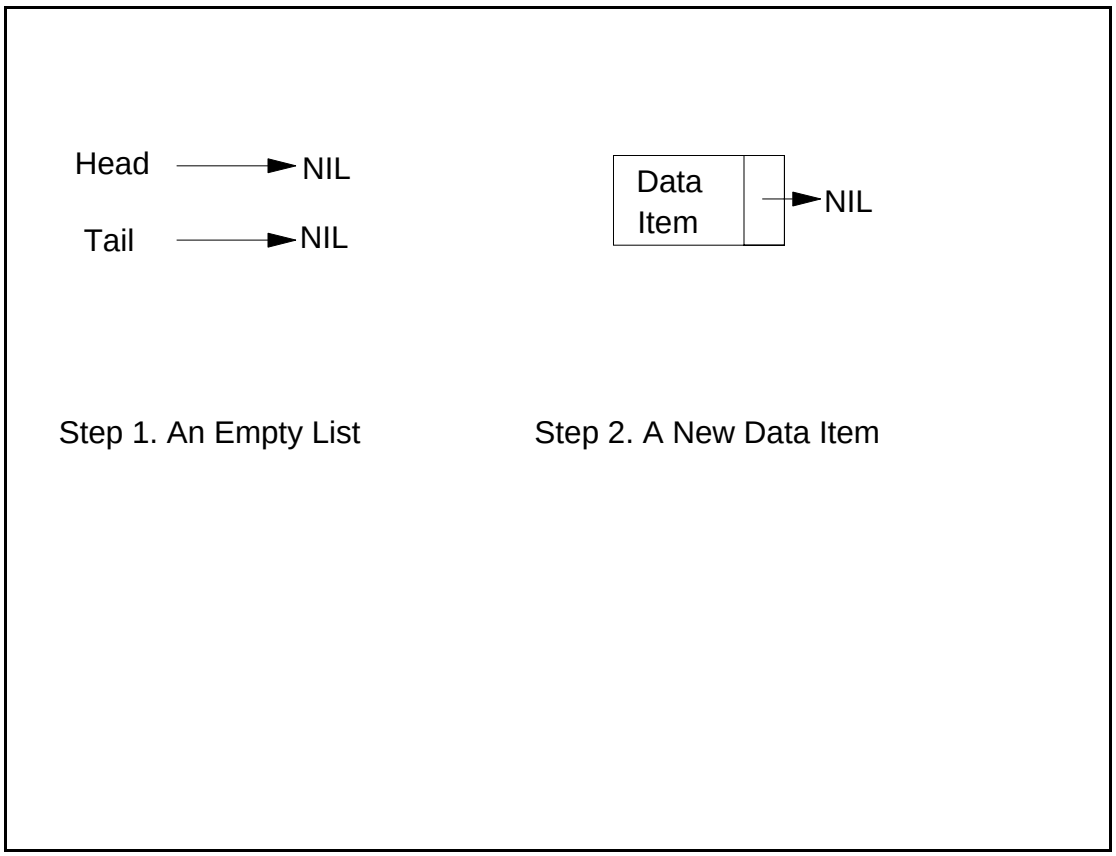


Figure IV.2 - Creating a new list

When a new element is to be added to the list it is first created dynamically. A special function is used to create each element and this function explicitly initialises any pointers in the element to NIL. Most of the problems with linked lists arise when pointers are not initialised to NIL. By making this process explicit errors of moving beyond the end of a list cannot arise. Any data is then added to the new element and it is placed at the end of the list. If the head of the list is NIL then the list is empty and the head is made to point to this new element. The tail also points to this element. When a second element is added to the list the head no longer points to NIL and so the new element must be added to the end of the list. The end of the list is marked by the tail pointer to tail^.next is made to point to the new element. This new element is now the last element in the list and so the tail pointer is updated. This process is shown graphically in **Figure IV.1**, **Figure IV.2** and **Figure IV.3**.

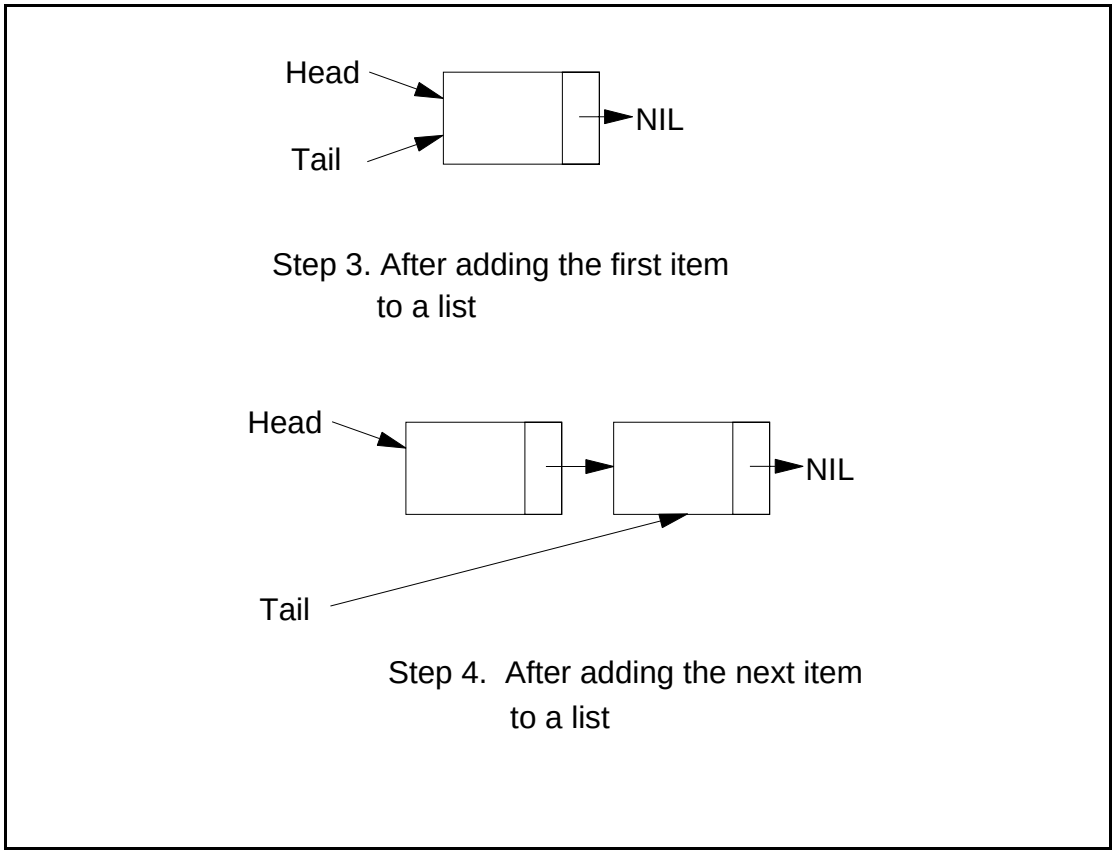


Figure IV.3 - Elements added to the list

The Sent_ptr list

An example of such a list is the sent_ptr which is used to represent lines of text, such as the clauses of rules. Each element contains a pointer to the dictionary and a number (real). It also contains an indication as to whether the element represents a word or a number. If the element represents a word then the pointer points to the word in the dictionary, otherwise it points to NIL. The number is either the actual number represented or 0.

The type definitions for this list, together with the routines for creating and the list are shown below:

```
TYPE
   sent_ptr = ^sent_rec;
   sent_rec = RECORD
              word_ptr : vocab_ptr; (*to dictionary*)
              numeric : boolean;
              number : real;
              next : sent_ptr; (*next element in the list*)
            END;
```

```
FUNCTION Make_sent_el : sent_ptr;

VAR
  tag : sent_ptr;

BEGIN
    NEW(tag);
    tag^.word_ptr := NIL;
    tag^.next := NIL;
    tag^.number := 0;
    tag^.numeric := FALSE;
    make_sent_el := tag;
END (*Make_sent_el*);

PROCEDURE Add_to_sent (VAR head, tail : sent_ptr; this_word : vocab_ptr; this_num : real; is_numeric : boolean);

VAR
  new_el : sent_ptr;

BEGIN
    new_el := make_sent_el;
    new_el^.word_ptr := this_word;
    new_el^.number := this_num;
    new_el^.numeric := is_numeric;



    IF head = NIL
      THEN BEGIN
           head := new_el;
           tail := new_el;
          END
      ELSE BEGIN
           tail^.next := new_el;
           tail := new_el;
          END;
END (*Add_to_sent*);
```

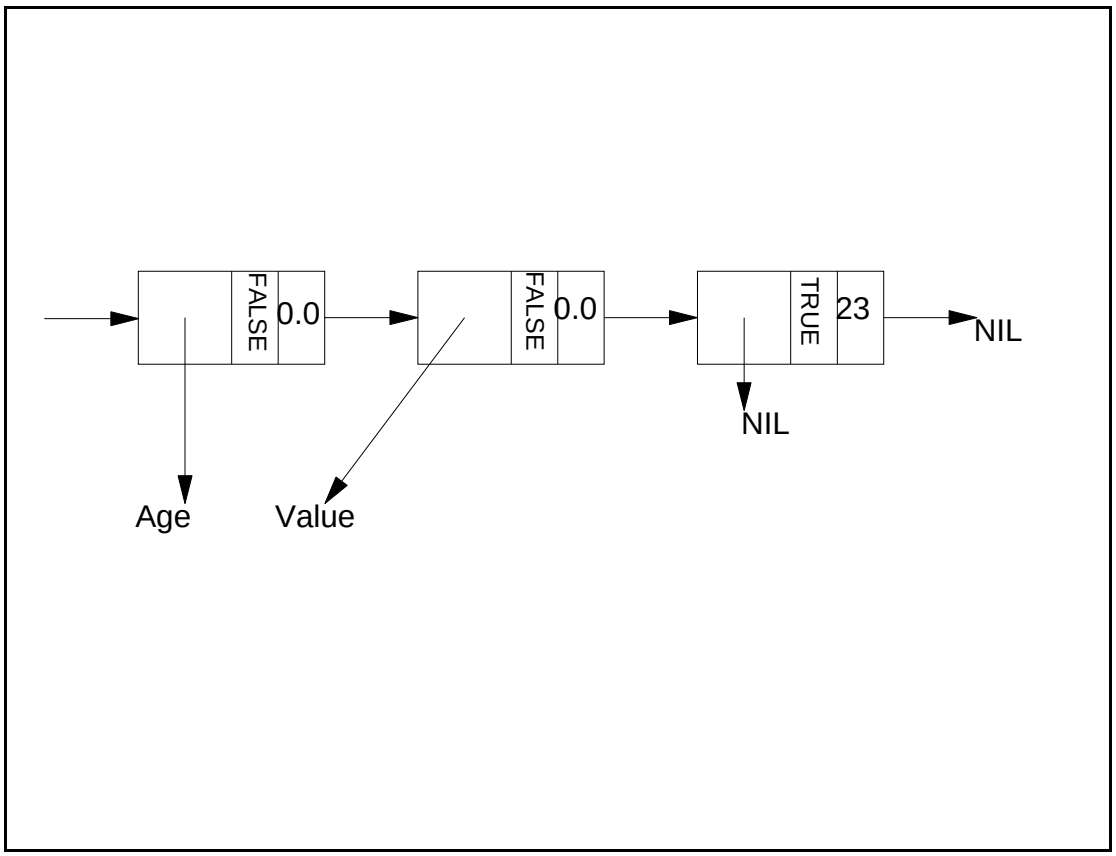


Figure IV.4 - A sentence list

Moving through a list

Moving through a list is a very simple process. A variable is assigned to point to the same element as the head. It then moves from the current element to the current^.next element until either some condition is met or the pointer points to NIL, indicating that the end of the list has been reached.

Most of the data structures in the knowledge base are based on these lists. For example, the rules are stored in a list and each rule is made up of a list of if-clauses and a list of then-clauses. Each if-clause is itself a list of or-clauses and the or-clauses and then-clauses make use of the sent_ptr list described above.

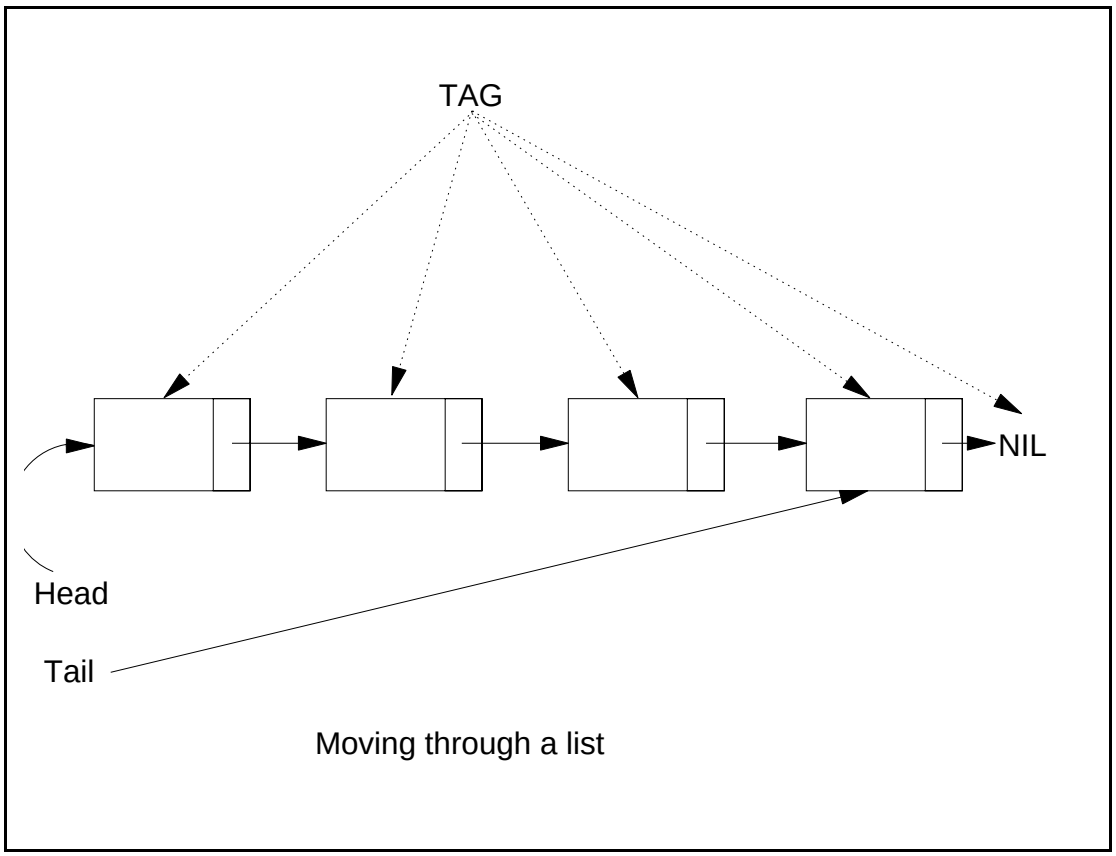


Figure IV.5 - Moving through a linked list

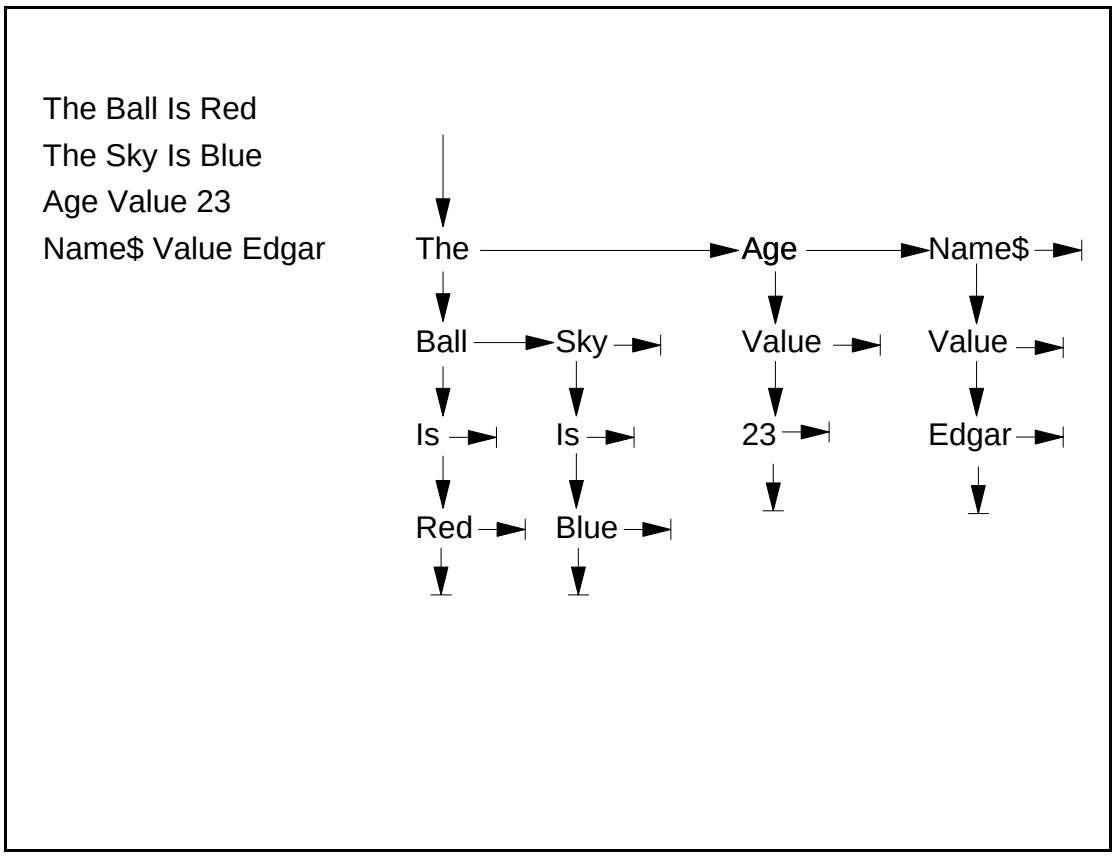


Figure IV.6 - The tree used for the working memory

IV.3 THE WORKING MEMORY

The working memory of the expert system is effectively a binary tree, however it is easier to visualise it in a slightly different way. Each node in the tree contains a pointer to the dictionary and a number (in the same way that the sent_ptr did). The node also has two pointers to the remainder of the tree. One of these pointers points "down", whilst the other points "across".

The tree is used as follows. If the tree is empty, or the head of the tree matches the current word in the clause being added, then follow the "down" branch, whilst if the words do not match, use the "across" branch. This is shown in **Figure IV.6**.

One immediate disadvantage with this approach is that the first level of the tree will become very congested since most words will not match the first word added to the tree and so heavy use will be made of the first "across" link. It is therefore sensible to represent the outer layer using an ordered binary tree to speed up access to the particular clause and this is shown in **Figure IV.7**.

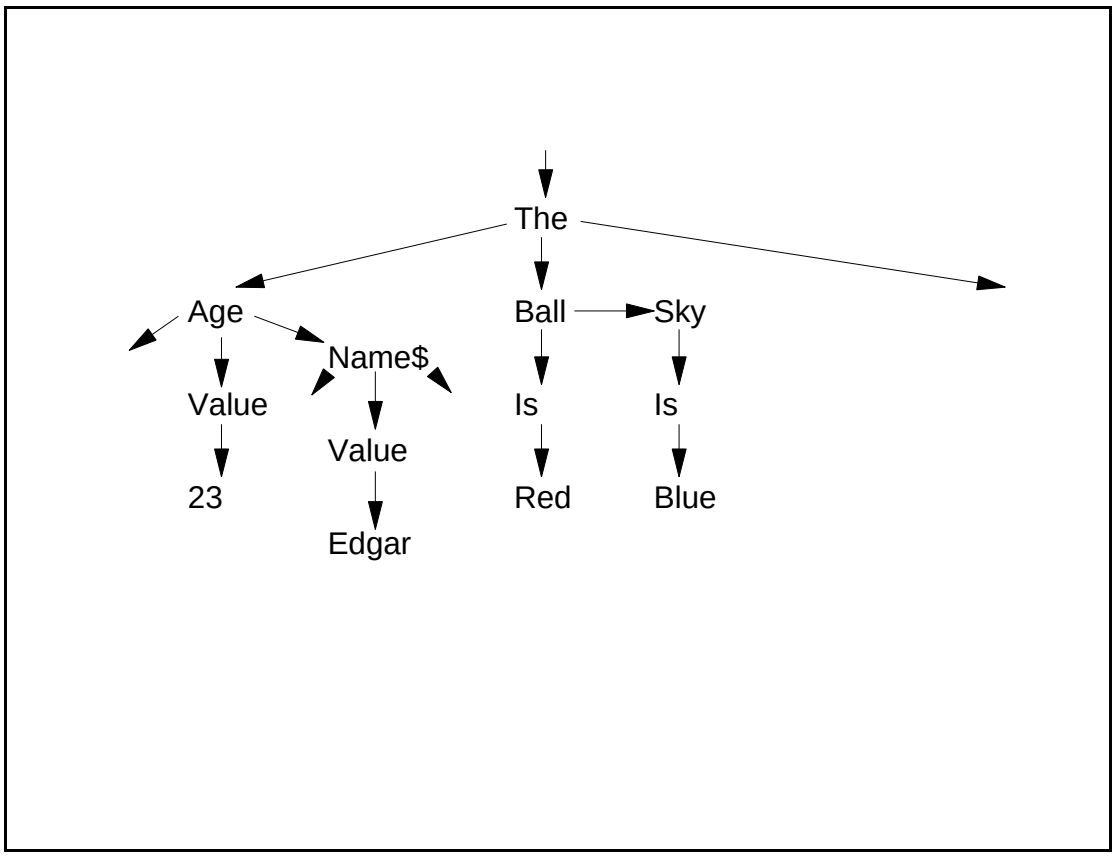


Figure IV.7 - A binary tree for the outer level

The use of the dictionary and PASCAL pointers means, however, that it is possible to improve this mechanism further. Each element in the dictionary tree can point directly to that part of the knowledge base that begins with that word. Access to the knowledge base from, for example, a sent_ptr clause will therefore involve no searching whatsoever. Instead two pointer movements are performed. The first goes from the sent_ptr to the dictionary and the second goes from the dictionary to the working memory tree at the point beginning with that word, this is shown in **Figure IV.8**. If the second pointer is not present, then this means that there is no data starting with that particular word.

In the PESYS system there are, in fact, two working memories. One which contains "permanent" information, the other which contains "temporary" information generated by running the knowledge base. This problem is overcome by simply having two pointers from the dictionary, one to each working memory.

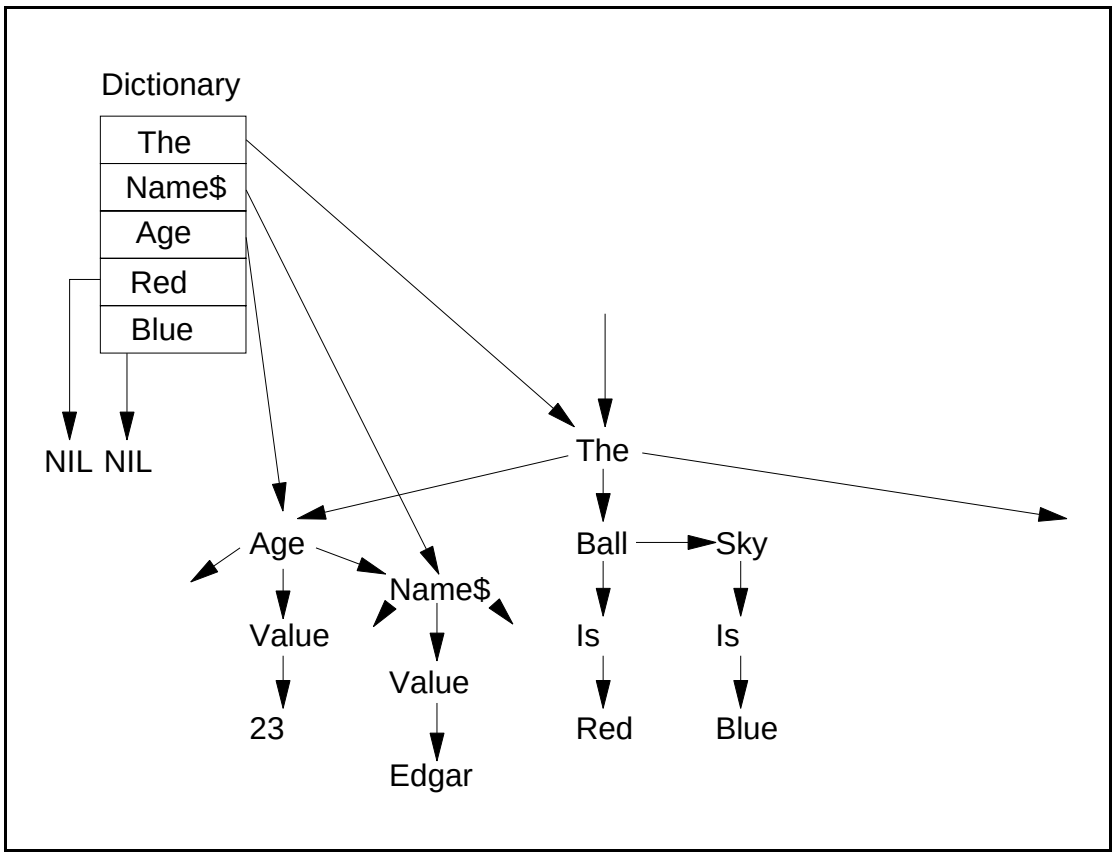


Figure IV.8 - The working memory as it is actually implemented in PESYS

Retrieving data from the working memory

The retrieval of data can be performed in two distinct ways. The first involves verifying that a particular fact exists in the working memory. This is done by following the clause down through the tree. If the complete clause is present then it is possible to obtain an indication of whether it is known to be true or false.

The second retrieval method involves a partial move through the tree. By using the first part of a clause, it is possible to determine all the possible endings to that clause since they will be found in the remainder of the tree. Thus, for example, entering Television Kind-of as a query will return a tree containing all the elements that a television is a kind-of. This tree can then be converted into a linear linked list for use by the program.

Deleting and replacing data

Data can be deleted and replaced by removing all those elements of the tree, starting from the bottom until an element is found which has another element "across" from it. Replacing data, such as for the What-if analysis, then simply involves deleting data and adding new data.

IV.4 NON-LINEAR DOCUMENTS

Non-linear documents are stored as records in a file and two files are used. The first stores the text of the screens and the second file is used to store information that links the various screens together.

A record in the screen file is made up of the text of the screen, together with information about whether the screen has been deleted and details about the first link associated with that screen. This link information is simply the position of the link element in the second file. The positions of screens that can be located using **PageUp (PgUp)** and **PageDown (PgDn)** are also stored in this record.

The link record contains information about the name of the link and where it is positioned on the screen. The location of the screen record that is being linked to is also stored, as is the location of the next link record for the current screen. Any of these links may be undefined and the end of a list of links is indicated by marking the link as being to NIL.

The final file used in non-linear documents is an index file and this is simply made up of records containing the names of the screens and their associated positions in the screen file.

When non-linear documents are created, care must be taken to add the new records to the appropriate position in the file. To this end, two arrays are maintained which keep track of free positions in the file. Each element in the array corresponds to a record in the file and the array is used to determine where the next free position in the file can be located.

APPENDIX V - CASE STUDIES

V.1 CASE STUDY I - AN EXPERT SYSTEM FOR THE COMPETITIVE USE OF INFORMATION SYSTEMS TECHNOLOGY

V.2 INTRODUCTION

The knowledge base is based on an article by Ives and Learmonth (1984) and is described by Doukidis and Whitley (1987).

Ives and Learmonth define an application of information systems technology as `strategic' if `it changes a firm's product or the way a firm competes in an industry' and they suggest the use of the customer resource life cycle as a means of investigating potential areas for using this technology. By concentrating on the needs of the customer, technology can be used to enhance customer service. This leads to greater customer loyalty and hence differentiates the firm from others in the market. The model they propose can be considered in four or thirteen stages and is presented below:

The thirteen stage model presented by Ives and Learmonth (1984)

SIMPLE MODEL	EXTENDED MODEL	DESCRIPTION
Requirements	Establish requirements	To determine how much of a resource is required
	Specify	To determine the attributes of the resource
Acquisition	Select source	To determine where customers will buy the resource
	Order	To order a quantity of the resource
	Authorize and pay for	To transfer funds or extend credit
	Acquire	To take possession of a resource
	Test and accept	To ensure that a resource meets specification
Stewardship	Integrate	To add to an existing inventory
	Monitor	To control access and use of a resource
	Upgrade	To upgrade a resource if conditions change
	Maintain	To repair a resource if necessary
Retirement	Transfer or dispose	To move, return, or dispose of inventory as necessary
	Account for	To monitor spending

The knowledge base was created by combining the model and the examples of information system technology that Ives and Learmonth present in each of the thirteen stages of the model. These examples were then converted into rules. The rules provided are by no means exhaustive, but provide a basic idea as to the usefulness of information systems technology for an organisation.

A number of rules from this knowledge base, together with information about the relations in the clauses is presented below.

V.3 THE RULES USED

rule-1
if
there is a large customer_base
and the customers are homogenous
then
inform 2 it is possible to create a resource analyzer

rule-2
if
a resource analyzer has been created
then
inform 2 the system may also allow the preparation of sales_proposals

rule-3
if
there is a resource analyzer being created
then
inform 2 an information system can help establish the requirements for the resources

rule-4
if
the suppliers of a service are distributed randomly
then
inform 2 an information system can help in directing suppliers to empty_areas

rule-5
if
an information system can direct suppliers to empty_areas
then
inform 2 an information system can help establish the requirements for the resources

rule-6
if
there are many items available
then
inform 2 there is a possibility of creating an inventory handler

rule-7
if
an inventory handler has been created
and there are many suppliers
then
inform 2 the system can dispatch request_for_quotation notes

rule-8
if
an inventory handler system has been created
then
inform 2 the resources required can be specified

rule-9
if
a resource has been chosen
and there are many suppliers
then
inform 2 an information system may suggest similar alternatives
inform 2 an information system may display the available choices

rule-10
if
an information system can display the available choices
then
inform 2 an information system can help select suppliers for resources

rule-11
if
there is a large customer_base
and orders are easily specified
then
inform 2 there is a possibility to create a round_the_clock order_entry_system for users

rule-12
if
a round_the_clock order_entry_system has been created
then
inform 2 an information system can assist in ordering resources

rule-13
if
payments for resources require credit_authorization
and the customer_base is large
then
inform 2 an information system can be used to provide automatic credit_authorization

rule-14
if
an information system provides automatic credit_authorization
then
inform 2 there is a possibility for a system to allow payment of bills

V.4 THE INFORMATION FOR RELATIONS

Large Kind-of Size
Size Kind-of Implies
Size Implies Dimensions
Dimensions Kind-of Relation
Dimensions Needs Object Size
Customer_base Kind-of Object
Homogenous Kind-of Relation
Homogenous Needs Object
Customers Kind-of Object
Create Kind-of Relation
Create Needs Object Processor
Resource Kind-of Object
Analyzer Kind-of Processor
Created Kind-of Implies
Created Implies Create
Allow Kind-of Relation
Allow Needs Action Object
Preparation Kind-of Action
Sales_proposals Kind-of Object
Establish Kind-of Relation
Establish Needs Need Object
Requirements Kind-of Need
Resources Kind-of Object
Distribute Kind-of Relation
Distribute Needs Object Probability
Randomly Kind-of Probability
Distributed Kind-of Implies
Distributed Implies Distribute
Suppliers Kind-of Object
Direct Kind-of Relation
Direct Needs Object Place
Empty_areas Kind-of Place
Directing Kind-of Implies
Directing Implies Direct
Many Kind-of Multitude
Multitude Kind-of Implies
Multitude Implies Quantity
Quantity Kind-of Relation
Quantity Needs Object Multitude
Items Kind-of Object
Creating Kind-of Implies
Creating Implies Create
Inventory Kind-of Object
Handler Kind-of Processor
Dispatch Kind-of Relation
Dispatch Needs Format Letter
Notes Kind-of Letter
Request_for_quotation Kind-of Format
Specify Kind-of Relation
Specify Needs Object
Specified Kind-of Implies

Specified Implies Specify
Chose Kind-of Relation
Chose Needs Object
Chosen Kind-of Implies
Chosen Implies Chose
Suggest Kind-of Relation
Suggest Needs Substitute
Alternatives Kind-of Substitute
Display Kind-of Relation
Display Needs Option
Choices Kind-of Option
Select Kind-of Relation
Select Needs Object
Source Kind-of Object
Orders Kind-of Object
Order_entry_system Kind-of Processor
Round_the_clock Kind-of Object
Order Kind-of Relation
Order Needs Object
Ordering Kind-of Implies
Ordering Implies Order
Require Kind-of Relation
Require Needs Action Permission Object
Payments Kind-of Action
Credit_authorization Kind-of Permission
Provide Kind-of Relation
Provide Needs Permission
Provides Kind-of Implies
Provides Implies Provide
Bills Kind-of Object
Payment Kind-of Action

V.5 CASE STUDY II - AN EXPERT SYSTEM TO ASSIST IN FILING INCOME TAX RETURNS

V.6 INTRODUCTION

The expert system to assist in filing tax returns for the Indian Income Tax authorities is described by Whitley *et al.* (1989).

Income taxes have long been the principal means of taxation in industrial countries. With relatively few distortions they can generate a great deal of revenue and leave scope for income redistribution. Experience in developing countries, however, suggests that personal income taxes are difficult to administer, raise little revenue, are weak in redistribution and

are often unfair. This has led to recent reforms of taxes on personal and company income which will often be necessary to enhance the revenue and efficiency of a tax system.

Personal Income Taxes account for about a tenth of total tax revenue in developing countries as against a higher proportion in the developed countries. The low yield reflects limited coverage and poor design. Improving the yield requires changes in the base rate, simplification in the procedure and law and more efficient administration and collection of these taxes.

In a major legislation amendment the Income tax act has been amended by Tax Laws. This is seen as an effort to simplify the tax laws and structure. Further, the use of computers is being encouraged at various collection and assessment centres with a view to increased efficiency in the filing and assessment of tax returns. Dependence on middlemen (experts - accountants and lawyers) is being discouraged with a view to allowing the filing of returns by the assessees themselves. Returns up to a specified amount are accepted without verification in good faith and a small percentage of these returns are put to sample check every year. There is thus a shift from avoidance of tax to tax planning.

In view of the shift in the expert's practice - stressing future tax planning instead of filing routine returns - an expert system was designed that would enable the completion of returns by the assessee himself or with the help of the junior staff of the expert. The overall aim was to provide an accurate portrayal of the role of the expert in filing tax returns, without requiring the expert to be present at the time.

V.7 DETERMINING THE RESIDENTIAL STATUS OF ASSESSEES

One of the activities that needs to be performed by the expert system is determining the residential status of the assessee. The piece of legislation relating to this is rather complicated and is presented below:

Residence in India
SECTION 6
For the purposes of this Act, -
(1) An individual is said to be resident in India in any previous year, if he --

(a) is in India in that year for a period or periods amounting in all to one hundred and eighty-two days or more; or
(b) 75[1][***]
(c) having within the four years preceding that year been in India for a period or periods amounting in all to three hundred and sixty-five days or more, is in India for a period or periods amounting in all to sixty days or more in that year.
76[2][Explanation : In the case of an individual, being a citizen of India, --
(a) who leaves India in any previous year for the purposes of employment outside India, the provisions of sub-clause (c) shall apply in relation to that year as if for the words "sixty days", occurring therein, the words "one hundred and eighty-two days" had been substituted;
(b) who being outside India, comes on a visit to India in any previous year, the provisions of sub-clause (c) shall apply in relation to that year as if for the words "sixty days", occurring therein, the words "ninety days" had been substituted.]

(2) Not applicable to the system.
(3) Not applicable to the system.
(4) Not applicable to the system.
(5) Not applicable to the system.

(6) A person is said to be "not ordinarily resident" in India in any previous year if such is --
(a) an individual who has not been resident in India in nine out of the ten previous years preceding that year, or has not during the seven previous years preceding that year been in India for a period of, or periods amounting to, seven hundred and thirty days or more;
(b) Not applicable to the system.

75 Omitted by the Finance Act, 1982, w.e.f. 1-4-1983
76 Substituted by the Finance Act, 1982, w.e.f. 1-4-1983. Original explanation was inserted by the Finance Act, 1978, w.e.f. 1-4-1979.

V.8 THE RULES FOR DETERMINING THE RESIDENTIAL STATUS OF AN ASSESSEE

```
control of residential status
if
* We must not have any mixed chaining for the system !!!!!
we have found the residential status
then
print 'Residential status is :' res_status$
writeln 'Residential status is :' res_status$
press_key
inform 1 we have residential status


rule for not ordinarily resident
if
we have found person is not ordinarily resident
then
let res_status$ = n o t ordinarily resident
we have found the residential status
```

```
rule for resident
if
we have found person is resident
then
let res_status$ = resident
we have found the residential status

rule for non resident
if
we have found the person is non resident
then
let res_status$ = non resident
we have found the residential status

rule for resident 0
if
we know person is resident
then
we have found person is resident

rule for resident 1
if
enter total_number_of_days_last_year_spent_in_india prompt
Q1
total_number_of_days_last_year_spent_in_india >= 182
then
we know person is resident

rule for resident 3
if
total_number_of_days_last_year_spent_in_india >= 182
person has been in India for a total of at least 365 days in the preceding 4 years prompt
Q2
and person has left
then
we know person is resident

rule for resident 4
if
total_number_of_days_last_year_spent_in_india >= 90
person has been in India for a total of at least 365 days in the preceding 4 years prompt
Q2
and person has visited
then
we know person is resident
```

rule for non resident
if
total_number_of_days_last_year_spent_in_india < 60
then
we have found the person is non resident

rule for non resident
if
total_number_of_days_last_year_spent_in_india >= 60
person has not been in India for a total of at least 365 days in the preceding 4 years prompt Q2
then
we have found the person is non resident

rule for non resident since left
if
total_number_of_days_last_year_spent_in_india < 182
person has been in India for a total of at least 365 days in the preceding 4 years prompt Q2
and person has left
then
we have found the person is non resident

rule for non resident since visit
if
total_number_of_days_last_year_spent_in_india < 90
person has been in India for a total of at least 365 days in the preceding 4 years prompt Q2
and person has visited
then
we have found the person is non resident

rule for resident 2
if
total_number_of_days_last_year_spent_in_india >= 60
person has been in India for a total of at least 365 days in the preceding 4 years prompt Q2
and person has not left
and person has not visited
then
we know person is resident

rule for person not ordinarily resident
if
we know person is resident

and person has not been resident for 9 out of 10 previous years prompt
Q5
or person has not been in India for 730 days in the previous 7 years prompt
Q6
then
we have found person is not ordinarily resident

rule to check for leaving
if
you are an Indian citizen who has left India in previous year for the purpose of employment outside India prompt
Q3
then
person has left

rule to check for not leaving
if
you are an Indian citizen who has not left India in previous year for the purpose of employment outside India prompt
Q3
then
person has not left

rule to check for visiting
if
you are an Indian citizen who is abroad, comes on a visit to India in the previous year prompt
Q4
then
person has visited

rule to check for not visiting
if
you are not an Indian citizen who is abroad, comes on a visit to India in the previous year prompt
Q4
then
person has not visited

V.9 THE PROMPTS FOR THE RULE FILE

Q1 As per Sec 6(1)(a) An individual is said to be resident of India in any previous year if he is in India in that year for a period or periods amounting in all to 182 days or more.
Q2 Sec 6(1)(c) Normally an Individual is a resident of India in any previous year if he has within the 4 years preceding that year been in India for a period or periods of at least 365 days and is in India for at least 60 days in that year.

Q3 In the case of an individual being a citizen of India who leaves India for the purposes of employment outside India 60 days, in subclause (c) would be substituted by 182 days.
Q4 In the case of an individual being a citizen of India who being outside India comes on a visit to India in any previous year 60 days in subclause (c) would be substituted by 90 days.
Q5 Sec 6(6)(a) A person is said to be not ordinarily resident in India in any previous year if such person is an individual who has not been resident in India in 9 out of 10 previous years preceding that year Or Next Question.
Q6 Sec 6(6)(a) A person is said to be not ordinarily resident in India in any previous year if such person is an individual who has not been resident in India for a period of or periods amounting in all to 730 days or more.

V.10 THE INFORMATION ABOUT VARIABLES FOR THE RULE FILE

Total_number_of_days_last_year_spent_in_india INPUT Please Enter The Total Number Of Days Spent In India In The Previous Year
Total_number_of_days_last_year_spent_in_india Range 0.0 TO 366.0
Total_number_of_days_last_year_spent_in_india Dp 0.0

REFERENCES

Austin, J. L., *How to do things with words: The William James lectures delivered at Harvard University in 1955*, London: Oxford University Press, 1962.

Backhouse, James and Jonathan Liebenau, *Understanding information: An introduction*, London: Macmillan, 1990.

Ball, Ian, Man can still hold computer in check, *The Daily Telegraph*, (October 24th, 1989), page 1.

Benner, Patricia, *From novice to expert: excellence and power in clinical nursing practice*, Reading, MA:, Addison-Wesley, 1984.

Black, W. J., *Intelligent knowledge based systems: An introduction*, Wokingham: Van Nostrand Reinhold, 1986.

Blakemore, Colin and Susan Greenfield (eds.) *Mindwaves: Thoughts on intelligence, identity and consciousness* Oxford: Basil Blackwell, 1987.

Bloomfield, Brian P., *The question of artificial intelligence: Philosophical and sociological perspectives*, London: Croom Helm, 1987.

Bloomfield, Brian P., Expert systems and human knowledge: A view from the sociology of science, *AI & Society*, Volume 2, Number 1, (January - March 1988), pages 17-29.

Boden, Margaret A., *Artificial intelligence and natural man* (second edition, expanded), New York: Basic Books, 1977, 1987.

Bodkin, Tim and Ian Graham, Case studies of expert systems development using microcomputer software packages, *Expert systems: the international journal of knowledge engineering*, Volume 6, Number 1, (February 1989), pages 12-16.

Boisgontier, J. and C. Donay, *File handling in Turbo Pascal*, London: Paradigm, 1988.

Boley, Harold, Expert system shells: very-high-level languages for artificial intelligence, *Expert systems: the international journal of knowledge engineering*, Volume 7, Number 1, (February 1990), pages 2-8.

Bolter, J. David, *Turing's man: Western culture in the computer age*, London: Duckworth, 1984.

Borland Inc., *Turbo Pascal V3.0*, 1985.

Borland Inc., *Turbo Pascal V4.0*, 1987.

Born, Rainer P. (Ed.), *Artificial intelligence: The case against*, London: Croom Helm, 1987.

Brownston, L., R. Farrell, E. Kant and N. Martin, *Programming expert systems in OPS5*, Reading, MA: Addison-Wesley, 1985.

Butler, C., E. Hodil and G. Richardson, Building knowledge base systems with procedural languages, *IEEE Expert*, Volume 3, Number 2, (Summer 1988), pages 47-59.

Capper, Phillip and Richard Susskind, *Latent damage law: The expert system*, London: Butterworths, 1988.

Checkland, Peter, *Systems thinking, systems practice*, Chichester: John Wiley & Sons, 1981.

Checkland, P. B., Information systems and systems thinking: Time to unite?, *International Journal of Information Management*, Volume 8, (1988), pages 239-248.

Church, Chas, Xi+ with the brakes off, *Expert systems user*, Volume 5, Number 3, (March 1989), pages 18-21.

Clarke, Roger A., Information technology and dataveillance, *Communications of the ACM*, Volume 31, Number 5, (May 1988), pages 498-512.

Cohen, Paul R., and Edward A. Feigenbaum (Eds.), *The handbook of artificial intelligence: Volume III*, Reading, MA: Addison-Wesley, 1982.

Collins, H. M., R. H. Green and R. C. Draper, Where's the expertise? Expert systems as a medium of knowledge transfer, in *Expert Systems 85* (ed. Martin Merry), Cambridge: Cambridge University Press, 1985, pages 323-334.

Collins, H. M., *Expert systems, artificial intelligence and the behavioural co-ordinates of skill*, in Bloomfield (1987), pages 258-281.

Cooley, Mike, *Architect or Bee? The human price of technology*, London: A Tigerstripe book, Hogarth Press, 1987.

Cooley, Mike, The human use of expert systems, *Aries at City*, Quarterly Review No. 2, (August 1988).

Cornford, Tony and Barbera Farbey, *User representation in large systems: the case of the DHSS* (presented at the Unicom Seminar on Participation, London 1987), Working Paper 16, Information Systems Department, London School of Economics and Political Science, 1989.

Coulter, J., On comprehension and `mental representation' in Gilbert and Heath (1985), pages 8-23.

Crookes, John G., and B. Valentine, Simulation in microcomputers, *Journal of the Operational Research Society*, Volume 33, Number 9, (September 1982), pages 855-858.

Crookes, John G., David W. Balmer, Sew Tee Chew and Ray J. Paul, *Journal of the Operational Research Society*, Volume 37, Number 6, (June 1986), pages 603-618.

Crookes, John G., Simulation using C, in *Computer modelling for discrete simulation* (ed. Michael Pidd), Chichester: John Wiley and sons, 1989.

Report of a working party - Council for Science and Society, *Benefits and risks of knowledge based systems*, Oxford: Oxford University Press, 1989.

D'Agapeyeff, A., and C. J. B. Hawkins, Expert systems in UK business: A critical assessment, *The knowledge engineering review*, Volume 2, Number 3, (September 1987), pages 185-201.

Doukidis, Georgios I., and Ray J. Paul, ASPES: A Skeletal Pascal Expert System, in *Expert systems and artificial intelligence in decision support systems* (eds. H. G. Sol, C. A. Th. Takkenberg, Robbe P. F. De Vries), The Netherlands: Reidel Publishing, 1987, pages 227-246.

Doukidis, Georgios I., and Edgar A. Whitley, Developing and running expert systems with PESYS, *Future Generation Computer Systems*, Volume 3, Number 3, (September 1987), pages 189-199.

Doukidis, Georgios I., Marios C. Angelides and James L. Harlow, Towards an intelligent tutoring system for Pascal programming, *Education and Computing*, Volume 4, (1988b), pages 273-286.

Doukidis, Georgios I., Vijal P. Shah and Marios C. Angelides, *Lisp: From foundations to applications*, Bromley: Chartwell-Bratt, 1988a.

Doukidis, Georgios I. and Edgar A. Whitley, *Developing Expert Systems*, Bromley: Chartwell-Bratt, 1988.

Doukidis, Georgios I., and Ray J. Paul, A survey of the application of artificial intelligence techniques within the OR society, forthcoming in *Journal of the Operational Research Society*, 1990a.

Doukidis, Georgios I., and Ray J. Paul, *SIPDES: A simulation program debugger using an expert system*, Internal report, CASM group, Information Systems Department, London School of Economics and Political Science, 1990b.

Dreyfus, Hubert L. and Stuart E. Dreyfus with Tom Athanasiou, *Mind over machine: The power of human intuition and expertise in the era of the computer* (updated, paperback edition), New York: The Free Press, 1986a.

Dreyfus, Hubert L. and Stuart E. Dreyfus, Competent systems: the only future for inference making computers, *Future Generation Computer Systems*, Volume 2, Number 4, (December 1986b), pages 233-244.

Dreyfus, Hubert L. and Stuart E. Dreyfus, Making a mind versus modelling the brain: Artificial intelligence back at a branch point, *Dædalus*, Winter 1988, pages 15-43.

Earnest, Les, Can computer cope with human races, *Communications of the ACM*, Voume 32, Number 2, (February 1989), pages 174-182.

Feigenbaum, Edward A. and Pamela McCorduck, *The fifth generation: Artificial intelligence and Japan's computer challenge to the world*, Reading, MA: Addison-Wesley, 1983.

Feigenbaum, Edward A., Pamela McCorduck and H. Penny Nii, *The rise of the expert company: How visionary companies are using artificial intelligence to achieve higher productivity and profits*, London: Macmillan, 1988.

Florentin, J. J., Software review: KEE, *Expert systems: the international journal of knowledge engineering*, Volume 4, Number 2, (May 1987), pages 118-120.

Forsyth, Richard, Software review: Xi+, *Expert systems: the internation journal of knowledge engineering*, Volume 4, Number 1, (February 1987), pages 48-51.

Forsyth, Richard, Software review: Leonardo, *Expert systems: the international journal of knowledge engineering*, Volume 5, Number 2, (May 1988), pages 160-164.

Gammack, John G., and Anthony Anderson, Constructive interaction in knowledge engineering, *Expert systems: the international journal of knowledge engineering*, Volume 7, Number 1, (February 1990), pages 19-26.

Genesereth, Michael R., and Nils J Nilsson, *Logical foundations of artificial intelligence*, Los Altos, CA: Morgan Kaufman, 1987.

Gilbert, G. N. and C. Heath, (eds.) *Social actions and artificial intelligence: Surrey conference on sociological theory and method; 3*, Aldershot: Gower Publishing, 1985.

Grégoire, E., Evaluation of the expert system tools KEE and ART, *Applied artificial intelligence, an international journal*, Volume 2, Number 1, (1988), pages 1-23.

Gunderson, Keith, *Mentality and machines* (second edition), London: Croom Helm, 1971, 1985.

Hall, Lawrence O., and Abraham Kandel, Toward a methodology for building expert systems for imprecise domains, *International journal of expert systems*, Volume 1, Number 3, (1988), pages 237-251.

Harmon, Paul and David King, *Expert systems: Artificial intelligence in business*, Chichester: John Wiley & Sons, 1985.

Harmon, Paul, R. Maus and W. Morrissey, *Expert systems, tools and applications*, Chichester: John Wiley and sons, 1988.

Hart, Anna, *Knowledge acquisition for expert systems*, London: Kogan Page, 1986.

Heidegger, Martin, *Being and time* (translated by John Macquarrie and Edward Robinson), Oxford: Basil Blackwell, 1962.

Hinde, C., R. Allwood, D. Steward and B. Negus, *Evalutaion of expert system shells for construction industry applications*, Department of Civil Engineering, Loughborough University of Technology, August 1985.

Hirschheim, Rudy and Heinz K. Klein, Four paradigms of information systems development, *Communications of the ACM*, Volume 32, Number 10, (October 1989), pages 1199-1216.

Ives, Blake and Gerald P. Learmonth, The information system as a competitive weapon, *Communications of the ACM*, Volume 27, Number 12, (December 1984), pages 1193-1201.

Josefson, Ingela, The nurse as an engineer, *AI & Society*, Volume 1, Number 2, (October - December 1987a), pages 115-126.

Josefson, Ingela, Knowledge and experience, *Applied artificial intelligence, an international journal*, Volume 1, Number 2, (1987b), pages 173-180.

Kaplan, Simon M. and Medhi T. Harandi, Expert assistance in conversational design tools, *Proceedings of CASE '89, The 3rd Annual Workshop on CASE* (July 17-21), BCS/IEEE, London, 1989.

Keen, Peter G. W., Information systems and organizational change, *Communications of the ACM*, Volume 24, Number 1, (January 1981), pages 24-33.

Keller, R., *Expert system technology*, Hemel Hempstead: Yourdon Press, Prentice Hall, 1987.

Kent, William, *Data and reality: Basic assumptions in data processing reconsidered*, Amsterdam: North-Holland, 1978.

Kowalski, Robert, Position statement, *Sigart Newsletter*, Number 70, (February 1980), page 44.

Land, Frank F., Tony Cornford and Georgios I. Doukidis, *In search of the expert systems product* (presented at the IFIP Joint International Symposium on Information Systems, Sydney, March 1988), Working Paper 17, Information Systems Department, London School of Economics and Political Science, 1989.

Leonard-Barton, Dorothy and John J. Sviokla, Putting expert systems to work, *Harvard Business Review*, Volume 66, Number 2, (March-April 1988), pages 91-98.

Levy, Zeeva, *The software estimation process, a small step forward*, Working Paper 23, Information Systems Department, London School of Economics and Political Science, 1990.

Linderholm, Owen, Screentest: Crystal & VP-Expert, *Personal Computer World*, Volume 10, Number 4, (April 1987), pages 142-146.

Lipscombe, Barrie, Expert Systems and computer controlled decision making in medicine, *AI & Society*, Volume 3, Number 3, (July - September 1989), pages 184-197

Lipsey, Richard G., *An introduction to positive economics* (7th edition), London: Wiedenfeld and Nicholson, 1989.

Lyytinen, Kalle, Two views of information modelling, *Information and Management*, Volume 12, Number 1, (1987), pages 9-19.

Massotte, A-M, M. Maury and H. Betaille, An experience in knowledge engineering, in *The proceedings of the second international expert systems conference*, London, 1986, pages 229-235.

McCorduck, Pamela, *Machines who think: A personal inquiry into the history and prospects of Artificial Intelligence*, New York: W. H. Freeman and Co., 1979.

McCoy, Kathleen F., Generating context-sensitive responses to object-related misconceptions, *Artificial Intelligence*, Volume 45, Number 2, (December 1989), pages 157-195.

Michie, Donald and Rory Johnston, *The creative computer: Machine intelligence and human knowledge*, Harmondsworth: Viking, 1984.

Microsoft Inc., *Pascal V3.32*, 1986.

Minsky, Marvin, *The society of mind*, New York: Simon and Schuster, 1986. BF431 M66

Mumford, Enid and Mary Weir, *Computer systems in work design: The ETHICS method. Effective Technical and Human Implementation of Computer Systems*, London: Associated Business Press, 1979.

Nuttall, S., Nexpert makes an expert of you, *Expert systems user*, Volume 4, Number 5, (July 1988), pages 12-13.

Paris, Jeff, *Advice to those about to work on inexact reasoning*, Lecture Notes for the SERC Logic for IT seminar, Glasgow, September 1988.

Partridge, Derek, The scope and limitation of first generation expert systems, *Future Generation Computer Systems*, Volume 3, Number 1, (February 1987), pages 1-10.

Paul, Ray J., *Simulation Modelling: The CASM project* (presented at The Annual Operational Research Symposium of Yugoslavia, Brioni, Yugoslavia, 11-14 October 1988 and at the 2nd Brazilian Workshop on Simulation, Sao Jose dos Campos, Sao Paulo, Brazil, 1-2 September 1988), Working Paper 18, Information Systems Department, London School of Economics and Political Science, 1989.

Polanyi, Michael, *The tacit dimension*, London: Routledge & Kegan Paul, 1967.

Polanyi, Michael, *Knowing and being: Essays by Michael Polanyi* (edited by Marjorie Grene), London: Routledge & Kegan Paul, 1969.

Prosch, Harry, *Michael Polanyi: A critical exposition*, Albany, NY: State University of New York Press, 1986.

Quinlan, J. Ross, *Applications of Expert Systems, The proceedings of the second Australian conference: Volume one*, Reading, MA: Addison-Wesley/Turing Institute Press, 1987.

Rajan, Tim, Goldhill finds the midas touch, *Expert systems user*, Volume 4, Number 3, (May 1988), pages 14-16.

Ramsay, Allan, *Formal methods in artificial intelligence*, Cambridge: Cambridge University Press, 1988.

Richer, Mark H., An evaluation of expert system development tools, *Expert Systems: the international journal of knowledge engineering*, Volume 3, Number 3, (July 1986), pages 166-183.

Roth, Alan, Bridging the gap?, *Expert systems user*, Volume 4, Number 1, (March 1988), pages 10-12.

Roth, E. M., K. B. Bennett and D. D. Woods, Human interaction with an "intelligent" machine, *International Journal of Man-Machine Studies*, Volume 27, Number 5/6, (November/December 1987), pages 479-525.

Rousset, Marie-Christine and Brigitte Safar, Negative and positive explanations in expert systems, *Applied artificial intelligence, an international journal*, Volume 1, Number 1, (1987), pages 25-38.

Samuel, Arthur L., AI, Where it has been and where it is going, *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 1983, Volume 2, pages 1152-1157.

Santene, Ano, *The impact of expert systems in financial institutions*, M.Sc. report, Information Systems Department, London School of Economics and Political Science, 1989.

Schank, Roger C. with Peter G Childers, *The cognitive computer: On language, learning and artificial intelligence*, Reading, MA: Addison-Wesley, 1984.

Searle, John R., *Speech Acts: An essay in the philosophy of language*, Cambridge: Cambridge University Press, 1969.

Searle, J. R., Minds, brains and programs (with commentary and author's response), *The Behavioral and Brain Sciences*, Volume 3, Number 3, (September 1980), pages 417-457.

Searle, John R., *Minds, brains and science: The 1984 Reith lectures*, London: British Broadcasting Corporation, 1984.

Searle, J. R., Minds and brains without programs, in Blakemore and Greenfield (1987), pages 209-233.

Sergot, M. J., F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond and H. T. Cory, The British Nationality Act as a logic program, *Communications of the ACM*, Volume 29, Number 5, (May 1986), pages 370-386.

Smithson, Steve and Rudy Hirschheim, *End-user computing: a debate on the user-system interface* (revised version of a paper presented to IFIP TC 8.2 Working Conference on Desktop Technology, Cornell, June 1989), Working Paper 14, Information Systems Department, London School of Economics and Political Science, 1989.

Stamper, Ronald, Management epistemology: Garbage in, garbage out (and what about deontology and axiology), in *Knowledge representation for decision support systems* (eds. L. B. Methlie and R. H. Sprague), Amsterdam: North-Holland, 1985, pages 55-77.

Stamper, Ronald, James Backhouse, Sunny Marche and Karl Althaus, Meaning: The frontier of Informatics - Semantic Normal Form?, *Proceedings of the Aslib Conference Informatics-9*, 1987.

Stamper, Ronald, Pathologies of AI: Responsible use of artificial intelligence in professional work, *AI & Society*, Volume 2, Number 1, (January - March 1988), pages 3-16.

Suchman, Lucy A., *Plans and situated actions: The problem of human machine communication*, Cambridge: Cambridge University Press, 1987.

Susskind, Richard E., *Expert systems in law: A jurisprudential inquiry*, Oxford: Clarendon Press, 1987.

Tilghman, B. R., Seeing and seeing-as, *AI & Society*, Volume 2, Number 4, (October - December 1988), pages 303-313.

Turing, A. M., Computing machinery and intelligence, *Mind*, Volume LIX, Number 236, (October 1950), pages 433-460.

Turkle, Sherry, *The second self: Computers and the human spirit*, New York: Simon and Schuster, 1984.

Twine, Steven, Towards a knowledge engineering procedure, in *Expert systems V* (eds. B. Kelly and A. Rector), Cambridge: Cambridge University Press, 1989, pages 90-102.

van Koppen, J. and C. Philips, A survey of expert system development tools, in *The proceedings of the second international expert systems conference*, London, 1986, pages 157-173.

Vedder, Richard G., PC-based expert system shells: some desirable and less desirable characteristics, *Expert systems: the international journal of knowledge engineering*, Volume 6, Number 1, (February 1989), pages 28-42.

Vrba, Joseph A., and Juan A. Herrera, Expert system tools: The next generation, *IEEE Expert*, Volume 4, Number 1, (Spring 1989), pages 75-76.

Waller, Paul, *General purpose expert system tools sold in Britain*, presented at a meeting of the OR society study group on aritifical intelligence and expert systems, London School of Economics and Political Science, October 25th, 1989.

Wallsgrove, Ruth, Screentest: Crystal, *Personal Computer World*, Volume 11, Number 11, (November 1988), pages 172-175.

Waterman, Donald A., Jody Paul and Mark Peterson, *Expert Systems for legal decision making*, in Quinlan (1987), pages 23-47.

Weizenbaum, Joseph, *Computer power and human reason: From judgement to calculation*, San Francisco: W. H. Freeman and co., 1976.

Weizenbaum, Joseph, ELIZA a computer program for the study of natural language communication between man and machine, *Communications of the ACM*, Volume 26, Number 1, (January 1983), pages 23-28.

Whitley, Edgar A., Ashwajit Singh and Georgios I. Doukidis, An expert system to assist in filing tax returns: The case of Indian income tax, in *The proceedings of the fifth international expert systems conference*, London, 1989, pages 115-129.

Winograd, Terry, *Understanding natural language*, Edinburgh: Edinburgh University Press, 1972.

Winograd, Terry and Fernando Flores, *Understanding computers and cognition: A new foundation for design*, Reading, MA: Addison-Wesley, 1986.

Winograd, Terry, Where the action is, *Byte*, Volume 13, Number 13, (December 1988), pages 256A-258.

Winston, Patrick H. and Berthold K. P. Horn, *Lisp* (second edition), Reading, MA: Addison-Wesley, 1984.

Winston, Patrick H., *Artificial intelligence* (second edition), Reading, MA: Addison-Wesley, 1984.

Wittgenstein, Ludwig, *Philosophical Investigations* (trans. G. E. M. Anscombe), Oxford: Basil Blackwell, 1953.