The Use of Expert System Shells in Education: an Explanation-Based Approach

Karen Valley

Ph.D.

University of Edinburgh

1990

Declaration

I hereby declare that this thesis has been composed by myself and that the work described in it is my own.

Karen Valley

Acknowledgements

Thanks are owed to the following:

- The Economic and Social Research Council for financial support

- My supervisors, Peter Ross and Tom Conlon, for their help and encouragement along the way. Sorry it took so long.

- Research Machines Ltd; Bob Lewis, Alan Greenwell, Ken Tate and Alan Martin; the teachers at Moray House College, St Martin's College, and the Computer Based Learning Unit at Leeds University who helped with the initial evaluation; Ian King, Norman Bowman, David Bethune, Ann Cole, Paul Brna, Helen Lowe and Tom Conlon, who helped (at very short notice) with the final evaluation.

- Helen Pain for providing the necessary employment which got me through my fourth year, and also for being a good friend; Judy Delin, Jane Hillston and Richard White for providing food and friendship, and helping me through the bad bits; Paul Brna for helping me with all my LaTeXproblems; and to Frank Gormlie for providing me with a roof over my head and the occasional glass of whisky, both of which were much-needed and appreciated. Special thanks are also due to Bill McGrew for encouraging me on to this path in the first place, and for his love and support.

- My family, who have been very patient and supportive throughout my student years. Love and thanks are due in particular to my mother, Anna Stones, but also my sister, Lesley Pacitti, my aunt, Nan McKenzie, and my uncle, Nicky McKenzie. I would also like to thank Derek Stones, Graeme, Ashley and Allan Pacitti, Alison McKenzie and Larry Cartwright for all they have done over the years.

Finally, I would like to mention two people who have contributed more than can ever be quantified. The first is Carol Glen, who left this country in August 1989 to study in the USA. A close friend for many years, she has been an endless source of love, support, good advice and most of all fun. Life is not the same without her. The second is Rod Moyse, who is proof that there are indeed worthwhile things to be gained from attending academic conferences. He, more than any other, helped me get down and tame the beast when it looked like taking over. He is a kindred spirit, and I look forward to our future together.

Published Papers

The following published papers report aspects of the work described in this thesis:

- The overall structure of the shell, as described in Chapter 6, is outlined in "Designing an Expert System Shell for use in Education", published in the proceedings of the *PICKET Education Group Conference* (1988)

- An outline of the explanation categories and question-templates available in the EXPLORES explanation tool appears in "Explanation Generation in an Expert System Shell", published in the proceedings of the *4th Alvey Explanation Workshop* (1988)

- The initial evaluation and the design criteria which emerged from it are discussed in "Realising the Potential of Expert System Shells in Education", published in the proceedings of the of the *4th International Conference on AI and Education* (1989)

- The framework for explanation generation described in Chapter 5 appears in "EXPLORES: an EXPLanation ORiented Expert System shell for use in education", published in the proceedings of the *5th Alvey Explanation Workshop* (1990)

Abstract

Recent years have seen the increasing use of expert systems and expert system shells in industry and commerce. However, there has been little use of the technology in schools and colleges, partly due to dissatisfaction with the available software. This thesis describes the design, implementation and evaluation of an expert system shell for use in education.

An initial evaluation was carried out to establish some criteria for the design of the new shell. This comprised an evaluation of expert system shells by experienced teachers; an evaluation of several expert system shells by the author; and a case-study visit to a school Computer Studies department. The teachers were also asked for their views on the possible uses of shells in the classroom.

An expert system shell was designed and implemented as a result of this evaluation. This has an environment for building knowledge bases; an environment for consulting knowledge bases; a knowledge representation language allowing separation of domain and problem solving knowledge; and three supplementary tools. The most important of these is an explanation tool, which allows exploration of the domain knowledge in a knowledge base. In response to posed questions, the tool can produce domain-based explanations using a domain-independent generation technique, and the topic of each explanation can be explored further through relevant follow-up questions. This tool provides the potential for users to learn about the domain of the knowledge base being explored.

An evaluation of this shell by teachers with previous experience of expert system shells suggests that it has the potential for use throughout the school curriculum as an educational medium.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1983 the Education and Human Development Committee (shortly to be renamed the Economic and Social Research Council) identified the broad area of Information Technology in Education as a topic which warranted expanded support and accelerated development. This eventually led to the establishment of the Council's Information Technology in Education (ESRC-ITE) programme, run by a co-ordinator who was responsible for the development of the initiative throughout the country. The main aims of the programme were [Lewis 86]:

- the review, evaluation and dissemination of the recent and current activity in the field of Information Technology in Education
- the identification of the needs of education in relation to Information Technology
- the stimulation of relevant research and the formulation of research guidelines
- the establishment and maintenance of a database of relevant work, and undertaking arrangements for coordinating and networking of those active in the field, including cognitive scientists, educational researchers, practitioners and policy makers.

In particular, the Council

> ...recognised the importance of involving practitioners and policy makers in the development of its programme of substantive research and research-related activities, and the necessity of ensuring close collaboration with commercial organisations such as publishers, software houses and hardware manufacturers.

The research project described in this thesis emerged from that programme, and we will now describe the background to it.

For several years the development and use of expert systems in academic and commercial environments has been widespread and increasing. In 1986 a directory of expert systems was published [CRI 86] with the purpose of providing

> ...an abstract of every report ...about an expert computer system that is intended to perform an expert task in some application area that is ...of real scientific, commercial or industrial importance.

It listed several hundred systems, in a wide variety of domains, such as medicine, architecture, investment and weather-forecasting. Given the inevitable time lag between compiling the directory and publishing it, it was undoubtedly out of date by the time it appeared. In addition, as it was restricted by practical necessity to those systems about which there were published details, there were almost certainly many systems which were not included. If we also consider the number of systems developed since then, this directory must now represent only a fraction of the total number of expert systems in use.

In commercial or industrial environments, these systems are used for many tasks, such as design, diagnosis, planning and monitoring. They can be developed within the company itself, or, as is more often the case, they are developed for the company by a research laboratory or software house. The most common use of expert systems in commercial settings is **consultation**, where the user interacts with the system by providing information about the current situation, and in turn receives a decision from it about the problem in hand, for example a diagnosis about what is wrong or advice about how to proceed. As an academic subject, research on expert systems covers many sub-areas such

as **knowledge acquisition**, which looks at techniques for acquiring the expert knowledge needed by the system; **knowledge representation** and **inference**, which respectively look at the ways in which that knowledge can be represented in the system and at strategies for reasoning with it during a consultation; **uncertainty handling**, which looks at techniques for dealing with situations such as those where the conclusions which can be drawn from the available data are not definite, or where the available data are incomplete; and **explanation generation**, which looks at ways to explain aspects of the system to the user, such as the inferences made and the reasoning steps taken during a consultation (some systems can also **justify** their conclusions, in that they can not only show which inferences were made, but can also say why).

At its simplest, an expert system can be said to have two main components: a **knowledge base** and an **inference engine**. The knowledge base contains a representation of the expert knowledge for the domain to which the system is dedicated. This will have been acquired from one or more experts in the field through either automated knowledge acquisition techniques (interaction between the expert(s) and computerised knowledge acquisition software); manual knowledge acquisition techniques (interaction between the expert(s) and the knowledge engineer(s) responsible for developing the knowledge base); or a combination of both.

The inference engine contains the information needed to run the consultation, which enables it to establish what knowledge is needed from the user, and to decide what inferences can be made from the **static** knowledge in its knowledge base and the **dynamic** knowledge acquired during the consultation (from the user or from previous inferences). The inference engine and the knowledge base will normally be wrapped up in a **user-system interface**, which provides the facilities for the user and the system to interact.

Along with this increase in the use of expert systems there has been a parallel rise in the development and use of expert system shells. An expert system shell can be regarded as an expert system without a knowledge base: that is, it provides a framework consisting of an inference engine and an interface into

which a knowledge base in a range of domains can be slotted and consulted. In addition, a shell will also have a **knowledge representation language**, a high-level language in which domain knowledge can be represented, to create a knowledge base. The attractions of such a framework are obvious: it is senseless to build each system from the beginning when essentially the only difference between many systems is in the knowledge residing in the knowledge base.

The early expert system shells were developed in academic environments, and include shells such as EMYCIN, HEARSAY-III and EXPERT. EMYCIN (Essential or Empty MYCIN) [vanMelle 79, vanMelle *et al* 84, Bennett & Englemore 84] was a rule-based shell which was a generalisation of the basic framework of MYCIN [Buchanan & Shortliffe 84], an expert system for diagnosing infectious diseases: simplistically, EMYCIN was MYCIN without its knowledge base. As a test of its domain-independence, EMYCIN was used to build successful expert systems in several different domains, such as medicine (PUFF, CLOT and HEADMED) and engineering (SACON). HEARSAY-III [Belzer *et al* 80] was a domain-independent version of HEARSAY-II, a blackboard system for speech-understanding: it was used to develop several applications, such as a system for producing natural language descriptions of expert system data structures. EXPERT [Weiss & Kulikowski 79] was another rule-based shell, used to develop systems in domains such as rheumatology, opthalmology and endocrinology. These shells and others like them formed the basis for the many commercial shells developed shortly afterwards (for example, many of the features of HEARSAY-III were used in the ART expert system shell [Laurent *et al* 86]).

When this research project began in 1986, the main use of expert system shells was in commercial environments: they were heavily marketed as providing a means by which companies could develop their own expert systems, without the need to liaise with computer professionals as before (this was often both time-consuming and expensive). However, developing such a system was not always as simple as the vendors of these shells made it appear, and consequently shells were also heavily used in academic departments and research laboratories to develop dedicated expert systems commissioned by companies. Regardless,

their provision of a ready-made environment still represented a substantial saving in time and resources, and in fact both types of use are still very much in evidence today.

A user who is consulting a knowledge base has access to a body of knowledge about a particular domain. The developer of a knowledge base must acquire, organise, and represent knowledge about a domain. It would therefore seem that both of these tasks provide an opportunity for domain learning, and consequently that expert system shells could be used as an educational medium. An example of such use would be in schools, where pupils could build and consult knowledge bases related to the subjects which they are being taught. However, in 1986 few, if any, expert system shells were being used in this way.

1.1 The Research Project

This research project began in 1986, as part of the ESRC-ITE programme. In keeping with the aims of that programme, it was a collaborative project between two university departments, two colleges, a university-based unit and a commercial company. The main aims of the project were:

- to explore the potential of expert systems in education, the characteristics of existing expert system shells, and the value of these tools as perceived by experienced teachers

- to define, and where necessary develop, appropriate expert tools (software and documentation) for use in education and training

In 1986 educational establishments such as schools and colleges had some computer hardware and software at their disposal. However, unlike industrial and commercial environments almost none of them had begun to use expert system shells in any major way, in spite of the learning opportunities which such software would seem to provide. There were several possible reasons for this. In Great

Britain as a whole the introduction of computers into education was carried out in a haphazard and unprincipled way, with, for example, little by way of resources and no infrastructure for training and support. (An analysis of the social and political reasons for this and a wider view of the full extent of the problem can be found in [Conlon & Cope 89]). Thus, although many schools had computer hardware of some kind, few had any staff with the time or the background to investigate appropriate software and to introduce it into the school curriculum. This problem was compounded by the fact that much of the software around was second-rate, developed by computer programmers with little or no teaching experience. Thus, even those teachers who had received suitable training or who were granted time during school hours in which to train themselves, were often disappointed in the software they were given to use.

Although there were many expert system shells on the market at that time, very few of those available were suitable for use on the kind of hardware commonly found in schools, which in 1986 was mainly the BBC micro. In addition, almost all of the available shells had been designed with an eye very firmly on business and commercial use: none were designed specifically for educational use. Thus, even if there had been sufficient resources to pay for teacher training and appropriate hardware, it does not follow that expert system shells would automatically have been taken up and used in classrooms. There still remained the problem of the suitability of existing shells for educational use. This begged several questions. If an expert system shell was to be used in the classroom, then it was necessary to look at how it could be used. Who could it be used by? For what tasks? How could pupils benefit from using it? Once questions such as these had been answered, then existing shells could be examined in the light of those answers, and this would provide some clue as to what, if anything, was wrong with them to prevent their use in schools and colleges. Consequently, it would also provide a picture of the characteristics of an expert system shell which would be suitable for educational use. Thus, this project was set up with a view to providing a piece of educational software with a more sound educational basis than had previously been the case.

The initial intention was to have two PhD students working collaboratively on the project, one based in the Department of Education and the other in the Department of Artificial Intelligence, both at the University of Edinburgh. Circumstances were such that in the end there was no PhD student based in the Department of Education: however, some of the initial evaluation was done by a temporary Research Associate who was based there for the first year of the project.

The industrial collaborators on the project were Research Machines Ltd, who provided the hardware (an RML Nimbus) on which the new expert system shell would be developed, and appropriate software in the form of some existing shells, which would provide a starting point for the project.

The experienced teachers who would participate in the initial evaluation of existing software and in the eventual evaluation of the new software were drawn from three institutions running fellowships in Information Technology (or related courses such as Educational Computing at Advanced Diploma or Master's level). These were Moray House College (Edinburgh), St. Martin's College (Lancaster) and the Computer Based Learning Unit in the Department of Education at the University of Leeds, and each institution undertook to introduce the topic of expert systems and expert system shells to the teachers on those courses. This would then allow the researchers to get from those teachers their opinions of the shells they had used and their views on the educational use of shells in general. In addition, it was hoped that two or three of the teachers might emerge as further participants in the project, by undertaking to do their end-of course project on some aspect of expert systems or expert system shells. These would then form the basis of more detailed case studies, where the research student from the Department of Education would monitor and interpret the attitudes, problems and perceptions of the teachers during their projects.

We will now describe the relevant courses offered at each of the three institutions, and the alterations made to them to accommodate the institution's participation in this research project.

1.2 The Participating Institutions

There were three institutions which agreed to participate in the project by introducing the topic of expert system shells into their Diploma courses in IT or Educational Computing. For this, they had available five expert system shells:

Xi: a small rule-based shell with three modes of use, which supports both forward- and backward-chaining [Exp85, anon 85]

apes: a shell based on a subset of micro-PROLOG, with an English-like front end [Hammond & Sergot 85, Lehner & Barth 85]

ESP ADVISOR: a shell said to be suitable for 'text-animation', that is, the presentation of relevant sections from a large body of text [ESP85]

ESP Frame-Engine: a shell with a knowledge representation language based on frames [ESP86]

I-1: a shell based on an induction algorithm, which induces decision rules about a domain from a given set of examples [PAL86]

We will now describe the structure of the courses offered at each of the three institutions, and how these were altered to include the topic of expert system shells. [1]

1.2.1 Moray House College

The course on offer at Moray House was a Diploma in Professional Studies in Education (DPSE), in the field of Educational Computing. The course was open

[1] The reader should note that the descriptions we give of the three courses reflect them as they were in 1986: many aspects of those courses may now have changed.

to registered teachers in primary, secondary, special and further education, and to staff in Colleges of Education. Previous computing experience was not essential, and the course had between twelve and twenty people every year. It was run over two years, with teachers attending for one-day per week over the six terms. In the first four terms, they took four core units in the areas of 'Programming and Problem Solving', 'Computers and Society', 'Computers and Computing', and 'Educational Analysis and Design'. In the final two terms, the teachers took one of several optional units, such as 'Modelling and Simulation', 'Computer Aided Learning', and 'Data Management Systems'. In addition, they had to carry out a project based on that option. The project involved the development of a computer based unit for use in school or college, either for teaching, assessment, or administration. This unit was then tested *in situ* and was submitted as a documented package along with a report on its use. The teachers were assessed on each of the four core units, using a combination of written assignments, written examinations, and practical work, and on their project. Award of the Diploma was on a pass/fail basis.

The Moray House contribution to this research project involved the setting up of a 25 hour optional module called 'An introduction to expert systems'. The course offered a general introduction to the theory and practice of expert systems technology. Its objectives were that at the end of the module participants would be able to

- describe the general characteristic features of expert systems
- explain some ways by which knowledge may be usefully represented, including semantic nets, frames, production rules and logic
- explain some ways by which inference may be arranged, including forwards and backwards reasoning
- state the distinction between rule-based and rule-induction systems
- identify the main features of some existing expert system shells, including APES, ESP ADVISOR, ESP Frame-Engine and Xi

- design simple expert systems and implement them using either Prolog or one or more of the shells mentioned above

- identify the main limitations of existing expert systems technology.

The module consisted of ten sessions, each two and a half hours long, with a combination of lectures, demonstrations, experimental work and discussion, and during it the participants would either use or would see demonstrated all five of the available shells.

1.2.2 St. Martin's College

The Institute for Educational Computing at St. Martin's College ran a Diploma of Advanced Studies in Education (DASE) in the area of Educational Computing. This was a one-year, full-time in-service course: the entry requirements for it were a minimum of two years' teaching experience with primary or secondary pupils, and some prior experience of using computers in schools. Each year there were around 8–12 people on the course, and in fact those teachers who did have previous computing experience were usually in a minority. The course had four main topics: 'Curriculum trends and priorities', 'Micro-computers as a learning resource', 'Computer awareness and computer science', and 'Research methods'. The course assessment had three components, each of which counted for one-third of the final mark: a 10,000 word dissertation; a 3-hour, written final exam; and course work. The latter had three equally weighted elements: a 2,500 word essay on curriculum strategies; computer studies assignments (program development and an essay); and the development of original learning materials for use within schools, including or involving software.

To accommodate the requirements of this project, it was proposed that teachers on the course be introduced to expert system shells within its 'Micro-computers as a learning resource' component, which was allocated approximately one day a week for a year of contact time. The organiser of this component decided to teach a one and a half day introduction to expert system shells, from

both a theoretical perspective and a practical, 'hands-on' perspective. Given the limited time available, he felt that participants should be given only one shell to use, and of those available Xi was chosen by him as being the simplest for novice users. This was important because it meant that in the time allocated students could not only consult existing knowledge bases but could begin building a small one of their own: a simple shell would allow them to get further with that task.

The aim of this short introduction, which took place in December 1986, was that students would acquire an overview of expert system shells, with regard to their current power and their future potential in school work. Its objectives were to provide teachers with a basic familiarity with Xi up to the ability to create their own simple knowledge base, and to get from them their views on the potential for using expert system shells within schools over the next two or three years. The author attended this short course, helping out where necessary.

The course began with a background talk on AI, expert systems, and forward and backward chaining. This was followed by a short demonstration of Xi. The students then had a practical session using Xi in TUTORIAL mode, a self-paced introduction to the shell which is included with it. This was followed by another demonstration and some practical work using Xi, firstly in DIALOG mode and secondly in TOOLKIT mode. DIALOG mode is a way of interrogating an Xi knowledge base where the user is supplied with information about what is happening during the consultation, such as the inferences made. TOOLKIT mode incorporates the features of DIALOG mode, with additional facilities for knowledge base editing. For the practical work, the teachers were divided into groups of two, and in the final session each group demonstrated the knowledge base that it had built, after which the class as a whole discussed the possibility of using Xi and similar shells in schools.

Students on the Diploma course the following year received a similar, one day introduction. This was developed and run by the author, and took place in March 1988. Although Xi was an easy shell to use in the first instance, our experience from the previous year's course had led us to believe that it might be almost too trivial. In addition, we wanted feedback from the teachers on as

many different shells as possible. For these reasons, we had decided to use ESP ADVISOR during a similar introduction held at the CBLU in Leeds in January 1987 (see below). Although this shell had its own problems, it was judged to be the best of the available shells for this kind of use, and the success of the Leeds course prompted us to repeat it at St. Martin's. As before, the aims of this short introduction were to give a general overview of the characteristics of expert systems and expert system shells, in terms of their components, uses and classroom potential. Its objectives were to provide students with a working knowledge of ESP ADVISOR for both building and consulting a knowledge base. The day began with an introduction to AI, expert systems and expert system shells. Following a short demonstration of ESP ADVISOR the students were given more detailed information, such as an overview of its knowledge representation language; the structure of an ADVISOR knowledge base; and the control strategy used during a consultation. This led to two practical sessions: one where students could consult pre-built ADVISOR knowledge bases, and one where they could attempt to build their own. The day ended with a discussion on ESP ADVISOR in particular and on expert system shells in general with regard to their possible uses in education.

1.2.3 Computer Based Learning Unit

The Computer Based Learning Unit, situated within Leeds University, offered a Diploma in Information Technology and Education. The duration of the course was either one year of full-time study; an equivalent period of part-time study; or an equivalent period of full-time and part-time study. The entry requirements were that candidates must be qualified teachers with not less than three years teaching experience. The course was divided into modules, of which students had to take the six required modules and two of the four optional modules. The six required modules were 'Information Technology: context and implications', 'Hardware and Software Systems', 'Computer-based Learning: theory and practice', 'Problem-solving through structured programming I' (and II) and 'Educational software construction'. In addition, the four optional modules offered

were 'Artificial Intelligence and education', 'Software tools for the classroom', and 'Micro-technology I' (and II). The teachers were assessed on five pieces of prescribed work based on the required and optional courses, and a project or special study of approximately 12,000 words (or an equivalent piece of work).

The timing of the start of this research project was such that it was too late for any of the students on the current course to be introduced to expert system shells with a view to subsequently using them in their own course project (the topics of those projects had already been chosen). However, it was proposed that an introduction to expert system shells be incorporated into one of the modules in the latter half of the term beginning in January 1987. This would provide an opportunity to evaluate the activities of course members and their reactions to the shell as they were using it. In addition, there was also a teacher associate who was on secondment in the School of Education at Leeds University. He was looking at the topic of Maths Education, and was keen to do some work involving expert system shells.

The topic of expert systems and expert system shells was introduced to the teachers at the CBLU during a one day course run by the author in June 1987. This was essentially the same as that run at St. Martin's college in March 1988, described in detail above.

The next Diploma course in the CBLU was scheduled to begin in September 1987, and it was suggested that an introduction to expert system shells could be incorporated into one of the modules offered during the second term, which began in January 1988. This would then offer students the possibility of going on to do further work involving shells during their course project. In the event, this did not happen: the course organisers were unwilling to teach the topic themselves, and as we were by that point reduced to one researcher, time constraints made it difficult to run another one-day course. In any case, projects which might have resulted from such a course would have been carried out too late to contribute to the initial evaluation.

1.2.4 Summary

In this section we have described the three institutions who agreed to take part in this research project, in terms of the structure of the courses run there, and the alterations made to those courses to accommodate the requirements of the project. These were that teachers on the courses be introduced to the topic of expert system shells to the extent that they could then participate in our initial evaluation. During this evaluation, the teachers would be asked to both evaluate the shell(s) they had used from the perspective of their potential use in education, and to discuss their views on the educational use of expert system shells in general. We describe this evaluation in Chapter 3. For the rest of this chapter, we provide an outline of the thesis, in terms of the contributions which it makes to the knowledge of the field, and of a chapter-by-chapter account of its overall structure.

1.3 An Outline of the Thesis

1.3.1 The contribution of the thesis

This thesis contributes to two main areas. The first complies with the stated aims of this research project: to develop an expert system shell for educational use based on an analysis of existing shells by experienced teachers. In the thesis we consider the implications of such use, in terms of *who* in a classroom will use an expert system shell; of *how* they will use it; and, most importantly, of *why* they should use it. In addition, we describe the EXPLORES (EXPLanation ORiented Expert System) shell, which resulted from those considerations. Thus, EXPLORES is an expert system shell designed for use as an educational tool throughout the curriculum, with the potential for helping users to learn about any knowledge domain which can be structured to fit the shell's knowledge representation language (KRL). This is a new direction in the educational use of expert system shells, moving away from the narrow use of shells in the

Computing Studies curriculum and towards their wider use as a tool which can help promote learning in a variety of subject areas.

The feature of the EXPLORES shell which most contributes to its use as a learning tool throughout the curriculum is its explanation tool. Thus, the second contribution of this thesis is in the area of the effective generation of explanations in expert system shells. Several techniques have been developed for the generation of domain-based and system-based explanations in expert systems. Many of these techniques are domain-dependent, and as such are unsuitable for use within expert system shells, which are by definition domain-independent. However, even those expert system techniques which are domain-independent have not been used, and until now the explanation facilities in expert system shells have consisted of little more than the standard system-based explanation of a rule-trace presented in response to *how* and *why* questions. In this thesis we describe a domain-independent technique for the generation of domain-based explanations in a micro-based expert system shell. This technique allows the shell to provide such explanations in response to a variety of questions within a range of explanation categories, with additional information available in the form of follow-up questions.

1.3.2 The structure of the thesis

The thesis is divided into eight chapters as follows:

Chapter 2 We describe the knowledge representation formalisms commonly used in expert system shells, with regard to their structure, their relative advantages and disadvantages, and the kind of knowledge which they can be used to represent. We follow this with a brief description of the inferencing strategies used to reason with the knowledge in a knowledge base, and of some techniques used to represent and handle uncertainty. We then discuss some currently available expert system shells with regard to the knowledge representation formalisms which they use and the facilities which they offer for the development and consultation of knowledge bases.

We finish by compiling a list of all of the features currently offered by expert system shells, and comparing these with the features offered by the EXPLORES shell.

Chapter 3 We describe the initial evaluation, outlining the methodology used and the results obtained. We then discuss these results, touching on related work in this area, and finish by specifying a set of criteria for the design of the new shell, based on the findings of the evaluation.

Chapter 4 We describe the KRL developed for the EXPLORES shell, and illustrate this using examples from a simple knowledge base for plant classification. Using a subset of that knowledge base, we then describe the strategy used by the shell to control inferencing during a consultation. Following this, we discuss the limitations of the EXPLORES KRL, with regard to the kind of knowledge which can best be represented in it.

Chapter 5 We begin by analysing the explanation facilities of a number of expert systems, within a framework which examines the techniques used and the factors which can affect the type, structure and content of the generated explanation. We then move on to discuss the limited explanation facilities of most expert system shells, in terms of the framework constructed previously. Based on these analyses, we discuss the requirements of a hypothetical explanation facility as part of an expert system shell intended for educational use, and go on to describe the explanation tool developed as part of the EXPLORES shell. We end this chapter with some conclusions regarding the function of explanation in an educational shell.

Chapter 6 We look in greater detail at the the EXPLORES shell, beginning with a description of its overall structure. We follow this by taking each part of the shell in turn and outlining its main features; describing how it relates to other parts of the shell; and discussing any problems which were encountered during its implementation.

Chapter 7 We describe the evaluation of the EXPLORES shell by teachers at one of the participating institutions, with regard to the methodology used, the results obtained, and the conclusions which can be drawn from it.

Chapter 8 The final chapter of this thesis discusses the conclusions which can be drawn about the utility of the EXPLORES shell as an educational tool, with regard to the criteria upon which it was based and the extent to which it satisfies these. We end this chapter by describing alterations which could be made to the shell (in terms of those which could be made to its existing features and of any new features which could be added); by listing additional work which would have to be done with regard to the development of user-support tools if the shell was to be used in schools; and by discussing further research which could be carried out in the area of domain-based explanation generation in educational expert system shells.

Chapter 2

Knowledge Representation and Expert System Shells

In Chapter 1 we described how an expert system shell can be thought of as a framework into which a knowledge base can be slotted and consulted. In theory, the knowledge in this knowledge base can be from any domain: however, it must be presented in a form which can be understood by the shell. Thus, an expert system shell must provide a language into which the knowledge in a domain can be translated, in order to form an acceptable knowledge base. This language is known as a shell's **knowledge representation language** (or KRL). In theory, this could be a programming language such as Lisp or Prolog, and in fact that is all that some expert system shells do provide (for example EESS [Scherz *et al* 88b, Scherz *et al* 88a]). However, in practice, if the shell is to be used by non-programmers, then a higher-level KRL should be provided, usually containing a set syntax of reserved words and system primitives, and based on one or more **knowledge representation formalisms**, such as production rules or frames.

In this chapter we describe some of the knowledge representation formalisms upon which such a KRL might be based, in terms of their structure, their relative advantages and disadvantages, and the kind of knowledge which they can be used to represent. Following a brief description of the inferencing strategies used by expert systems to reason with the knowledge in a knowledge base, we look at several techniques for dealing with the representation and handling of

uncertainty. We then discuss some currently available expert system shells in terms of the knowledge representation formalisms which they use and the facilities which they offer, and follow this with a comparison between the features available in existing shells and those available in the EXPLORES shell. We finish with a summary and some conclusions about the design and implementation of a new expert system shell, with regard to the features that it should provide for the development and consultation of knowledge bases if it is to be used in an educational environment.

2.1 Knowledge Representation Formalisms

Expert systems come as a complete package dedicated to one specific domain, and as such they can represent the knowledge that they have in the language used to program the rest of the system, for example Prolog or Lisp. However, an expert system shell, if it is to be used by non-programmers, should provide a KRL, that is, a structured language in which the knowledge for a range of domains can be represented and subsequently used by the system. This KRL will consist of reserved words, knowledge structures and system primitives, and will be based on one or more knowledge representation formalism such as frames or production rules. A representation has been defined by Winston [Winston 84] as 'a set of syntactic and semantic conventions that make it possible to describe things': Jackson [Jackson 90] defines it further, saying that:

> The syntax of a representation specifies a set of rules for combining symbols and arrangements of symbols to form expressions in the representation language. The semantics of a representation specify how expressions so constructed should be interpreted, i.e. how meaning can be derived from form.

There has been an enormous amount of research carried out on the subject of knowledge representation, much of which has been reviewed in many other publications (for example [Brachman & Levesque 85]). For this reason, a complete review is inappropriate here, and therefore we will restrict our review to

a description of the knowledge representation formalisms commonly used as the basis for the KRLs in expert systems and expert system shells, namely production rules and frames.

2.1.1 Production rules

One of the most commonly found knowledge representation formalisms (particularly in small expert system shells) is production rules (also known as condition-action rules and if-then rules). These generally have a syntax of the form:

if P1 & P2 ... Pn **then** C1 & C2 ... Cn

where P1 to Pn are a series of **premises** (such as *sky_colour* = *blue* or *size* < *10*), which, if true, will lead to the **conclusions** C1 to Cn being true (such as *weather* = *sunny* or *type* = *small*). Note that the meaning of the symbol = often differs in a premise and a conclusion: in the former it means 'if the variable X *has* the value Y', while in the latter it means 'then the variable X *takes* the value Y'.

There are many situations where a conclusion cannot be drawn with absolute certainty, due to problems such as missing data or the fact that even with a complete set of data the conclusion is at best only a strong possibility. For this reason, production rules often have a measure of confidence in the rule's conclusion attached to them, such as a probability rating [Duda 80] or a certainty factor [Buchanan & Shortliffe 84]. The representation and handling of uncertainty is discussed in more detail below (Section 2.3).

For a small, well-defined body of knowledge, production rules provide a quick and relatively simple method of representation. However, outwith these restrictions, such as domains where the knowledge is less well structured, their limits can be quickly reached, and it may prove difficult to structure the knowledge to fit the representation (this and other problems are discussed in [Jackson 90]). Another problem often discussed in relation to the use of production rules is

the fact that much of the domain knowledge is 'compiled' to become part of a rule, and it is this compiled knowledge which is represented: knowledge about the domain is not represented explicitly, and consequently is not available to the system [Swartout 81]. This also means that there is no separation of domain knowledge (declarative knowledge) and problem-solving knowledge (procedural knowledge), and thus there is no way of distinguishing between the two.

This problem is in fact a wider one than simply the use of production rules as a representation formalism: it involves a distinction between representations of the deep knowledge and the surface knowledge of a domain [Steels 84]. A deep representation [Steels 90] 'makes explicit the models of the domain and the inference calculus that operates over these models', while a surface representation

> ...contains selected portions of the deep knowledge, in particular, those portions that are relevant for the class of problems that is likely to be encountered. It also contains additional heuristics and optimizations...

This distinction is separate from any consideration of the knowledge representation formalism: both deep and surface knowledge can be represented using production rules. However, many first-generation expert systems have only a surface representation of their domain, represented using a production rule formalism (although later systems have begun to investigate the use of deeper representations). The latter is particularly relevant if the knowledge base of the system is to be used for some other task such as tutoring or explanation generation: these points are discussed in detail in Chapter 5.

2.1.2 Structured objects

Structured objects is the generic name for knowledge representation formalisms [Jackson 90]

> ...whose fundamental building building blocks are analogical to the nodes and arcs of graph theory or the slots and fillers of record structures.

They provide a way of grouping information in a more or less 'natural' way, and Jackson describes several such formalisms, such as graphs, trees, semantic and associative nets, and frames. Of these, frames are the most commonly used in expert systems and expert system shells, and as such are the only formalism of this type which we shall discuss here. Note that the term frame is often used interchangeably with other terms such as *schema* and *object*.

Frames

Frames were first described by Minsky as 'data structures for representing stereotyped situations' [Minsky 75], and they can be used [Jackson 90]

> ... to reason about classes of objects by using prototypical representations of knowledge which hold good for the majority of cases ...

They are essentially a way of describing objects using 'slots' and 'fillers' to hold the name and properties of each object (these properties can themselves be frames, with their own attributes). Other slots in a frame can contain information about how the frame can be used, and frames can be linked to one another using subclass/superclass relationships (such as 'is a kind of'). With these links, properties can be inherited from frames higher up in the hierarchy (which can represent classes of objects, such as 'plants') by frames lower down, which can represent specific instances of that class (such as 'old man cactus').

There have been several knowledge representation languages produced which are based on frames, in that they are based on the principal of [Jackson 90]

> ... the packaging of both data and procedures into structures related by some form of inheritance mechanism.

However, in these languages, procedures can be inherited as well as data, and objects can communicate by passing messages to one another. These are collectively known as object-oriented languages, where early ones such as KRL [Bobrow & Winograd 77], can be thought of as 'a precursor of some of today's software tools' [Jackson 90]. Later systems include FLAVORS (the basis of

SMALLTALK [Goldberg 81, Goldberg & Robson 83] and other systems), LOOPS and CLOS [Jackson 90].

As a representation formalism, frames are relatively easy to use, and provide a good way of exploiting the structure inherent in many knowledge domains. There are many domains which fit well into this type of representation, and many others which can be made to fit it. In addition, frames also provide a good way of handling exceptions and defaults in a domain. The main arguments against the use of frames seem to focus on semantic aspects of the representation. The first of these is concerned with what it really means to say that something 'is an example of' something else [Brachman 83]. Brachman lists several possible meanings for an ISA link between two objects, under two main types: generic/generic relations and generic/individual relations. Within the first category, there are interpretations of ISA such as a subset/superset relationship ('a spiny plant is a plant'); conceptual containment ('a square is a quadrilateral'); and role value restriction ('the trunk of an elephant is a cylinder 1.3 metres long'). In the second category, Brachman includes interpretations such as set membership ('Nellie is an elephant', that is, 'Nellie is a member of (the set of) elephants'); and abstraction ('an eagle is an endangered species'). Such a wide range of interpretations can lead to lack of clarity and logic concerning any statement(s) containing an ISA construct. For example, if we say that 'resinifera is a spiny euphorbia', and that 'Mary collects spiny euphorbia', then it logically follows that Mary collects resinifera, which may not necessarily be the case.

A related problem concerns the way in which frame-based representations often deal with objects which are exceptional within their class, such as birds which cannot fly [Brachman 85]. Many frame systems allow inherited properties to be cancelled or overwritten at a point further down the hierarchy. For example, the frame representing the general class 'birds' may contain the property 'flies': this could then be cancelled further down the hierarchy within a frame representing an instance of the class 'birds', such as 'ostrich'. Brachman argued that such unrestrained overwriting makes it impossible to represent 'definitional conditions' (such as 'all triangles have three sides') or 'contingent universal state-

ments' (such as 'all the cars in this showroom are Fords'). The problem is that many systems make no distinction between 'essential' and 'accidental' properties, where essential properties are those which are necessary for an individual to be considered as an instance of a class, and accidental properties are those which all instances of a particular concept happen to possess.

Although this problem is a real one, many recent systems have circumvented it, for example by differentiating between 'class' and 'instance' variables (LOOPS) or between 'shared' and 'local' slots (CLOS). Other KRLs have restrictions such that inherited properties cannot be cancelled or overridden. We return to this point in Chapter 4, when we discuss the representation of *things*, the frame-like structures present in the EXPLORES KRL.

2.1.3 Combining knowledge representation formalisms

Some developers of expert systems have found it valuable to use a combination of formalisms. An example of such a system is CENTAUR [Aikins 83], an expert system in the domain of pulmanory function. CENTAUR uses frames to represent prototypical objects in the domain, such as disease patterns. These are arranged in a hierarchy, where the frame at the top of the hierarchy is a consultation prototype which controls the way the consultation is run. Rules are associated with the slots of these frames, and are classified according to their function in the system (such as inference rules or summary rules). Other slots in a frame provide the context in which these rules are applied, and can themselves contain prototypes.

Such a representation clearly separates knowledge about the domain (situational knowledge) from knowledge about how to run the consultation (strategic knowledge) [Aikins 83]. There are many advantages to such a representation, with regard to the efficiency of the system (in that the context of a rule's application is always explicit) and to the clarity of the representation. The purpose of the knowledge in the system and the context in which it should be used are both made explicit. In addition, access to the domain knowledge in such a sys-

tem enables the generation of justified explanations, that is, explanations which make clear not only *what* the system is doing but *why* it is doing it. We return to this point in Chapter 5.

2.2 Inference

The main function of an expert system or expert system shell is consultation, where an inference engine reaches conclusions by reasoning with the **static** knowledge that it has in the knowledge base and any **dynamic** knowledge which it has acquired during the consultation. The inferencing carried out during a consultation must be done in a controlled and structured way: many rule-based systems use either **forward** or **backward chaining**, or a combination of both. and systems using a representation other than production rules usually have an inferencing mechanism based on these. Another mechanism for controlling inference is the use of a **conflict resolution strategy**. The details of all of these mechanisms are covered in many publications (for example, [Jackson 90] or any basic expert systems textbook). Here, we will limit ourselves to a brief description of each.

2.2.1 Backward chaining

Inferencing based on backward chaining involves trying to prove a stated top level goal using the available rules and facts. For example, if the stated goal is X, the system will find all of the rules in the knowledge base which conclude about X, for example:

rule1: **IF** *A* **THEN** *X*
rule4: **IF** *B* **THEN** *X* **AND** *Y*
rule8: **IF** *C* **AND** *D* **THEN** *X*

It will then find which of A, B, C and D can be proved, again by finding all of the rules which conclude values for these. This chaining continues until all of the

relevant premises have been proved, at which time a value for X can be found by inferencing back through the chain. In the above example, a value for Y will also be found if the premise B can be proved: although this has occurred as a side-effect of the inference process, any reasonable system will store that value and will retrieve it if required at a later point in the consultation, thus saving duplicate inferencing.

In summary, backward chaining can be thought of as being goal-driven, that is, it is a reasoning strategy which moves from goals to facts. For this reason, it is most commonly used in expert systems where a value is required for a known thing (the goal), and where the system will try to establish that goal by deducing facts using its static and dynamic knowledge.

2.2.2 Forward chaining

Inferencing based on forward chaining involves making inferences from an initial data set by firing all applicable rules. The system then makes further inferences from the information gathered by the initial firing, and this cycling continues until no more inferences can be made. For example, if a knowledge base contains rules of the form:

rule1: **IF** *A* **THEN** *B*
rule4: **IF** *B* **THEN** *C*
rule8: **IF** *C* **THEN** *D*

and the initial data given is that A is true, then *rule1* will fire, concluding that B is true. On the second round, B is true, and *rule4* will fire, concluding that C is true: on the third round, *rule8* will fire, concluding that D is true. At this point no further inferences can be made, and so D will be the result of the consultation. However, in many systems, all rules which can fire will fire on each round (for example, *rule1*, *rule4* and *rule8* would all fire on the third round of the above example). Thus, for a large rule set, forward-chaining can be very inefficient, and requires some kind of mechanism to regulate it.

An example of a simple mechanism for making forward chaining more efficient can be seen in the FLEX expert system shell, described in Section 2.4.5. This involves the setting of a 'once only' option for appropriate rules when the knowledge base is being developed, which means that the rule will only fire once during a consultation. An example of a more sophisticated mechanism would be the use of a Rete match algorithm [Forgy 82], which can be seen in the OPS production rule language. This algorithm is based on two problems commonly found in naive forward-chaining systems: **within-cycle iteration** and **between-cycle iteration**. On each round of rule-firing, the premise(s) of a rule are compared to the facts which have been deduced during the consultation and stored in working memory. Within-cycle interaction is where several production rules in a system share the same premise(s), but where each rule is compared in any one round, even if the first rule of the group did not fire. Between-cycle iteration is where the information gained on any one round of rule-firing is minimal, but where on the succeeding round all rules are compared again.

A Rete match algorithm reduces within-cycle interaction by compiling the premises of all of the rules in the knowledge base into a tree-structured sorting network. It then computes the set of rules which can fire by processing this network. Between-cycle iteration is controlled using a set of tokens which indicate which rule premises match which elements of working memory: this set is updated when working memory changes, that is, when new facts are deduced. Although this algorithm makes forward-chaining more efficient, it is itself computationally expensive, and so is unsuitable for smaller systems.

In summary, forward-chaining can be thought of as being data-driven, that is, it is a reasoning strategy which moves from facts to goals. For this reason, it is most commonly used in expert systems where it is not clear what the goal will be, but where initial data is available.

2.2.3 Combining inference mechanisms

Many larger systems combine forward and backward chaining in an attempt to limit inferencing and thus reduce processing time. An example of this is the use of **rule sets** in the Goldworks shell (described in Section 2.4.5) where the rules in a knowledge base are tagged as belonging to a particular set, and where the tag is something for which a value could be found by forward-chaining through the rules in the set. During a consultation, the system carries out normal inferencing using backward chaining until a value for the tag of a rule set is needed. At this point it switches to forward chaining within the tagged rule set to try to establish the value needed.

The efficiency of such a combination depends heavily on the algorithm used, and, more specifically, on a trade off between the respective costs of backward and forward chaining with regard to the number and complexity of the rules under consideration. For this reason, use of a combined technique may involve much fine-tuning of the system for it to be maximally efficient.

2.2.4 Conflict resolution

In many larger systems, a conflict resolution strategy is used during inferencing to determine how many and which rules fire at any one time. Three of the most common strategies used (often in combination) are ones based on:

refractoriness which imposes the condition that a rule may not fire more than once on the same data

recency which time-tags the dynamic knowledge acquired during a consultation and imposes the constraint that rules which use more recent knowledge are preferred over those using older knowledge

specificity which imposes the constraint that more specific rules (for example those with a greater number of premises) are preferred over more general rules

Systems which use one or more of these strategies can often require a great deal of fine-tuning. However, used effectively, they can greatly improve the efficiency of a system, and as such the cost of implementing them is usually deemed to be worth the benefits that it brings.

2.3 Representing Uncertainty

There are many domains where knowledge is is not certain enough for inferencing to be as straightforward as that described above. This can be the case if not all of the facts are known which would enable a conclusion to be reached with absolute certainty. In addition, there are situations where a complete set of facts will often only make it possible to deduce that something is likely to be the case, but not definitely. Thus, there are two main sources of uncertainty in expert systems: imperfect domain knowledge, where the actual domain theory may be incomplete or may contain errors; and imperfect case data, where the evidence upon which a conclusion must be based is imprecise (partial) or unreliable (approximate). Within the area of expert systems research, several methods have been developed for representing this uncertainty within a knowledge base. The three which we will touch on here are conditional probabilities, certainty factors and fuzzy logic: others include a theory of belief functions (the Dempster-Shafer theory) [Gordon & Shortliffe 84]; the use of exact inference [Stefik 78]; default reasoning and nonmonotonic logics [Reiter 80, McDermott & Doyle 80, Davis 80]; and endorsements [Cohen *et al* 85]. A full description, analysis and comparison of all of these methods is outwith the scope of this thesis, but is available in many other sources (for example ([Buchanan & Shortliffe 84, Rich 83, Jackson 90]). Each of these methods has its champions and its detractors, and overall there appears to be little consensus on what constitutes a sound, reliable method for dealing with uncertainty in expert systems. To quote Stefik [Stefik *et al* 82]

> There is little agreement among AI researchers on the utility of these ...for intelligent systems, or even on their advantages for reasoning with incomplete data.

We will now briefly describe three methods for representing uncertainty (conditional probabilities, certainty factors and fuzzy logic), outlining the main features of each and the advantages and disadvantages which its use provides.

2.3.1 Conditional probabilities

The use of conditional probabilities for reasoning with uncertainty is based on the likelihood of an event occurring given the existence of some other event, for example, the likelihood of a patient suffering from disease d if he complains of symptom s. The computation of such probabilities can be done in several ways: the most useful with regard to expert systems involves the application of Bayes' theorem. This states that the probability of d given s, written as $p(d|s)$, depends on a ratio of three other measures:

- $p(d)$, the prior probability of d, before the evidence s was available
- $p(s|d)$, the probability of s given d
- $p(s)$, the probability of s

These combine to give the formula:

$$p(d|s) = \frac{p(s|d)p(d)}{p(s)}$$

In an expert system, the probabilities associated with each piece of evidence are slotted into Bayes' theorem, which is then used to compute the probability of a given outcome. This can be seen in the expert system PROSPECTOR [Duda 80], which examines geological evidence to discover whether or not it is worthwhile digging for a particular mineral at a given location. The system contains the prior probability of finding each mineral and the probability that if a particular mineral is present then certain physical characteristics will be observed. Bayes' theorem is then applied to those values, to compute the probability that a mineral is present at a specified location.

There are several problems with the use of Bayes' theorem for reasoning with uncertainty:

- it is not always possible to have the necessary prior probabilities, either because the information is not available, or, in many cases, because it is simply unworkable to gather all of the empirical evidence on which such probabilities could be based for a large data set. Even if subjective probabilities are used (that is, probabilities based on the judgement of experts in the domain), it quickly becomes unrealistic for cases where a large number of hypotheses are possible.

- it is difficult to alter a knowledge base containing Bayesian probabilities: the probabilities of all possible outcomes must add up to one, and so any alterations to the knowledge base (for example new knowledge being added) involve re-calculation of all the other probabilities.

- only one possible outcome can be considered, for example it would not be possible for the system to conclude that a patient was suffering from more than one disease.

For these reasons, researchers have tried to find other methods of representing uncertainty.

2.3.2 Certainty factors

In an attempt to get away from the limitations and problems associated with the use of probability based methods for dealing with uncertainty, the developers of MYCIN developed an alternative method involving certainty factors [Buchanan & Shortliffe 84, Shortliffe & Buchanan 84]. These are associated with the conclusion of a rule and in MYCIN are numbers on a scale of -1 to 1, where 1 means that the conclusion is certain to be true if all of the premises of the rule are true, and -1 means that the conclusion is certain to be false if all of the

premises of the rule are true. Points in between suggest evidence for the conclusion (if positive) and against it (if negative): for example, a certainty factor of 0.7 would indicate that the evidence is strongly indicative of the conclusion, but is not certain. This value refers to the the certainty of the application of the rule: the certainty of any one conclusion is a factor of this value and of the overall certainty of the rule's premises. The overall certainty of the premises of a rule is determined by how these are linked: if they are linked conjunctively, then the minimum certainty factor associated with those premises is used; if they are linked disjunctively, then the maximum one is used. For example, a rule may exist of the form:

if	A
and	B
and	C
then	D (0.8)

that is, D is true with a certainty of 0.8 if A, B and C are true. If it has been established that A is true with certainty 1.0, B is true with certainty 0.8, and C is true with certainty 0.6, then using the 'minimum' rule, the certainty of the conjunction of these premises is 0.6, and the certainty of the conclusion is

$$0.6 \times 0.8 = 0.48$$

If the premises A, B and C had been joined by disjunction rather than by conjunction, then the 'maximum' rule would have been applied, making the certainty of their disjunction 1.0 and the certainty of the conclusion

$$1.0 \times 0.8 = 0.8$$

A full discussion of the mathematical basis underlying the use of certainty factors can be found in [Shortliffe & Buchanan 84].

The use of certainty factors to handle uncertainty can often produce results which are not correct from a mathematical point of view, in that they are not always in agreement with the measurements produced using probability theory. However, there is little doubt that the computation of certainty factors is

generally more feasible than the computation of the correct probabilities (see [Adams 84] for a comparison of the two methods). There is also some question as to the accuracy of the values computed and to their overall importance. The internal representation of the MYCIN certainty factor scale has 1000 intervals from 0 to 1000. A survey carried out by members of the MYCIN research team [Buchanan & Shortliffe 84] showed that this interval could be reduced to a 10 point interval scale with almost no change in the diagnosis and the therapy suggested (9 out of 10 trials had identical diagnosis and therapy suggested, and 1 out of the 10 had a different diagnosis but the same therapy suggested). Even with a four point scale (0, 250, 500, 750 and 1000) the results were 8 identical diagnoses and 2 with the same therapy. It was only when the scale was reduced to 2 and 3 points that any noticeable difference in diagnosis occurred, and even then the therapy suggested was consistent in 9 cases for a two point interval (a 3 point scale was the worst, with 5 in each category). However, the developers of MYCIN [Buchanan & Shortliffe 84] claim that its performance record is so good with regard to the accuracy of its diagnoses that this factor is irrelevant. This point is of general relevance when comparing any techniques for carrying out a particular expert system related task: overall, the main factor on which any system should be judged is the accuracy of its conclusions when compared to those made by human experts in the field, irrespective of the methods by which those conclusions were reached.

2.3.3 Fuzzy logic

In fuzzy logic, a statement is interpreted as having an imprecise denotation characterised by a fuzzy set, that is, a set of values, each with corresponding possibility values [Zadeh 76]. For example, the proposition 'X is a large number' could be represented by the fuzzy set:

$(X \in [0,10], 0.1)$
$(X \in [10,1000], 0.2)$
$(X > 1000, 0.7)$

interpreted as 'the possibility of X being less than 10 is 0.1; being between 10 and 1000 is 0.2; and being greater than 1000 is 0.7'. Fuzzy logic is concerned with inference rules for reasoning with fuzzy sets, and a fuzzy rule-based system can be thought of as one where there are successive rounds of rule firing, and where all rules fire on each round, but with varying strengths (ranging from 'not at all' to 'completely'). Whalen & Schott [Whalen & Schott 83] examine some of the important issues in the development of expert systems which make use of fuzzy logic. They analyse eight systems, some already implemented and some only at the proposal stage, with regard to 'the overall structure and application of the system; the nature and locus of uncertainty; and the representation of the logical implication (If-Then) operator'.

Schefe [Schefe 80] criticises fuzzy set theory and the fuzzy logic based upon it on the grounds that the foundations upon which they are based are still disputed. For example, he claims that the definitions of some set operations and logical connectives often appear to be arbitrary, that the relationship between 'fuzziness', 'probability' and 'possibility' is unclear, and that it does not preserve the tautologies of propositional logic. Schefe proposes a probabalistic base for fuzzy sets, rather than the possibilistic one proposed by Zadeh. This provides an alternative foundation for reasoning with 'graded agreement values', which Schefe claims corrects the deficiencies of Zadeh's theory.

In the next section, we will examine the use made in some commercially available expert system shells of the formalisms, inference strategies and uncertainty handling mechanisms which we have just described.

2.4 Expert System Shells

We will now discuss some commercially produced expert system shells, in terms of their KRLs and the other features which they have to offer. A recent book on expert system shells (or expert system tools as they are referred to there) separates the shells reviewed into five main categories: simple, rule-based tools;

inductive tools; mid-size rule-based tools; large rule-based tools and hybrid tools [Harmon *et al* 88]. As much of the information used below comes from that source, we will also use this classification. Note that many of the features mentioned below are generalised across the group of shells being discussed, and as such any one shell in that group may not have all of the features mentioned, or may in fact have more features than are described.

2.4.1 Small rule-based shells

Small, rule-based shells are those which run on a personal computer and use a KRL based on production rules. Although they are often capable of processing thousands of rules, they usually lack sophisticated editing facilities, and so impose a limit of only a few hundred rules in a single knowledge base. That being the case, some of them are still very powerful, providing facilities such as certainty factors; hooks to spreadsheet and database software; complex algebraic expressions; and access to more powerful programming languages such as LISP or C for behind-the-scenes subroutines. The inference engines of such shells generally work by backward chaining, although a few of the more powerful ones offer options for forcing forward chaining.

Some of the early shells in this category were often very difficult to use. For example interacting with APES [Hammond & Sergot 85], a tool for constructing logic-based expert systems with a KRL based on micro-PROLOG, could be very complex and unfriendly. Consulting an APES knowledge base often meant using queries such as:

find(_n: John has-children _n)

In addition, knowledge base developers had to be more or less proficient Prolog programmers before they could successfully build an APES knowledge base. Other shells such as ESP ADVISOR [ESP85] and Xi [Exp85, anon 85] were mainly regarded as trivial, although later versions of Xi, such as Xi Plus [Forsyth 87] and Xi User [Harmon *et al* 88] were more successful. Other more recent tools of

this type are by and large more powerful and more sophisticated: these include (from [Harmon *et al* 88]) Insight 2, Exsys, Personal Consultant Easy and VP Expert.

Also included in this category are three educational shells which we will discuss in the next chapter: Expert Builder [Exp89], which has both textual and diagrammatic methods for entering rules, ADEX Advisor (a backward-chaining shell) and EGSPERT Browser (a shell designed for browsing a knowledge base) (both [Briggs 87]).

2.4.2 Inductive shells

These are shells which use an induction algorithm to automatically generate domain rules from a large number of examples entered by the system developer. These examples (known as a training set) must be coded in terms of the important and relevant attributes which they possess, and often a weight (or confidence factor) can be attached to each one. From these, the shell will generate a decision tree on which it will base its conclusions during a consultation.

There are several problems with the use of induction. The first is that the inferences made, and consequently any conclusions drawn from them, may be based on incomplete or partial knowledge of the domain (for example the initial training set may be too small or be unrepresentative of the domain as a whole). Thus, inductive inference is not really knowledge based, but is instead a form of conjecture: the inferences generated by it cannot be proved to be true unless all possible domain examples can be given. For example, different induction algorithms will often give different results from the same examples, and in some cases it is possible to force a system to generate a very different rule or set of rules by increasing the size of the training set by only one example.

In addition, there are many instances where a decision tree of the type produced by inductive inference may not be the most efficient (or indeed correct) way of reaching a decision, in that it often fails to capture the obvious, for example in classification problems. Bloomfield illustrates this point well [Bloomfield 86]

Table 2-1: A sample training set

X	Y	Class	X	Y	Class
1	0	no	5	1	no
0	1	no	6	6	yes
1	1	yes	6	7	no
2	1	no	8	9	no
3	3	yes	10	10	yes

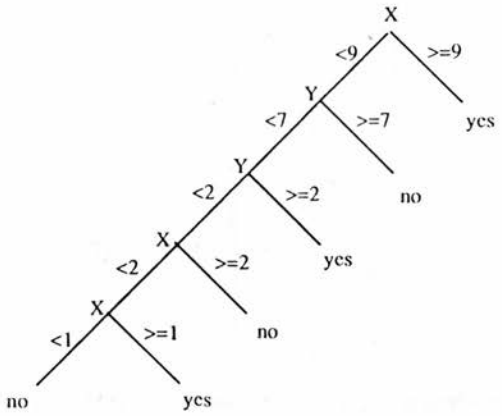
with an example using the training set shown in Table 2-1, a series of X and Y co-ordinates classified as either yes (if they lie on the straight line X=Y) or no (if they do not). By applying Quinlan's ID3 induction algorithm [Quinlan 79], which is used commercially in many shells (such as Expert-Ease), the decision tree shown in Figure 2-1 is generated.

This contains several rules which deal with the specifics of the co-ordinates given in the training set (for example, the classification is yes if X >=9 or if X <9 and Y is <7 but >=2), but does not contain the obvious, higher-level classification rule. At best this is computationally inefficient: at worst it could result in misclassification, which, depending on the domain, could be a serious matter. For example, using the decision tree in Figure 2-1, the co-ordinate pair (8,8) would be given a 'no' classification.

However, there are some domains where rule induction may be suitable, and Bloomfield attempts some criteria for identifying these, such as domains where there are 'sufficient examples available to construct a training set which constitutes a comprehensive encapsulation of expertise in that domain'. Quinlan cites fault diagnosis and chess end games as being two examples of domains where all possible examples are known [Quinlan 79].

Inductive shells can run on a range of hardware from a personal computer to a mainframe. Some examples include 1st-Class and Super Expert

Figure 2–1: A decision tree induced from the sample training set



(both [Harmon *et al* 88]; TIMM and RuleMaster (both [Lehner & Barth 85]); and Expert-Ease [Bloomfield 86]. In addition, there are two small shells which were both used at Moray House college during the expert systems module described in Chapter 1: I-1, which was used during the first year of the module and which is evaluated in the next chapter [PAL86]; and REFINE, which was used in subsequent years [REF87].

2.4.3 Mid-size rule-based shells

These are defined by Harmon as shells which can run on personal computers or workstations, with a KRL again based on production rules, but which is more structured than that found in the smaller rule-based shells. Thus, rules can be arranged into hierarchical sets, and use can be made of multiple instantiation and simple inheritance: some of the more powerful shells provide contextualisation, where rules can be assigned to specific contexts and one set of rules can inherit information acquired from rules in another context. By and large, such tools

also have more sophisticated methods for handling uncertainty, and the ability to write rules which specify a particular inference strategy.

Some examples of mid-size rule-based shells include KES [Harrington]; M.1, Nexpert, KES 2 (a later version of KES) and GURU (all [Harmon *et al* 88]); and Personal Consultant Plus [Forsyth 86, Epp *et al* 88].

2.4.4 Large rule-based shells

Most of the shells described in this category are not significantly different from those categorised as mid-size rule-based shells: the main difference is that they are designed to run on mini- and mainframe computers, that is, to be integrated into traditional computer environments. Although this can have advantages, Harmon claims that for most of these tools '...power and flexibility have been sacrificed to obtain compatibility' (that is, compatibility with the host mainframe), and that many of them are usable only by experienced computer programmers.

Among the shells he names in this category are Envisage and OPS83 (which run on DEC VAX machines) and S.1., ESE and TWAICE, (which run on IBM machines).

2.4.5 Hybrid shells

These are tools which have a KRL based on representations other than production rules. Often, it is based on a combination of production rules and structured objects such as frames or semantic networks, and as such allows representation of structure and relationship, with features such as inheritance (representations based on structured objects are discussed in detail in Section 2.1.2). These shells are very powerful and flexible, and are mostly used to build very large, complex expert systems. A summary of the features which they offer includes multiple representation formalisms (such as the facts, schematas, rules and viewpoints offered by ART) which allow separation of procedural and declarative knowledge;

flexible inheritance mechanisms; truth maintenance; alternative worlds or viewpoints on a knowledge base; sophisticated editing facilities for knowledge base development; choice of forward or backward chaining and also 'agenda control'; and sophisticated HCI tools for user-interface design.

Hybrid shells can be divided into mid-size tools such as Goldworks [Harmon *et al* 88], Leonardo Level 2 and Level 3 [Leo89], and Nexpert Object [Epp *et al* 88]; and large tools such as ART, KEE, Knowledge Craft (see [Laurent *et al* 86] for a detailed comparison of these), Leonardo/VAX [Leo89] and LOOPS [Harmon *et al* 88, Jackson 90]. There have been very few small expert system shells with a KRL offering more than production rules. One early example was ESP Frame-Engine [ESP86], one of the shells which was part of the initial evaluation described in the next chapter. As a commercial product it was very unsuccessful, perhaps because of the complexity of its KRL and the poor quality of its documentation. FLEX, a new shell which has recently appeared for the Apple Macintosh, has many features such as frames, forward and backward chaining, and daemons, but it is fairly difficult to use and like ESP Frame-Engine it also seems to be doing badly commercially.

2.4.6 A summary of possible features

Table 2-2 summarises the most important features which were available in the above shells, and indicates which of these are currently found in the EXPLORES shell [1]. The three features marked with an asterisk (how and why explanations and uncertainty handling) are features which were available in CRESS (the shell upon which some of the EXPLORES shell is based) but which were removed due to lack of memory when the EXPLORES shell was developed. However, it would be a simple matter to include them in later versions of EXPLORES, if required.

---

[1] In addition, EXPLORES has the ability to generate domain-based explanations, a feature which none of the shells mentioned in this chapter have. We return to this point in Chapter 5.

Table 2–2: Features of existing expert system shells

List of Features	Present in EXPLORES
Dedicated building environment	yes
Built-in editor (text-based)	yes
Built-in editor (graphics-based)	no
Browsing facilities	yes
Checking of knowledge base	yes
Multiple representation formalisms	yes
Alternative worlds/viewpoints/contextualisation	no
Flexible inheritance	yes
Rule sets	no
When daemons	no
Truth maintenance	no
Choice of forward/backward chaining	no
Agenda control	no
Conflict resolution	no
Uncertainty handling	*
How explanations	*
Why explanations	*
Display files	no
Use of complex algebraic expressions	no
Access to low-level programming languages	no
Access to database or spreadsheet s/ware	no
WIMP environment	partial
HCI tools for user-interface design	no

Many of the feature listed in Table 2–2 relate to the inferencing done by the system during a consultation, in terms of the ways in which knowledge can be represented to make inferencing more efficient. Features in this category include when daemons (rules which fire as soon as a tagged value changes); rule sets (the grouping of rules into forward chaining sets as described in Section 2.2.2); the ability to control the use of forward and backward chaining; and the use of conflict resolution strategies. Other features depend on the availability of reasonably sophisticated graphics facilities: these include the use of a graphical editor; the provision of a full WIMP environment; and the provision of tools for user-interface design. In addition, the availability of a reasonable amount of computing power makes it possible to include features such as truth maintenance; hooks to database and spreadsheet software; and access to low-level programming languages.

The development of the EXPLORES shell was constrained by the lack of facilities such as a full WIMP environment and graphics tools, as well as the power and memory restrictions imposed by the hardware on which it was implemented. Thus, the shell does not have any features which depend on these. More importantly, the features of the shell were restricted by the criteria established for its design during the initial evaluation, described in the next chapter. For these reasons EXPLORES does not have any features which require the user to consider the inferencing which will be done during a consultation, and has several features such as a built-in editor and a knowledge-base checker, which make the task of building a knowledge base easier. An overall criterion for the design of the shell was that it be simple to use for both developing and building a knowledge base, and this was a major reason for the exclusion of many of the features in Table 2–2, even in circumstances where it would have been possible to include them. However useful they may be in other environments, schools would have little use for features such as contextualisation of a knowledge base, when daemons, agenda control or conflict resolution: they merely introduce an extra, unnecessary level of complexity into what is already a difficult task. Finally, there are features which although useful in some circumstances, are generally not partic-

ularly desirable features to have. An example of such a feature would be the use of display files, which are a way of displaying text on the screen by linking the knowledge base to text files stored separately. These files are then displayed automatically under appropriate conditions or specifically when requested by the user. Canned text such as this is often used to explain certain aspects of the domain knowledge to the user. However, it can present problems, for example with regard to its consistency and completeness if a knowledge base is altered or updated. We return to this point in Chapter 5, when we describe a technique for the generation of domain-based explanations using the knowledge in a knowledge base, which presents no such problems.

A full description of the EXPLORES shell and the features which it provides can be found in Chapter 6.

2.5 Summary and Conclusions

In this chapter we have discussed knowledge representation and expert system shells. We began by describing production rules and frames, two knowledge representation formalisms upon which a knowledge representation language might be based. After discussing some of the inference techniques which can be used by a system to reason with the knowledge in a knowledge base, and some methods of representing uncertainty in expert systems, we went on to analyse a number of currently available expert system shells in terms of their knowledge representation languages and the main facilities which they provide. We finished by discussing the features of existing shells with regard to the EXPLORES shell, listing the features which it does and does not have, and justifying why the shell has the features that it does and why other features are missing.

As illustrated by Table 2-2, there are many features which a new shell could have, and, realistically, it is not possible for any one shell to provide everything. In deciding which features to include and which to leave out, it is not necessarily a case of deciding which features are 'better' than others: even if this could be

quantified, it is not necessarily the criterion by which a feature should be included or excluded. The criteria which should be used involve careful consideration of factors such as who the potential users are likely to be; what level of computer skill will be required to use the shell; how much time a user will be able to spend on building a knowledge base; and, most importantly, what the main use of the shell is likely to be. In the case of the EXPLORES shell, we have considered many such factors with regard to the features which it has and the knowledge representation language which it uses. We will discuss these factors in full in the next chapter, and in Chapter 4 we will describe the EXPLORES KRL, outlining the knowledge structures used in it; the kind of knowledge which can best be represented using it; and its benefits and limitations.

Chapter 3

The Initial Evaluation

In Chapter 1 we described the background to this research project, in terms of its aims, objectives, and personnel. We also discussed how teachers at the three participating institutions would be part of the initial evaluation, and how this evaluation would provide the feedback upon which we could base the design of a new expert system shell intended for educational use.

In this chapter we describe that initial evaluation, outlining the methodology used and the results obtained. We then discuss these results, touching on related work in this area, and finish by outlining a set of criteria upon which the design of the new shell should be based.

3.1 The Evaluation

In Chapter 1 we described the three institutions which agreed to participate in this research project, in terms of the Diploma courses run there and the adjustments which were made to those courses to accommodate this project. We will now describe the initial evaluation, with regard to the ways in which the teachers at those institutions participated in it, both formally and informally, and to the other aspects of it which were carried out away from the participating institutions. All of the initial evaluation was formative rather than summative, using techniques such as questionnaires, interviews and discussions. After describing

the results which we obtained from it, we will discuss their implications both specifically for the design of a new expert system shell for educational use and more generally for the introduction of expert system shells into an educational environment.

3.1.1 Methodology

All of the teachers at the participating institutions were given some experience of at least one expert system shell from the five available, although, as described in Chapter 1, the amount of experience which they were given varied across institutions. Following this, the teachers were then in a position to contribute to our evaluation of these shells. The complete evaluation was formative in nature and had several parts, some formal and some informal, some involving the teachers and some not. We will now describe each aspect of it in more detail.

Questionnaires and interviews

The more formal part of the evaluation was carried out using questionnaires and interviews. The questionnaires were used to evaluate individual shells: teachers were asked to complete a questionnaire after they had used a new shell during their course. The questions which they were asked related to various aspects of that shell, such as rating how easy they had found it to consult an existing knowledge base or to build one of their own (the full questionnaire can be found in Appendix C). Many of the teachers were also interviewed: these were taped, and were semi-structured in format, in that although there was a set agenda of topics which the interviewer wished to cover, there was not a specific list of questions which had to be rigidly adhered to. During these interviews, the teachers discussed their opinions of specific shells which they had used and their general feelings about the possible use of expert system shells in the classroom.

Three teachers elected to do a final project which involved making use of an expert system shell: two from St. Martin's College and one from Moray House. In addition, one teacher at Moray House carried out a project involving the

development of a small expert system shell which would run on a BBC micro, and another looked at uncertainty in expert systems. Although these projects were inspired by the teachers' participation in the expert systems module, neither made use of any of the shells used there, and as such they could not directly contribute to this initial evaluation. However, one of these projects does highlight a problem which we mentioned in Chapter 1 and will return to in this one: that of schools having the appropriate hardware available on which to run existing shells.

The two teachers at St. Martin's College carried out projects using the Xi expert system shell. One developed a knowledge base whose domain was a biological taxonomy of insects, which could be used for identification and classification: the other teacher developed a knowledge base for weather prediction, which would predict the weather for the next day based on inputs from the user about current conditions. The teacher at Moray House carried out a project using the APES expert system shell, for which he developed a small knowledge base containing grammar rules for Latin verb formation. All three teachers were interviewed both during and at the end of their projects, again using a semi-structured format.

Informal observation and discussion

In addition to the more formal evaluation we carried out some informal evaluation: this mainly consisted of informal observations of and discussions with the teachers at each of the institutions. In the case of Moray House, both researchers attended all ten sessions of the expert systems module; in the case of St. Martin's College, one researcher was present during both of the short introductions held there; and in the case of the CBLU, both researchers were present during the short introduction run there. During all of these encounters we were able to observe teachers using the shells and could note any problems they encountered, as well as being able to discuss these problems with the teachers *in situ*. We were also able to listen in during the many discussions which took place: these were usually about the shells which the teachers had been using and about the

use of expert system shells in education. This informal contact proved to be a valuable source of ideas for the design of the new shell.

Personal survey of existing shells

A personal survey of some existing shells was also carried out by the author. For a number of these, this was based on extensive use of the shell for building and consulting knowledge bases. For others, it took the form of reading and analysing published reviews. Table 3–1 lists all of the shells which were surveyed, in both of these categories.

This survey also provided many pointers to desirable and undesirable features for the new shell. [1]

School case-study

The final aspect of the evaluation was not concerned with the use of expert system shells as such, but instead with the practical side of computer use in schools, with regard to factors such as the hardware used; the software available; who computers were used by; and the purposes for which they were used. Although these were topics which were discussed with teachers during their interviews, it was felt that first-hand experience of computer use in schools would help to give a realistic picture of whether or not there is a place there for expert system shells. In addition, it would also highlight potential barriers in the way of their successful introduction. For these reasons we arranged a visit to a 'typical' Scottish secondary school in Fife to spend a day with the Principal Teacher of Computing (who was a former student on the Diploma Course at Moray House). This took place on the 5th of November 1986, and again it should be noted that

---

[1] At a later stage in the project other shells were also evaluated through personal use, published reviews and demonstrations. Although it was by then too late for them to have any influence on the design of the shell, they provided a useful basis for comparison. The shells in this category include Expert Builder (described below), Goldworks, Leonardo 2 and 3, and FLEX (described in the previous chapter).

Table 3–1: Shells surveyed during the initial evaluation

Used for building and consulting	
APES	[Hammond & Sergot 85]
Xi	[Exp85, anon 85]
Xi Plus	[Forsyth 87]
ESP ADVISOR	[ESP85]
ESP Frame-Engine	[ESP86]
I-1	[PAL86]
ADEX-Advisor	[Briggs 87]
EGSPERT-Browser	[Briggs 87]
Published reviews	
Crystal	[Curran 86]
Personal Consultant Plus	[Forsyth 86]
TIMM	[Lehner & Barth 85]
RuleMaster	[Lehner & Barth 85]
Expert Ease	[Bloomfield 86]
Kee	[Laurent *et al* 86]
Art	[Laurent *et al* 86]
Knowledge Craft	[Laurent *et al* 86]
KES	[Harrington , Allwood *et al* 85]
TESS	[Allwood *et al* 85]
SAVOIR	[Allwood *et al* 85]
EX-TRAN 7	[Allwood *et al* 85]
ENVISAGE	[Allwood *et al* 85]
SAGE	[Allwood *et al* 85]

what we report here reflects the situation as it was then, and that many aspects may have changed in the intervening years.

During our visit to the school, we discovered that Fife Regional Council Education Committee had set up a local Computer Studies certificate which pupils in the region could work towards. For this, the Council had produced worksheets and information booklets for both teachers and pupils. To achieve the certificate, pupils had to pass a two-part exam which contained a wide range of questions on a number of topics. The exam papers which we were shown had questions such as predicting the output of a simple LOGO program; labelling pieces of hardware and showing how they should be connected; and setting up a database for an imaginary collection of albums and tapes. Some of the pupils we observed in the school were working for this certificate (see below).

The school had 20 BBC B+ micros and one Amstrad PC, although the latter was mainly used by the school secretary for word processing. The Computer Studies department had one full-time member of staff and eight part-time members who were volunteers from other departments and who, by and large, had very little computer experience. The department had a software budget of approximately £1000 per annum, and the Principal Teacher was very keen on buying a lot of new software of various kinds. On the day we spent there the pupils had available equipment such as light pens, Acorn databases, speech chips, AMX mice and AMX PageMaker, LOGO, a small expert system package put together by a lecturer at Moray House College, games and graphpads.

During our visit, we were able to observe pupils in classes from four out of the six years in which there were pupils: we did not see any fourth or sixth year pupils. During all of these classes, the teaching policy seemed to be for the children to be given worksheets and for the teacher to play a mainly supportive role, giving help where necessary. The activities we observed in each of the classes were as follows:

First year The work done by first year pupils was mainly on computer familiarisation. For example, they had worksheets showing parts of computer

systems, such as a mouse, a keyboard and a monitor, each of which had to be labelled. They were also given games (on paper) such as a wordsearch grid where all the words to be found within the grid were computing terms.

Second year The second year pupils we observed were using LOGO turtle graphics, where they were given worksheets with exercises of increasing complexity, for example from drawing a square to drawing a boat.

Third year We observed three third year classes. The first of these was a class working for the regional Computer Studies Certificate described above. We observed them as they were starting to use simple BBC BASIC graphics (such as MOVE and DRAW).

The second class we observed was a 'fun' class for less able children, who were using the light pens (for drawing), the Acorn speech synthesiser, and lots of games software. These children were not working towards any kind of computing qualification.

The third class contained pupils who were starting to work towards the Standard Grade Computing Studies certificate, which had just been introduced. They were working with Acorn databases on a number of topics (such as **Fashion** and **Vehicles**), and had worksheets which set them tasks such as 'Find the names of all the people who have blue Fords'.

Fifth year The fifth year class consisted of pupils who had been too young to leave school at the end of their fourth year, and so had to remain there until they reached the required leaving age. Although not working towards any specific computing qualification, the pupils in this class were using the same databases and worksheets as the third year Standard Grade class.

3.1.2 Results

We will now discuss the results of the above evaluation, in terms of what we found out about the individual shells used by the teachers; about the educational use

of shells in the future; and about the features that an ideal shell should have if it is to be used successfully in the classroom.

Individual shells

The shells which were used and assessed by the teachers were Xi, ESP ADVISOR, APES, I-1 and ESP Frame-Engine. Of these, the assessment of the last two is based on one afternoon's use by teachers at Moray House; assessment of APES is based on one afternoon's use by teachers at Moray House, and the extensive use made of it by one teacher during his final project and by a teacher fellow at Leeds who was interested in this project; assessment of ESP ADVISOR is based in some cases on one afternoon's use and in others on a one-day introduction; and assessment of Xi is based in some cases on one afternoon's use, in others on a one and a half day introduction, and in two cases on extensive use of the shell for a final project.

Most of the teachers found consultation of a knowledge base a relatively simple matter for all of the shells, saying that it was 'straightforward' and 'fairly easy'. Shells which led the user straight into a consultation (such as ESP ADVISOR and Xi) were considered to be easier to use, in that once a knowledge base was loaded the consultation started immediately: it was not necessary to know what was in the knowledge base in order to consult it. This was not the case with APES, for example.

Most of the problems arose when teachers tried to develop their own knowledge bases: many of them found it more difficult than expected, not realising, for example, 'the amount of detail needed'. It is mainly on this task that we will focus in the following descriptions of the teachers' assessments of each of the shells.

On a general note, many of the teachers commented that the shells they encountered were not what they expected: for example, they were 'not as sophisticated as I thought they might be', and were 'more primitive than ... imagined and more cumbersome than expected'. This was in part due to the absence of

large, non-trivial knowledge bases: all of the shells had several demonstration knowledge bases, most of which were small and trivial. Many of the teachers felt that these did not fully demonstrate the potential of the shells, and as such they were suspicious about what the limitations of the shells might be. We will return to this point below, during our discussion of the results of this initial evaluation.

Xi

Xi is a small rule-based shell which supports both forward and backward chaining. It has three modes of use (TOOLKIT, DIALOG and TUTORIAL); a built-in editor; facilities for volunteering an answer and changing an answer during a consultation; and a KRL which allows the user to represent rules, questions, facts and default values in an 'English like' way (see Figure 3–1).

Of the teachers who used Xi, most agreed that for anything other than trivial knowledge bases it was 'cumbersome' and 'difficult to use in places'. The Xi editor, provided as a tool within the shell, was unanimously judged to be 'awful': the main complaint against it was that editing had to be done by replacement of a whole line of text or an entire rule, which was very time-consuming. Overall, it was seen as being 'very clumsy'. As an alternative, it is possible to build a knowledge base using a text editor and to load it at run-time: however, this is also very slow, particularly for a large knowledge base. (One of the teachers using Xi for his final project cited a time of over 15 minutes to load a knowledge base containing approximately eighty rules). This problem is compounded by the fact that once loaded, the knowledge base may be found to contain errors, and so must be edited using the Xi editor or using a text editor outwith the shell as before. Doing the latter means leaving Xi, making the appropriate alterations to the knowledge base, reloading the shell, and then reloading the knowledge base. This is a very time-consuming process.

The user manual for Xi was also judged to be inadequate, in that 'it doesn't give you enough detail... it takes a lot of working out'. The underlying inferencing also came in for criticism from teachers who had looked into it: one called it '...horrific. Really horrendous: backwards and forward chaining all mixed up...'.

Figure 3–1: Examples from an Xi knowledge base

Rules:

if weather is fine
and time is before_lunch
then picnic is possible
and gardening is rescheduled

if children is yes
then check child_allowance

Questions:

question weather
What is the weather like?
any of poor, average, fine, fair

Facts:

sky is blue

Default values:

default height is average

However, on the positive side, many teachers did express the opinion that for small, well structured domains they could see a use for a shell such as Xi.

Both teachers at St. Martin's College who attempted final projects using Xi found it very difficult to use, and both felt that they reached its limits very quickly. During their projects, both teachers had to be persuaded not to give up: they felt that the shell was 'too hostile' and 'too slow', and that by using it they were endangering their prospects of getting a good mark for their project, which counted for 30% of their final mark for the course. In the event they were persuaded that the systems which they built would be treated as prototypes and marked accordingly, because the work they were doing was so experimental.

The teacher who built the forecasting system, whom we will refer to as S, was a secondary school Science teacher with 9 years experience. He had taught a variety of Science subjects, such as Rural Science, General Science, Biology and Chemistry. As his main subject was Rural Science, which he saw as being 'a dwindling subject', S had decided to move into the computing field, and as a first step had enrolled in the St. Martin's course. He was in the unusual position of having moved to a new part of the country during the period of the course, and as such would be unemployed when it ended. Ideally, S was looking for employment not so much in Computing Studies but in Educational Computing, being a firm believer in the use of computers throughout the curriculum.

S found Xi to be very slow, and the whole process of using it to be very time consuming. In addition to the editing problems mentioned above, he also encountered the problem that the system did not degrade gracefully, and occasionally unforeseen errors would cause it to crash completely, losing any unsaved additions to a knowledge base. S estimated that over the course of his project he lost almost three and a half days of work this way.

S admitted that some of the problems he encountered were not necessarily to do with Xi: he had chosen a domain which did not fit neatly into a set of production rules. He said that he 'took on too much as a topic', and that he 'should have chosen a smaller, more well-defined domain'. His choice of a more

'vague' domain was partly deliberate, in that he wanted to

> ...get away from the sort of narrow idea that the only thing that you can put in an expert system [is] something that [has] already been written as a series of rules...

He argued that 'expert system shells are only going to be really any good if they can apply themselves to all sorts of knowledge'. S felt that many of his problems might also have stemmed from his own inexperience in the area of knowledge representation: he felt that he lacked the correct 'strategies' for this, and so the whole process took longer than it should have. He was disappointed with the finished product, although he had done all that he could in the time available.

S concluded that although in some ways Xi was very simple, in others it was overcomplicated for the kind of applications that novices would build. An example he cited was the fact that it has three different modes, one of which is a subset of another. However, he said that overall he liked the idea of expert system shells, even if this particular implementation was frustrating to work with: it was 'early days'.

The second teacher from the St. Martin's course who used Xi will be referred to as B. Prior to doing the course, B had been a Biology teacher for ten years, after doing a degree in Zoology. He had always taught in secondary schools. A discussion with an Adviser in his home school had led B to realise that there was a shortage of Computer Studies teachers, and that there would be an opening for him if he re-trained. Already interested in computer programing, B decided to enroll in the St. Martin's course, after which he would return to teach Computer Studies, either in his home school or working between three schools in the area.

During his interviews, B echoed many of the problems with Xi that had been encountered by S and by the other teachers, although overall he found things slightly easier than S did. This was mainly due to the fact that the domain which he chose (insect classification) could be made small enough and was well-defined enough, in that it had many mutually-exclusive alternatives, to fit well into a production-rule representation formalism. He chose this domain because using a reference book for such a task can 'take hours, even for a few simple features'.

However, B admitted that he 'could see problems with anything which was more vague'.

The organiser of the component of the St. Martin's course into which Xi had been introduced had doubts about whether he would use the same shell again the following year. He said that although Xi was 'a very good introductory shell as far as it goes', he 'wouldn't encourage any of them to use [it], because the hassles ... and inconsistencies that [it] presents are too great to bear'.

ESP ADVISOR

ESP ADVISOR is marketed as being a shell designed for the task of 'text animation', that is, the structured representation and presentation of a body of text (for example income tax regulations), only some of which may be relevant to the user. The 'expert' part of the system deduces what text in the knowledge base is relevant by asking the user questions and then making inferences based on the responses given and the knowledge in the knowledge base. The result of a consultation is the presentation of the relevant text on the screen.

An ESP ADVISOR knowledge base can be thought of as a number of paragraphs of textual advice or information, divided into one or more sections, where each section is concerned with a particular aspect of the domain. Each section will typically contain paragraphs of text; references to other sections; and parameter definitions. The displaying of paragraphs of text and the processing of other named sections caused by section references can be unconditional (always carried out) or conditional (carried out only if stated conditions hold). Conditions are expressed in terms of acceptable parameter values, and there are four types of parameter (fact, number, category and phrase). Values for parameters are found using either rules or by asking the user: the method to be used (and the appropriate rules or questions) must be stated in the body of the parameter definition. Parameters can also be used to display values on the screen. Figure 3-2 shows some examples from an ESP ADVISOR knowledge base. The shell has no facilities for knowledge base development: this must be done using a text editor, outwith the shell.

Figure 3–2: Examples from an ESP ADVISOR knowledge base

Conditional paragraph:

```
{student}
    'Since you are a student my next task' &
    'is to find out if you get a grant'.
```

Conditional section reference:

```
{student}
    reference get_grant.
```

Parameter definition (fact):

```
student: 'the user is a student'
fact
askable
    using 'Are you a student'.
```

Parameter definition (number):

```
mature_students_grant: 'type of grant given'
number
rule
    students_grant + 2000
```

Although many of the teachers were positive about the use of ESP ADVISOR for consulting a knowledge base, most found it difficult to then develop their own knowledge base for it. This was partly due to the fact that when representing knowledge in the ADVISOR KRL, the user must take into account the inferencing done by the system during a consultation, and so must structure the knowledge base accordingly (for example, a section cannot reference another section which comes before it in the knowledge base). Many of the teachers objected to this, saying that 'it was difficult enough without having to think about that', and that the 'inferencing should be done for you'. In addition, they felt that the kind of representation used by the shell was unsuitable for most types of knowledge, and that the domains in which it was applicable constituted a rather small group.

ESP ADVISOR is structured in such a way that a knowledge base must be built outwith the shell using a text editor, and then compiled using the shell, and it is only when it has been successfully compiled that it can then be consulted. This means that the ADVISOR KRL is very precise, and that the compiler is very fussy about what it will and won't accept with regard to syntax. Although this is the case with the KRLs of many shells, the problem is exacerbated within the ADVISOR KRL because its syntax is particularly complex. In the event, most of the teachers quickly became bogged down in syntax errors which had to be corrected before the knowledge base which they were developing could be used. The errors found were often trivial and frustrating, such as a missing comma or use of a reserved word as a variable name. This was usually compounded by the fact that one error could have repercussions throughout the entire knowledge base, and could be reported as fifty errors. In addition, the error messages produced were singularly unhelpful, and so finding and correcting the errors was also difficult. Most of the teachers commented on these problems, and informal observations of them building ADVISOR knowledge bases showed that most of their time was spent on a continual cycle of editing → compiling → tracking down syntax errors → re-editing.

Many of the teachers commented that it would take a lot of time to 'get to grips with the basics' of ESP ADVISOR, and that for use in schools the KRL 'would

have to be made simpler', as 'pupils wouldn't be able to cope with this'.

apes

APES (Augmented Prolog for Expert Systems) is an expert system shell based on the programming language micro-Prolog, and as such it contains many Prolog-like features. An APES knowledge base can be developed from within the shell using the built-in editing facilities, and knowledge can be represented as facts and rules. When information is needed from the user, the appropriate question is generated from the relevant rule, and there are facilities for constraining the answers which can be entered in response (using definitions such as 'unique-answer' and 'valid-answer'). User-system interaction is largely through micro-Prolog queries and responses, although the knowledge base developer can define 'natural language' templates and menus. Figure 3–3 shows some examples from an APES knowledge base.

Opinions were divided about APES, in that some of the teachers found it easy to use while others found it impossible. All of the teachers who used it had previous experience of Prolog programming, and many of those who responded positively seemed to be teachers who had found that relatively easy, although we have no concrete evidence for this. One interesting point which emerged was that even teachers who had few problems building their own knowledge base found it difficult to debug or to add to an existing one.

Many of the teachers thought that APES seemed to be the least trivial of the shells they had encountered, and that once they had mastered the representation language, they would be able to build quite powerful and non-trivial systems. However, few of them thought that it would be very useful in the classroom, since initially it was so difficult to use and so difficult to understand.

The teacher from Moray House who used APES in his final project, and whom we will refer to as G, was a secondary school teacher who had taught classics for fourteen years, and was a Principal Teacher in his home school. His participation in the Moray House course stemmed from his pessimism about the future of classics as a subject taught in day schools, and from his interest in computing.

Figure 3–3: Examples from an APES knowledge base

Rules:

```
_m grandfather-of _p
    if _A parent-of _p
    and _m father-of _A

_f grandmother-of _p
    if _f mother-of _A
    and _A parent-of _p
```

Facts:

```
William father-of Peter
Evelyn mother-of Peter
```

Query the user (no template):

```
Which _A: _A father-of William?
```

Template definition:

```
which-template(father-of (_f _s) (_f)
    (Who is the father of _s?))
```

Query the user is now:

```
Who is the father of William?
```

G experienced several problems with APES. The version he was using ran on an MS-DOS Amstrad PC1512, with 640K of memory, and even with the small knowledge base that G developed there were problems with shortage of memory. In addition, he found several bugs in the shell, and was very critical of the inadequate level of documentation provided with it. It took a long time for G to become familiar with APES: he built several small knowledge bases before feeling that he had the competence to tackle the one for his project. This meant that in the event he lacked the time to make full use of many of the facilities of APES which would have improved his system, such as the use of question- and answer-templates or explanation files. He was reasonably satisfied with the end result, recognising its limitations. However, he questioned whether or not it was worth the amount of time he had spent on it.

I-1

I-1 is an inductive shell, that is, it uses an induction algorithm to induce a decision tree from a set of domain examples (known as a training set). I-1 has editing facilities for entering examples, and a language for the user to query the resulting decision tree. Before a training set can be entered its structure must be defined, in terms of the name and the type of the factors to be represented. The three possible types are descriptive (D), that is, information which is for for reference purposes only, and is not used in the induction process; integer (I); and enumeration (E), where a list of the possible categories must be given. The user must also state which of these factors are relevant in the induction process. Table 3–2 shows an example definition for a training set, and Table 3–3 shows an example training set for I-1.

Few of the teachers had much to say about the induction shell I-1. Although they found it simple to use for both building and consulting, many expressed doubts about the rules produced by it, and about the unlikelihood of users being able to provide it with a large enough set of typical examples in most domains. The problems associated with induction as an expert systems technique were discussed in the previous chapter, during the review of inductive shells.

Table 3–2: Defining a training set for I-1

FACTORS		ENUMERATION TYPES			
Reference no.	D				
Age	I				
Qualifications	E	degree	professional	certificate	none
Years in job	E	1–3	4–7	8–12	>12

Table 3–3: An example training set for I-1

Reference no.	Age	Qualifications	Years in job
131	27	degree	4–7
132	39	professional	8–12
133	22	none	1–3
134	55	certificate	> 12
135	40	degree	1–3

ESP Frame-Engine

This shell was used in the Moray House module because at the time it was the only small shell using frames as well as production rules in its KRL: it was used for demonstration and consultation only. The Frame-Engine KRL allows knowledge to be represented as frames, instances, rules and questions. Frames are used to represent objects or classes of objects. They have **slots**, and each slot holds the name and type of a property belonging to the object represented by that frame (a default value can also be represented). One frame can be a subclass of another, and can inherit the properties of the superclass frame. Instances represent specific instances of a class of objects, and values for properties are established using rules (through inferencing) and questions (through asking the user). A knowledge base must be developed outwith the shell. Figure 3–4 shows from an ESP Frame-Engine knowledge base.

Most of the comments from the teachers about ESP Frame-Engine concerned the fact that its KRL was very complex: for example, they found it difficult to see how the frames in the knowledge base related to one another. However, a few teachers liked the ideas behind the language, that is, having knowledge represented as 'records or objects instead of just rules', finding it 'quite intuitive'.

The educational use of shells

We can divide the topic of the educational use of shells into four main subtopics, namely: *where* in schools shells could be used; *who* they could be used by; *what* they could be used for; and finally any barriers which could be seen to stand in the way of their successful introduction into schools.

Many of the teachers were in strong agreement that if introduced into schools, expert systems and expert system shells should be used throughout the curriculum, and should not be the sole province of Computer Studies departments. Many even went so far as to say that they thought Computer Studies should not

Figure 3–4: Examples from an ESP Frame-Engine knowledge base

Frames:

```
frame shape.
    slot number_of_sides
        type integer.
end_frame.

frame door is_a feature.
    slot shape_of_feature
        type instance_of shape
        default rectangle.
    slot expensive
        type boolean
end_frame.
```

Instances:

```
instance rectangle of shape.
    value number_of_sides = 4
end_instance.
```

Rules

```
if_needed number_of_sides.
    ask.
end_rules.

if_needed expensive.
    true if antique
        and wooden
        and carved.
    false.
end_rules.
```

Questions

```
value number_of_sides%question =
    'How many sides does it have?'.
```

be taught in schools, with statements such as:

> I don't agree with Computer Studies being taught as a subject. In so far as it is ...then expert systems will have their place within it, but ...I still think expert systems have a place within the whole curriculum

The main objection was that 'most pupils are computer literate fairly early on', and should be made to regard the computer as a tool '...to be adapted to our purposes rather than for us to adapt our practice to the machine'. For example,

> I don't know how a photocopier works precisely but I use one regularly ...[a computer is] a tool: it's a machine to be used, to serve a purpose, rather than the sort of inward looking view of the subject

However, many teachers were resigned to the fact that Computer Studies was likely to remain part of the syllabus for some time, and if that was the case then shells had a place there too, since

> ...expert systems are becoming quite popular in industry, and the Computer Studies syllabus must reflect what [pupils] are going to see when they go out of schools

For this reason, it was seen as the 'duty' of a Computer Studies course to introduce pupils to expert systems and expert system shells.

It must be pointed out that there was a notable difference between the teachers at Moray House and those at the two other institutions, in that the Moray House teachers were much more positive about the status of Computer Studies as a subject to be taught in schools (and consequently about the use of expert system shells within the subject area). This almost certainly reflects the growth of Computer Studies as a subject in Scottish schools at that time: many of the teachers on the Moray House course were from other subjects but were returning to jobs in Computer Studies departments. By comparison, the subject was in decline in England and Wales, having been taught in schools there for some time.

Most teachers felt that shells could be used by secondary school pupils of all ages (11-16) and abilities, although some were more specific, saying that for younger pupils in this age range the best use would be for consultation, and that

only older pupils would have the ability to develop their own knowledge bases. Overall, it was agreed that if shells were to be used in any way with first and second year pupils (11-13) then 'you've got the problem of making the system simple enough to use'.

Two main uses of expert systems and expert system shells in schools were identified. These were:

- having a number of non-trivial pre-built knowledge bases which pupils can consult in a wide variety of curriculum areas. Such knowledge bases could be used within a lesson by the teacher and by the pupils, or they could be used as an 'intelligent reference' which pupils consult when they feel that they want information about a topic. This was seen by many as the most appropriate use for younger or less able pupils, and as such they expressed the need for making the consultation facilities of the shell simple and user-friendly.

- pupils building knowledge bases to create their own expert systems. Some reservations were expressed about this, in that many pupils would find it easier to add to an existing knowledge base than to build one of their own from scratch. An extension of this was seen as being desirable by some teachers, where pupils co-operate to build a large knowledge base over a long period of time. The process of knowledge representation was viewed as being useful in that it forced pupils to '...order their knowledge, set it down in a logical way to be understood by a machine', and made them 'analyse their own thought processes', to give them 'some idea of how they are learning'.

Most of the responses to questions regarding possible barriers in the way of the introduction of expert systems and expert system shells into schools touched upon wider issues such as lack of time for teachers to become familiar with new software; timetabling and curriculum constraints; financial constraints, particularly with regard to schools having the kind of equipment needed to run such software; and the lack of any well-thought out, integrated policy with regard

to the introduction and use of computers in schools. The teachers were almost unanimous in singling out lack of appropriate hardware and the need for immense amounts of teacher training as the two most important factors which had to be dealt with before expert system shells could be used successfully in the classroom.

Many of the responses stated that teacher-resistance would also be a problem: that teachers would only agree to use such software if they were provided with an expert system which could be shown to remove some of their teaching load. For this reason, it was thought that the way forward in terms of winning teachers over was to

> ...construct some useful expert systems with the shells that are available and try them on a fairly broad basis since ...the only examples we've been able to see are trivial.

Resistance of pupils was not seen to be a problem as long they were capable of using the software without too much difficulty, and could see themselves making reasonable progress.

Ideal shell

The teachers were also asked what features they saw as being desirable in a shell for educational use.

One feature which none of the shells had and which many felt would be very useful was a graphics facility. This would make it possible to include pictures with any questions which are being asked of the user, for example to illustrate the possibilities in a classification system, and to present graphically as well as textually the result of a consultation.

With regard to having a shell which children could use to build their own knowledge bases, a few of the teachers mentioned the desirability of this task being done within the shell, saying that it

> would need to be highly structured. more like just filling in data rather than having to think about inference ...

On the general consulting interface of such a shell, most teachers stated that shells which made extensive use of menus were generally easier to use than those which didn't, and that overall a shell for schools 'needs to be very user-friendly'. Related to this, one feature was mentioned which only Xi had: this was the ability to, during a consultation, change a response made at an earlier point. One teacher said that 'it was irritating if I tried to change in the middle of a consultation, and had to go through the whole thing again'

Of the shells used, some teachers commented that they couldn't 'see the difference between these and intelligent databases', and that there were 'some things which we can do easily enough with database's so it's not worth while going into an expert system'. It was commented that for domains which are perhaps vague and not well-structured, 'then you want to go into an expert system where it will give out probabilities' (none of the shells in the initial evaluation had a facility for dealing with uncertainty). It was generally agreed by the teachers interviewed that if the shells they had seen were typical of what was available, then they couldn't 'see using them in education for a while'.

Overall, simplicity and flexibility were regarded as the most important criteria, although if possible not at the expense of power. More than one teacher stressed that 'it has to be kept simple', and the feelings of many were summed up by one who said that the shell should have:

> tremendous flexibility ... we want something which will enable you to go at it from a number of different directions so that you can ask questions of the system; you can offer information to it and it will then ask its own questions; and its not too fussy about the syntax in which you put it. It's looking for keywords ... if you can build a keyword search in it would be a lot easier

3.1.3 Discussion and conclusions

Before going on to discuss the above results in terms of their influence on the design of the new shell, there are several points which must be raised with regard to the adequacy of this evaluation, given that several unanticipated problems were encountered during it. The first of these was that although the project

was designed to have two researchers working on it, in the event there was only one for the most part. Although the temporary researcher in the Department of Education did a great deal of work in the short time that he was employed at the beginning of the project, lack of time meant that he was not able to produce a final report about the evaluation.

Another problem encountered was that only three teachers at the participating institutions chose to do their final year projects using an expert system shell. This was fewer than anticipated, and there were several reasons for it. The timing of this project was such that at one of the institutions all of the teachers had already chosen the subject of their end-of-course projects by the time the evaluation began, and no new teachers would be available until summer 1988: this was too late for any useful feedback with regard to the design of the new shell. At the other two institutions, although there were teachers who were willing to undertake a project in this area, most seemed to be afraid that, despite reassurances from their course organisers, doing something which was so experimental might jeopardise their chances of getting a high mark. For example, at one of the colleges, the project counted for around 30% of the final grade for the course. A telling comment came from one of the course organisers with regard to wasting the teachers' time by getting them involved in that sort of project:

> ...have you wasted their time, have they spent ...a lot of hours, 200, 300 hours on something which is of peripheral importance for the next two or three years within school education: and when it does become of importance in fact the software tools will be much more sophisticated and friendly. I have worried about that.

Time constraints also meant that teachers at two of the institutions had exposure to only one simple shell over a very short period of time (Xi for one and a half days at St. Martin's; ESP ADVISOR for one day at the CBLU and one day at St. Martin's). Although both of these institutions had agreed to participate in the project by introducing an expert systems component into their courses, in the event the course organisers found that doing this would take up more time than they had available. In addition, as the existing modules were already tightly

used in acquiring, organising and representing a body of knowledge. These are often referred to more generally as higher-order thinking skills, or metacognitive skills, defined by Brown as [Brown 87]

> An understanding of knowledge, an understanding that can be reflected in either effective use or overt description of the knowledge in question.

Although a full discussion of this subject is inappropriate here, it is one about which there is increasing interest with regard to the design and use of educational software in schools (for example, see [Weinert & Kluwe 87] and many of the papers in [Mandl & Lesgold 88]).

This suggests that the new shell should have facilities which promote both types of use, but particularly building. Building a knowledge base was a task that most of the teachers found difficult, and the general conclusion was that for use in schools it would have to be made as simple as possible. One way in which this can be done is to have a simple, flexible and expressive KRL which allows different types of knowledge to be represented, and where the user need not take into account the inferencing which the system will do during a consultation. Another possibility is to have as part of the shell a highly-structured environment built around the KRL, in which knowledge can be entered easily, perhaps with a means of providing feedback about the correctness of the knowledge base being developed.

There was some disagreement about the age at which children could use shells, with some teachers saying that all secondary school-aged children would be able to do both consultation and development of knowledge bases, while others thought that younger children (in secondary schools) could use shells for consultation and only older children would be able to use them for development. None of the teachers mentioned the possible use of shells in primary schools, but in theory there is no reason why this should not be the case. This is a problem which we need not address specifically, in terms of designing the new shell with a specific age range in mind. However, from a slightly different perspective, it seems desirable that it be suitable for users (both children and teachers) with a

structured, the introduction of a new topic would have meant that another topic would have had to be dropped, and this was seen as undesirable. They also felt that they could not produce an entirely new module just for this topic, as was done at the Moray House. This was not unreasonable: the organiser of the optional expert systems module there commented that developing it took 'a gigantic amount of work' and quoted a figure of at least five hours preparation time for every hour of time spent teaching the module. In addition, he also had to do a significant amount of background reading in the subject. For these reasons, teachers at two of the institutions were only able to give feedback on the one shell they had used, although they were still able to contribute to the general topic of the potential for using expert systems and expert system shells in schools.

In the event, many of the teachers stated that they would have liked much more exposure to each of the shells, and that in many cases they felt that the exposure they had was insufficient to make an accurate judgement. Often, it was difficult for them to distinguish between problems caused by their own lack of experience and those caused by some aspect of the shell they were using. This would have become apparent with more extensive use, but unfortunately, in most cases this was not possible.

Despite these problems, we were still able to get useful feedback from the teachers, from both the formal and the informal aspects of the evaluation in which they were involved. In addition, the one day school visit and the extensive use of several shells by the author were also valuable sources of input.

As described above, the teachers' comments on the way in which shells could be used in classrooms showed two distinct uses: pupils consulting pre-built knowledge bases and pupils developing their own knowledge bases (this includes developing a new knowledge base and altering an existing one in some way). From the feedback we received, teachers felt that pupils could benefit from both of these tasks: from consulting a knowledge base, they could learn about its domain and about the reasoning used by the system, and from developing knowledge bases they could learn much from the skills needed and the strategies

wide range of computing skills, neither being too complex for novice users nor too trivial for more experienced users.

The use of expert system shells throughout the curriculum found favour with many teachers. This can be regarded as being sympathetic with a wider view of the use of computers in schools known as the 'curriculum computing' approach [Conlon & Cope 89], which argues that 'computers should best be regarded as another type of classroom aid like the overhead projector or the textbook' instead of being used solely in the Computer Studies department. If expert systems are to be used in a variety of subjects, then the purpose of their use must be that they will be a tool which will in some way help pupils learn about the subject in question. This is in contrast to their use within a Computer Studies curriculum, where they will be used to teach pupils about expert system shells and related concepts such as knowledge representation. This corresponds to a 'computer literacy' or a 'vocationalist' view of the subject [Conlon & Cope 89]. We would propose that if the main use of expert system shells in schools is the latter then existing shells are probably adequate for this. However, if the most desirable use of shells is throughout the curriculum, and the findings of this evaluation suggest that it is, then we must look at how the new shell can be designed so that it is a tool which will help to promote learning in any domain for which a knowledge base can be developed.

The issue of whether or not expert systems and expert system shells can or should be used in schools with any degree of success is a far wider one than simply designing and implementing an appropriate piece of software. Other factors which must be considered are ones which affect the introduction and success of any piece of software into the classroom, namely having the appropriate hardware on which to run it; giving teachers enough training and enough time to become competent users of the software before expecting them to use it with pupils; and introducing the software as part of a well though out, integrated computing policy, with clear goals, aims and objectives. These were problems which many teachers expressed, not solely in the context of the classroom use of expert system shells: they have also prevented the take-up of other software

such as databases and word-processing packages in many schools. We observed first-hand the problems that this can cause in the school that we visited. There, software was bought on an *ad hoc* basis depending on what was available and how much money was left in the software budget (a more generous amount than would normally found in secondary schools), and the school was restricted to software which could run on a BBC micro. In addition, the eight part-time 'computing' teachers, all from other departments and with little or no computing experience, found it very difficult to find the time necessary to train themselves in the subject they were teaching. Even if they worked during lunch breaks and in the evenings, they were frequently only one or two steps ahead of the children, and often were unable to answer questions they were asked in class. In the event, most of the classes we observed were uninteresting and made little use of the potential that even a small micro such as a BBC has in the classroom. We have every reason to believe that this situation is all too common in secondary schools. A full discussion of the political and economic issues involved is outwith the scope of this thesis. However, discussion of these and other issues concerning the use of computers in education can be found in many pieces of published literature (for example [Conlon & Cope 89]).

3.2 Related work

Before we finish this chapter with a summary of the findings of this evaluation and a list of the conclusions which can be drawn from them, we will describe some related work in the area of the development and use of expert system shells in education (although there are many expert system shells on the market, few have been designed with educational use in mind). Specifically, we will describe three research projects which began around the same time as this one, and which produced four expert system shells: ADEX-Advisor, EGSPERT-Browser, Expert Builder and EESS (Educational Expert System Shell).

ADEX-Advisor and EGSPERT-Browser

In 1987 a report appeared which was the result of a project commissioned by the Further Education Unit (FEU) and the Manpower Services Commission (MSC) to explore the curriculum implications of using expert systems within Further Education [Briggs 87]. The project produced a 'starter pack' containing two simple expert system shells (ADEX-Advisor and EGSPERT-Browser) and some expert systems developed using these.

The development of these expert system shells resulted from an analysis of some existing shells, which indicated that there was a great need for much simpler shells if they were ever going to be used successfully in Further Education. EGSPERT-Browser was designed as a stripped-down expert system shell, with a simple rule syntax and query system and a simple model of use: it is a browsing system, where the query commands 'walk over' the rules. An example from the report illustrates this using a knowledge base for diagnosing what can go wrong when cooking a souffle: Figure 3–5 shows some of the rules from that knowledge base.

EGSPERT-Browser allows the user to browse these rules by first of all displaying the top-level goal: 'souffle is imperfect' in Figure 3–5. The query lan-

Figure 3–5: Rules from an EGSPERT-Browser knowledge base

```
RULE1
souffle is imperfect if
        souffle is black on top

RULE2
souffle is imperfect if
        souffle is watery
                :
                :
RULE7
souffle is black on top if
        temperature is too hot

RULE8
souffle is black on top if
        cooking time is too long
                :
```

guage has the options **why**, **if** and **similar**. If **why** is selected, the system displays all of the premises for rules which have that conclusion, such as 'soufflle is black on top' and 'soufflle is watery'. If the query **if** is selected, the opposite happens, that is, the system displays the conclusions of all rules in the knowledge base which have the selected premise. The final option, **similar**, displays concepts which are similar to the selected one. For example, if the selected concept is 'temperature is too hot', then similar concepts which could be displayed are 'temperature is uneven' and 'temperature is too cool'. Thus, EGSPERT-Browser allows fairly effective browsing of a simple knowledge base, in three different ways.

Briggs' other shell, ADEX-Advisor, is a more standard, simple, rule-based shell, with a built-in editor. Advisor has two types of rules: *advice* rules, which are really top-level rules such as

```
advice Take a scarf if
        you are going outside and
        weather is cold
```

and *other* rules such as

```
weather is cold if
        temperature less than 7 degrees C
```

When rules are being added to an Advisor knowledge base, a menu displays the conclusions of all of the rules which are there already, and which could form the premise of a new rule (such as 'weather is cold' in the above example).

ADEX-Advisor has been used to build a number of expert systems in a variety of domains such as food and nutrition [Christian-Carter 87]; history [Storrie 87]; geography and biology [Hassell 87]; and many others [Briggs 87]. There are many positive comments in these publications regarding the ease with which teachers involved in the project were able to use both shells, and how quickly they achieved working systems. However, there is some question as to how much this success was due to the amount of input the teachers had from researchers on the project and the amount of time they were able to spend working with the

shells. In addition, it is significant that most of the knowledge bases developed were very small and trivial. There are several other negative points with regard to ADEX-Advisor: its KRL is very weak (for example, it has no facility for using variables within a rule); it's explanation facilities provide even less than is normally available on a small shell (it provides 'how' explanations but not 'why'); and it has a poor user-interface. Having said that, there is no doubt that their simplicity is a strong point in favour of both ADEX-Advisor and EGSPERT-Browser. Many users found that they could produce a working, if trivial, knowledge base in a very short space of time, which is more than can be said for many of the shells evaluated above.

Expert Builder

Another research project which began around the same time as this one was concerned with the development of a software environment for modelling, to be used in education. This was envisaged as a modular system which would allow users (mainly children of secondary school-age) to construct models in a range of subject areas and in several different modelling domains. An early feasibility study concluded that expert system shells could provide a suitable environment for building 'conceptual models which are predominantly qualitative' [Webb 87, Webb 88], and a list of desirable features was produced, which stated that a suitable shell should have (from [Webb 87]):

- tools for constructing a knowledge base of facts and rules (with an optional graphical interface)
- an inference mechanism which would allow the user to select the inference methods used (forward or backward chaining)
- the ability to obtain facts from a relational database
- explanation facilities, which would include a textual reasoning trace, a graphical reasoning trace, and further explanations as text or diagrams

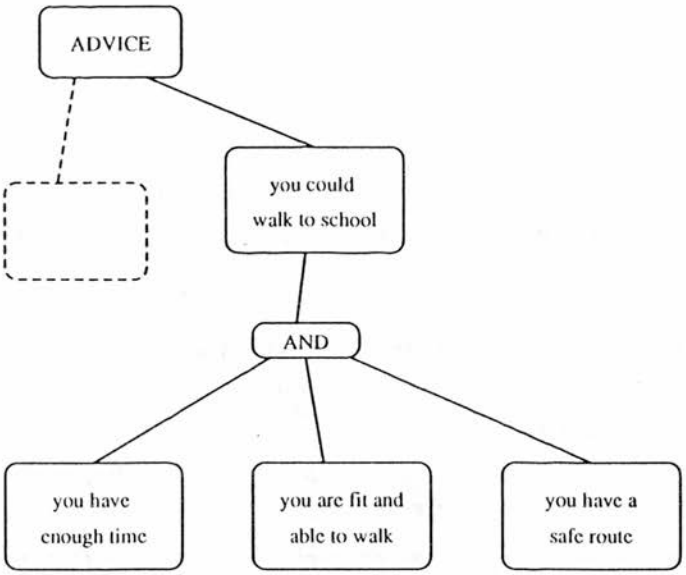
- an option for dealing with uncertainty
- tools for structuring a large knowledge base
- checking and help facilities
- the ability to supply advice, answer specific questions and/or respond to a general request depending on the nature of the knowledge base which has been built.
- the ability to browse textual rules, the graphical rule structure and explanations

The shell which resulted from this project is called Expert Builder, a rule-based shell which runs under Microsoft Windows. It is presented as 'an environment in which users can express and explore ideas and relationships', and claims to be 'designed for complete novices' [Exp89]. Although Expert Builder has some of the features listed above, there are many that it lacks, for example it cannot deal with uncertainty; it uses backward chaining unless the user 'volunteers' an answer, at which point it forward chains through the knowledge base as far as it can, before reverting to backward chaining again; it has no checking facilities and minimal on-line help facilities; and it has no way of obtaining facts from a relational database.

Expert Builder has two modes of use: Expert Builder Diagram and Expert Builder Text, and in both of these, a knowledge base can be built and consulted in the same environment. Expert Builder Diagram has facilities which enable a knowledge base to be constructed graphically, using boxes (which contain premises and conclusions) linked by arrows (representing dependencies). Figure 3-6 shows how a rule of the form:

IF you have enough time
AND you are fit and able to walk
AND you have a safe route
THEN you could walk to school

Figure 3–6: A rule represented in Expert Builder Diagram



could be represented in Expert Builder Diagram.

An Expert Builder knowledge base consists of many such links, and advice (for example 'you could walk to school' in Figure 3–6) is given to the user if the supporting evidence can be established. This is done by querying-the-user: questions are constructed by adding 'Is it true that...' to the front of the premise to be proved, such as 'Is it true that you have enough time'. Alternatively, the user can volunteer information at any point in the consultation. The diagram is also 'colour-coded', in that boxes become shaded if the facts in them have been established. Expert Builder Text has similar facilities to Expert Builder Diagram, the main difference being that rules are entered textually. Like that

of ADEX-Advisor, the Expert Builder KRL is variable-free, which greatly limits its expressiveness.

Although the idea of being able to represent a knowledge base diagrammatically may in theory be a good one, in practice, at least in Expert System Builder, it is less appealing. For anything more than one or two rules, diagrams quickly become complex and unwieldy, and it is very difficult to alter them once they have been constructed. However, this is partly due to Expert Builder's poor interface and overall slowness. For example, the shell has a 'zoom' facility which is supposed to give an overall picture of the reasoning tree: this is presented in a small corner of the screen, and is almost entirely useless, being nothing more than a birds-eye view of the entire tree. This would have to be vastly improved, for example along the lines of the AORTA diagrams used in the Transparent Prolog Machine [Eisenstadt & Brayshaw 88], before it would be a useful tool. There are also problems where a knowledge base can give more than one piece of advice, in that the diagram must be built with the advice boxes in chronological order from left to right. This is an overhead which, as we discussed above, users do not need or want. Informal discussions with users of Expert System Builder suggest that most quickly give up on its Diagram mode and use only its Text mode, although this also has problems: it has a line-based text-editor similar to that described above for Xi, and is slow and clumsy.

Although Expert Builder has itself no facilities for graphics, other than those used to build diagrams, it allows text and graphics to be pasted from other applications (such as the Windows application PAINT) using its 'clipboard' facility (the text or illustration is copied from the application on to the clipboard and then linked to a knowledge base). This allows more detailed instructions or information to be given during a consultation (in effect canned text) and allows the advice given to be illustrated with pictures and diagrams. Again, this is a good idea which in practice is unbearably slow.

In summary, Expert Builder demonstrates many features which are desirable in a an expert system shell for education, and although in their implementation

they are slow and often difficult to use, this does not detract from the overall soundness of the general principles upon which it is based.

EESS

EESS is an Educational Expert System Shell written in Prolog [Scherz *et al* 88a, Scherz *et al* 88b]. It was developed on and can run on small personal computers: although these have the disadvantage of certain speed and memory limitations, the developers of EESS rightly decided that a shell for use in education must be able to run on machines which will realistically be found in schools. EESS provides no KRL of its own, and instead requires the user to develop a knowledge base as an ordinary Prolog program. It provides templates with which the user can request an explanation, and which the system can use to present an answer.

The developers of EESS cite four potential uses of the shell in an educational setting. The first of these is as a debugging tool for students learning Prolog, although this is regarded as its most trivial use. Instead of the standard Prolog trace facility, EESS presents the user with a more 'understandable' translation of the Prolog proof tree. Using this, the user can debug the Prolog program currently being used as a knowledge base for the shell. The second use of the shell is as a tutor for a particular subject or skill. In this mode, the user can question the shell for answers and for the line of reasoning by which those answers were reached. Communication between the user and the shell is carried out using templates. The third use of the shell is where pupils learn about a domain by building a knowledge base for it, and then consulting that knowledge base using the shell. The fourth use of the shell is as an intelligent coach, where it presents a problem which the student must solve by showing all of the intermediate steps taken to reach the solution. During this task, the student can ask for help from the system in several different ways, such as asking for facts about the domain, or asking for an analysis of the problem-solving route taken.

It must be said at this point that there are only very sketchy details available about this shell. For example, there are no examples of the explanations provided

for debugging; no descriptions of the form and scope of the templates available for questioning the shell; and no details as to how the shell will be able to act as an intelligent tutor, with regard to the diagnostic or repair strategies which it will use.

From the information available, it is unclear as to how EESS could be used for knowledge base development by anyone other than an experienced Prolog programmer, since it has no higher-level KRL with which knowledge can be represented. This inevitably restricts its use as a tool throughout the curriculum, since most users will only be able to consult an EESS knowledge base, rather than building one of their own. In addition, although its developers claim that the interaction between the user and the system involves a natural language-like interface, this is really only question- and answer-templates. Since EESS is command-driven (as opposed to menu- or mouse-driven, for example), this means that the user must learn the exact format of each question-template before being able to interact with it. Finally, we agree with the developers of EESS that the provision of an intelligent tutor within an expert system shell would greatly enhance its educational potential. However, the absence of any details as to how exactly this will be achieved in EESS mean that we must remain sceptical about its tractability, particularly given that the shell must be usable on a small personal computer.

3.3 Summary and Conclusions

In this chapter we described the initial evaluation carried out by the researchers and by teachers at the three participating institutions. This evaluation was formative in nature and had three distinct parts:

- formal evaluation of the expert system shells used by the teachers. This involved questionnaires and semi-structured interviews. In addition, the interviews were also used to ascertain the teachers views on the use of expert systems shells in education.

- informal observation of discussions with teachers at each of the institutions while they were using the shells. The researchers also listened in on the many class discussions which took place. As before, these provided specific feedback on individual shells and general feedback on the educational use of shells.

- a review of existing shells by the author through personal use for both building and consulting knowledge bases and through the published literature. Shells which were reviewed in the early stages of the project helped contribute to the design of the new shell: those reviewed in the later stages provide a picture of the state-of-the-art in the field and served as a basis for comparison and evaluation.

In addition, a simple case-study of the Computer Studies department of a typical Scottish secondary school provided an interesting picture of the use of computers in schools, and the practical difficulties which will have to be overcome if expert system shells are to be successfully introduced into education.

The results of this evaluation have suggested some criteria which can be used as a basis for the design of the new shell. Overall, simplicity must be a key goal for all aspects of the shell, but in such a way that it is still fairly powerful, and can be used to develop reasonably large and non-trivial knowledge bases. In addition, the new shell should:

- have a knowledge representation language which is flexible and expressive enough to allow a variety of types of knowledge to be represented

- be designed such that the inferencing done by the system during a consultation does not affect the structure of a knowledge base, in that it should not be a factor which users need to consider when representing knowledge

- have facilities for both building and consulting a knowledge base, where the former should be highly structured such that knowledge can be entered easily

- provide good feedback about the knowledge base being developed, with regard to any errors and inconsistencies which it contains, and whether or not it is complete

- have the ability to deal with uncertainty

- have or provide access to graphics facilities, which can be used to illustrate questions asked of the user and the conclusions presented by the system

- be suitable for users from a range of age-groups, and more importantly for a range of abilities, both specifically in the area of computing and more generally in other areas

- be able to run on the kind of hardware which will be available in schools, which realistically means a small micro. Given the memory and power limitations of machines of this size, this restriction will impose many constraints on the facilities which the new shell can have

- be one which can be used throughout the curriculum. This carries with it the implication that it will be used as a tool to promote learning in those areas in which it is used. This is one of the shell's most important features, and is what will distinguish it from most of the other small, micro-based shells which are available

In subsequent chapters, we will describe the EXPLORES (EXPLanation ORiented Expert System) shell (the expert system shell which was developed as a result of this evaluation) and will discuss the extent to which it fulfills the above criteria.

Related work in this area has produced four shells designed for educational use: ADEX-Advisor, EGSPERT-Browser, Expert Builder and EESS. EESS requires that a knowledge base be developed in Prolog, and we have already discussed how this greatly limits the shell's use throughout the curriculum as a tool for both developing and consulting knowledge bases. The other three shells

all have a knowledge representation language based on production rules. For the reasons discussed in the previous chapter, we do not regard this as being a particularly expressive or flexible representation, particularly if the knowledge base is to be used for any purpose other then consultation. In addition, the KRLs of all three shells are variable-free, which again severely limits their expressive power. Only Expert Builder Diagram insists that the user take into account the inferencing which will be done during a consultation, by requiring that advice rules appear in chronological order.

Of the four only EESS does not claim to have facilities for building as well as consulting, but as we described above, in reality the facilities provided by the other three shells are fairly trivial and unhelpful. In particular, the graphical facilities provided by Expert Builder, although an innovative and desirable feature in a shell of this type, are poorly implemented and do little to make the task of developing a knowledge base easier. None of the shells provides feedback about the knowledge base being developed, or has the ability to deal with uncertainty.

ADEX-Advisor and EGSPERT-Browser are the easiest of the four shells to use, but are overall fairly limited. Expert Builder is more difficult for inexperienced users, and is the only one which claims to have been designed for use throughout the curriculum. Although this is true in that the shell could be used to build and consult a knowledge base in many subject areas, it assumes that these tasks are sufficient for the user to learn about the knowledge domain, and provides no other facilities to help promote such learning. EESS is of use as a building tool only by users who are also proficient Prolog programmers, and it is as yet unclear what facilities it has to promote domain learning.

In Chapter 2 we discussed the reasons why an expert system shell should provide a knowledge representation language: in the next chapter we describe the KRL developed for the EXPLORES shell. In later chapters we describe the other features of the shell which were developed to help promote the type of educational use described above; and the evaluation of the EXPLORES shell by teachers from one of the participating institutions.

Chapter 4

Representing Knowledge in the EXPLORES Shell

In Chapter 2 we described some of the knowledge representation formalisms upon which a knowledge representation language might be based in an expert system shell. In this chapter we describe the KRL developed for the EXPLORES shell, using examples from a simple knowledge base for plant classification. A subset of this knowledge base is then used to illustrate the strategy used by the shell to control inferencing during a consultation. Following this we discuss the limitations of the EXPLORES KRL, with regard to the kind of knowledge which can best be represented in it, and end with a summary and some conclusions regarding the EXPLORES KRL and the extent to which it fulfills the criteria on which it is said to be based.

4.1 Designing a KRL

There are many factors which have to be considered when designing a KRL for an expert system shell. In general, there is always a compromise between what diSessa calls the 'hacker bug', that is, providing a structure for every function and therefore having a language with a large number of primitives, and the 'formalism bug', that is, providing a sparse set of primitives out of which all functions can be built [diSessa 82]. These factors become particularly important in shells where

the knowledge base must be built outwith the shell using a standard text editor. There, users must learn the exact semantics and syntax of the KRL before being able to represent a body of knowledge, and the resulting knowledge base often has to be debugged several times, most commonly because of trivial syntax errors such as a missing comma or an extra colon. This makes the task of knowledge representation almost akin to programming, and in many cases it can become a time-consuming and frustrating exercise.

One way to alleviate this problem is to provide a means whereby a knowledge base can be built from within the shell. Such a facility can be built around the shell's KRL, and can therefore provide many aids to knowledge representation, such as templates for each knowledge structure which can be 'filled in' by the user. This means that the user need only be aware of the the knowledge structures available within the KRL, and what those structures can be used to represent, but need not be overly concerned with the exact syntax of the language. In addition, guidance can be given as a knowledge base is being developed about the knowledge already entered, in the form of menus from which, for example, variable names for use in rule premises can be selected. For these reasons, syntax and semantics errors occur much less frequently as a knowledge base is being developed. We return to this point in Chapter 6, when we describe the structure of the EXPLORES shell, and discuss its dedicated building environment.

Other factors which should be considered when designing a KRL are the purpose of the knowledge within the system (for example, will it be used for anything other than consultation) and the type of knowledge which will most easily be represented (such as hierarchical or causal knowledge), since no KRL can be suitable for representing all types of knowledge. Both of these factors will influence the choice of knowledge representation formalism upon which the KRL is based. Other uses to which the knowledge in a knowledge base can be put are discussed in more detail in the next chapter, when we look at the generation of explanations in expert systems and expert system shells. The type of knowledge which can be represented by particular formalisms is discussed both in Chapter

2 and below in a discussion on the limitations of the EXPLORES KRL (see Section 4.2.4).

In Chapter 3 we outlined several criteria upon which the design of the new shell should be based. We will restate here those which have influenced the development of the EXPLORES KRL. The first criterion states that the KRL should be flexible and expressive enough for a variety of types of knowledge to be represented. Others propose that the inferencing done by the system during a consultation should not be something which must be taken into account by the user when a knowledge base is being developed; and that the system should be able to handle representation of uncertainty. In addition, one of the main criterion for the shell as a whole was that it should be suitable for use throughout the curriculum as a tool which would help pupils learn about the domain of the knowledge base being consulted.

For these reasons the EXPLORES KRL is based on two main knowledge representation formalisms: production rules and frame-like structures called *things*. The use of these formalisms gives a language which can be used to represent any body of knowledge which is hierarchical or taxonomic in nature, or which can be made to fit such a structure. It allows users to represent their knowledge in a natural way, and as the strategy used by the inference engine to reason with that knowledge is implicit in the representation, it need not be considered when a knowledge base is being developed. Most importantly, the use of these two formalisms allows a separation of knowledge about the domain (declarative knowledge) and knowledge for running the consultation (procedural knowledge), such as we described for the CENTAUR system in Chapter 2. The benefit of such a separation here is that the shell can make use of that domain knowledge in its role as a tool to promote domain learning. We return to this point in the next chapter.

4.2 The EXPLORES KRL

The EXPLORES KRL is based on that of another expert system shell, CRESS. Thus, we must first describe the CRESS KRL before describing the ways in which it was adapted for use in the EXPLORES shell. Following our description of the EXPLORES KRL, we will describe the strategy used by EXPLORES to control inferencing during a consultation, and will end this section with a discussion of the benefits and limitations of the EXPLORES KRL.

4.2.1 The CRESS KRL

When the EXPLORES KRL was being developed, there was available another small expert system shell called CRESS (CRude Expert System Shell). [1] As CRESS was fully implemented and the time available for the development of the new shell was limited, we decided to make use of some of the features of CRESS, rather than implementing them again from the beginning. Thus, the EXPLORES KRL, inferencing strategies and query-the-user facilities, although not identical to, are based on those of CRESS.

CRESS is written in Edinburgh Prolog, and so the features which we used in the EXPLORES shell had to be translated into Arity Prolog (the Prolog dialect which we had available for the RML Nimbus). The CRESS KRL is based on *production rules* and two other knowledge structures, namely *questions* and *assignments*, and it can also deal with certainty factors. The full syntax of the CRESS KRL can be found in Appendix A.

---

[1] CRESS was designed and implemented by Dr Peter Ross, Department of Artificial Intelligence, University of Edinburgh. See [Ross 89] for a more complete description of it and for the full source code

Rules

Rules in the CRESS KRL have the form:

tag:
IF *premise*
THEN *conclusion*

The tag of a rule is its unique identifier, and each rule in a CRESS knowledge base must have a different tag: this allows it to be identified by name, for example when it is used in an explanation of the inferences made during a consultation.

The premise of a CRESS rule can be a single premise or two or more premises related by:

- conjunction (IF *premise1* AND *premise2* ...)
- disjunction (IF *premise1* OR *premise2* ...)
- a combination of these (IF *premise1* AND *premise2* OR *premise3* ...)

As always when conjunction and disjunction are combined, AND has a higher precedence than OR. The premises of a CRESS rule can contain the standard premise operators, such as =, < and >, for example:

IF A = B
IF A < B
IF A > B

The conclusion of a CRESS rule can be a single conclusion, or two or more conclusions related by conjunction (THEN *conclusion1* AND *conclusion2* AND ...) but not disjunction. The absence of disjunction is due to the fact that its use in the conclusion of a rule can lead to logical problems during inferencing. For example, if a rule existed of the form:

ruleN:
IF *A*
THEN *B*
OR *C*

and A were proved to be true, then a problem would arise as to which of the conclusions should be true. Should it be B, C or both B and C? This could have serious implications for the accuracy of the inferences made during a consultation, and ultimately the decisions reached by the system. Thus, multiple conclusions can only be linked conjunctively.

In addition to the standard operators used in conclusions (such as =), the CRESS KRL also has reserved words for operations such as clearing the screen and displaying text.

Questions

Questions in the CRESS KRL have the form:

question = *variable*

where the question part is used to query the user during a consultation, and the answer given by the user becomes the value for the named variable. For example, if a knowledge base contains a question of the form:

What day is it today? finds *day of week*

and during a consultation a value is needed for the variable *day of week*, the user will be asked the question *What day is it today?*, and the answer given will become the value of the variable called *day of week*.

Assignments

Assignments in the CRESS KRL have the form:

variable = *value*

Assignments are used to give a specific value to a named variable, and this value can then be displayed on the screen using the reserved word 'present' in the

conclusion of a rule. For example, if a knowledge base contained an assignment such as:

end_text = *'You are now at the end of the consultation'*

then the text could be displayed on the screen by placing a conclusion of the form

THEN ...
AND present end_text

in a rule designed to fire once the end of a consultation has been reached. When this happens, the value of the variable *end_text* will be displayed on the screen, that is, the text *You are now at the end of the consultation.*

The CRESS inferencing strategy

The inference engine of CRESS uses backward chaining to conclude facts from the available data. However, one slight augmentation to the method of backward chaining described in Chapter 2 is that in a CRESS knowledge base, a question which finds a value for a variable is used before any rules which conclude a value for it. For example, if the top level goal is X, CRESS will first search for a question which, when asked of the user, will provide a value for X. If one is found, it is put to the user and the answer given becomes the value of X. If there is no such question, CRESS will then gather all rules which conclude about X, for example:

rule1: **IF** *A* **THEN** *X*
rule4: **IF** *B* **THEN** *X*
rule8: **IF** *C* **AND** *D* **THEN** *X*

It will then find which of A, B, C and D can be proved, again using questions and rules as appropriate. This chaining continues until all of the relevant premises have been proved, at which time a value for X can be found by inferencing back through the chain. Inferencing can be restricted by declaring a variable to be single-valued. It can also be complicated slightly by the use of certainty factors, which can be attached to the rules in a CRESS knowledge base. CRESS

calculates a final certainly factor for a conclusion by combining certainty factors using a method similar to that used by MYCIN [Buchanan & Shortliffe 84], which is described in Chapter 2.

Although the structure and interface of EXPLORES are very different from those of CRESS, the EXPLORES KRL and inference engine are based on those used in CRESS. However, in addition to the three knowledge structures of the CRESS KRL, that is, *rules, questions* and *assignments*, EXPLORES also has a knowledge structure called *things*. We will now describe this aspect of the EXPLORES KRL in more detail, using examples from a knowledge base about plants built using the shell. The full knowledge base can be found in Appendix B.

4.2.2 Things in the EXPLORES KRL

The EXPLORES KRL has four main knowledge structures: *rules*, *questions*, *assignments* and *things*. The structure of *rules, questions* and *assignments* in the EXPLORES KRL is as described above for the CRESS KRL. The fourth knowledge structure, *things*, provide a way in which domain knowledge can be represented as objects or classes of objects, where the properties of these objects and their relationships to other objects in the knowledge base are described within the knowledge structure. They can be thought of as a kind of structured object (see Chapter 2), in that they are a simplified version of the frame structures described by Minsky [Minsky 75]. By simplified, we mean that the slots and fillers found in standard frames are here restricted to properties with a name and some specific values if required: they cannot contain rules or be frames in themselves. In addition, the possible relationships and subsequently the inheritance of properties between things is also restricted. In an EXPLORES knowledge base *rules* and *questions* are used to find values for the properties of *things* during a consultation.

CRESS is one of the many expert system shells which require that a knowledge base be built outwith the shell, using a text editor. Thus, its KRL is a specific language with reserved words and a fairly rigid syntax (for example the

variable name *day of week* is permitted, but *day of the week* is not). A CRESS knowledge base is checked as it is being loaded, and any syntax and semantics errors found are reported to the user [2] . EXPLORES, on the other hand, provides an interactive building environment which is built around its KRL, and where for each knowledge item, prompts are given for the part of the structure being added (for example by using menus containing all of the possible options for the next part of the structure). The entered knowledge is translated and stored as Prolog clauses, and if this stored knowledge has to be displayed on the screen at any point, it is translated into an appropriate format. This means that, unless the user is willing to represent his knowledge using Prolog clauses, it is not possible to build a knowledge base for EXPLORES using a text editor. This option is considered as a possible extension to the shell and is discussed in detail in a Chapter 8. Note that this only applies to the *things* in a knowledge base: most of the reserved words and operators for *rules, questions* and *assignments* from the original CRESS KRL have been retained. However, an EXPLORES knowledge base must contain *things*, and so currently the restriction of using the building environment always applies.

In the EXPLORES KRL, *things* are structures which can be used to describe objects and classes of objects in a particular domain. These are described in terms of their properties, such as the features which they have and the actions which they perform, as well as their relationships to other *things* in the knowledge base. Each *thing* in a knowledge base should have a unique name, and they should all be related in such a way as to form a hierarchy, with one *thing* at the top (an EXPLORES knowledge base can only contain one hierarchy). A *thing* can inherit properties from other *things* which are above it and on the same branch of the hierarchy. Properties can be inherited either fully or partially, and additionally a *thing* can have unique properties of its own. Some examples of *things* can be

[2]The procedure for this is described in more detail in Chapter 6, with regard to the changes made to it for use in the EXPLORES knowledge checking tool

Figure 4–1: Things in the EXPLORES KRL

NAME:	*plants*
Has:	*stem protection = thorns, spines or hair*
	milky sap in stem = yes or no
Relations:	**has member** ⇒ *spiny cactus*
	has member ⇒ *hairy cactus*
NAME:	*spiny cactus*
Has:	*stem protection = spines*
	milky sap in stem = no
Relations:	**has member** ⇒ *column cactus*
	has member ⇒ *goats horn cactus*
NAME:	*hairy cactus*
Has:	*stem protection = hair*
	milky sap in stem = no
Relations:	**has member** ⇒ *old man cactus*
NAME:	*column cactus*
Has:	*spine type = brown*
	ribs = yes
Does:	*flowers in July = yes*
NAME:	*goats horn cactus*
Has:	*spine type = curved*
	ribs = yes
Does:	*flowers in May = yes*

seen in Figure 4–1. These examples illustrate several aspects of the structure of *things* within the EXPLORES KRL:

Features All five *things* in Figure 4–1 are described in terms of features that they have and possible values for those features (listed under **Has:** in the examples). Thus, the *thing* called *plants* has the features *stem protection* and *milky sap in stem*: possible values for *stem protection* are thorns, spines and hair, and possible values for *milky sap in stem* are no and yes. This means that a member of the class of objects called *plants* could have any combination of these two features, for example *stem protection* with the value spines and *milky sap in stem* with the value no, or *stem protection* with the value hair and *milky sap in stem* with the value yes. It is possible to make the values that a feature can have restricted, semi-restricted or unrestricted. This is discussed in detail below.

Actions It is also possible to describe a *thing* in terms of actions that it performs: this can be seen under **Does:** in two of the examples in Figure 4–1 (*column cactus* and *goats horn cactus*). An action is something which is done or not done by the thing, that is, it can have the value yes or no. However, it is also possible to have an action which is only usually or occasionally done: these restrictions are discussed in detail below.

Full inheritance *column cactus* has two unique properties of its own: *spine type* and *ribs*. In addition, it fully inherits *stem protection* and *milky sap in stem* from *spiny cactus*, that is, it has those properties with the same values as *spiny cactus*. This happens automatically where there is a **has member** or **is an example of** relationship between two things (see below). Thus, full inheritance of properties is implicit, and need not be represented in a knowledge base.

Partial inheritance *spiny cactus* has no unique properties of its own. However, it partially inherits *stem protection* and *milky sap in stem* from *plants*. This is only partial inheritance because although it has the same property, it

does not have it with exactly the same value(s). Partial inheritance must be made explicit.

Reciprocal relationships there is no need to state both sides of a relationship between two *things*. For example, it is enough to say that *plants* has the relationship **has member** with *spiny cactus*: there is no need to also state that *spiny cactus* has the relationship **is an example of** with *plants*. The EXPLORES KRL offers several possible relationships between the *things* in a knowledge base: these are discussed in detail below.

Figure 4–1 also shows the way in which *things* in a knowledge base are displayed on the screen when a knowledge base is being viewed or edited: this output is a translation of the Prolog clauses stored as the knowledge base.

We will now discuss the properties of *things* and the relationships between *things* in more detail.

Features of things

As Figure 4–1 shows, *things* can be described by the features which they have, and the EXPLORES KRL provides the option of stating specific values for these features. In the examples in Figure 4–1, *plants* has the feature *milky sap in stem* with *yes* and *no* as two possible values. A feature has three possible states:

no stated values This means that during a consultation, the feature can be given any value.

suggested values This means that the knowledge base developer has suggested values for the feature, but that other values may be possible. Although during a consultation the suggested values will be shown, again, the feature can be given any value.

specific values This means that during a consultation, the feature can have the stated values and no other.

When adding the features of a *thing* to a knowledge base using the building environment, the user is always asked about specific values for each feature, and whether or not any other values can be accepted during a consultation.

In Chapter 2 we discussed some criticisms of frames as a representation formalism. One of these was the problem of representing default values when inherited properties can be overwritten further down the hierarchy, and consequently of representing objects which are unrepresentative of their class, for example, birds which cannot fly [Brachman 85]. We also described how some frame-based systems have attempted to circumvent the problem, for example by representing 'essential' and accidental properties. In the EXPLORES KRL, inherited properties cannot be cancelled or overridden, and the possibility of there being other possible values for a feature can be acknowledged as described above, for example, by having a feature (such as *flies*) which has a *suggested* value of yes, indicating that it can be instantiated to another value (for example no) during a consultation.

Actions things perform

As we also mentioned above, *things* can be described in terms of what they do, such as *flowers in winter* for some plants. There are five possible states for an action:

always done This is equivalent to having yes as a specific value, that is, only *yes* will be accepted as a value for the action.

usually done This is equivalent to having yes as a suggested value, that is, either *yes* or *no* will be accepted as a value for the action.

occasionally done This is equivalent to having yes and no as suggested values, that is, either *yes* or *no* will be accepted as a value for the action.

usually not done This is equivalent to having no as a suggested value, that is, either *yes* or *no* will be accepted as a value for the action.

never done This is equivalent to having no as a specific value, that is, only *no* will be accepted as a value for the action.

Again, the user is prompted for these states automatically when adding actions for *things* to a knowledge base.

Relationships to other things

There are several ways in which a *thing* can be related to other *things* in the knowledge base. These are as follows:

- *Thing1* **has member** *Thing2* that is, Thing1 is a group or class of *things* of which Thing2 is a member. Some examples of this relationship are:
  - hairy cactus **has member** old man cactus
  - hairy plants **has member** hairy cactus
- *Thing1* **is an example of** *Thing2* that is, Thing1 is an example of the group or class of *things* described by Thing2. This is, of course, the reverse of the **has member** relationship. Some examples of this relationship are:
  - old man cactus **is an example of** hairy cactus
  - hairy cactus **is an example of** hairy plants
- *Thing1* **is part of** *Thing2* that is, Thing1 is a component of Thing2. Some examples of this relationship are:
  - thorns **is part of** thorny plants
  - spines **is part of** spiny plants
- *Thing1* **has part** *Thing2* that is, Thing2 is a component of Thing1. This is the reverse of the **is part of** relationship. Some examples of this relationship are:
  - thorny plants **has part** thorns
  - spiny plants **has part** spines

- *Thing1* **does** *action* **to** *Thing2* (known as a **does to** relationship). An example of this relationship is:

  - greenfly **does** eats **to** plants

- *Thing1* **has** *action* **done by** *Thing2* (known as a **done by** relationship). This is the reverse of the **does to** relationship). An example of this relationship is:

  - plants **has** eats **done by** greenfly

- *Thing1* **relationship** *Thing2* known as an **other** relationship, this is a catch-all category which allows you to define any relationship between two *things*. An example of this relationship is:

  - column cactus **is similar to** old man cactus

The **done by, does to** and **other** relationships are only used by the explanation tool.

An EXPLORES knowledge base must contain a hierarchy of *things*, linked by one type of relationship, for example, a hierarchy of plants linked by **has member/is an example of** relationships or a hierarchy of car engine components linked by **has part/is part of** relationships.

In Chapter 2 we described some of the problems with frame representations, including the many possible interpretations of the use of an ISA relationship between two elements of a hierarchy or network [Brachman 83]. In the EXPLORES KRL, the use of ISA is avoided as much as possible, and the more specific 'is an example of' is used instead. Although this makes the meaning slightly clearer, it still allows no distinction between generic/generic relationships such as 'spiny cactus is an example of spiny plants' and generic/individual relationships such as 'column cactus is an example of spiny cactus'. In an EXPLORES knowledge base, the distinction is apparent in the structure of the hierarchy, in that the leaf nodes represent instances while other nodes represent generalisations. However, the logical problems described in Chapter 2 are still possible. Although there

is no obvious solution to this problem, there are several ways in which teachers could be helped to explain the difference between these possible interpretations to pupils using the shell (for example, by providing them with appropriate teaching and support materials). We return to this point in the final chapter of the thesis, in the section on further work.

We will now go on to discuss the EXPLORES inference engine, in terms of the control strategy it uses to reason with the knowledge in a knowledge base, represented using the EXPLORES KRL.

4.2.3 The EXPLORES inferencing strategy

An expert system shell must have a strategy for reasoning with the knowledge in a knowledge base in order to produce a result during a consultation. The inferencing strategy used by the shell is determined by the knowledge structures used in its knowledge representation language. As stated above, the KRL for the EXPLORES shell is based on four structures: *rules, questions, assignments* and *things*. Of these, assignments are the only structure not used during the inferencing process.

The strategy used during an EXPLORES consultation takes the hierarchy of *things* in a knowledge base, and, using *rules* and/or *questions* in the way described above for the CRESS inferencing strategy (Section 4.2.1), finds values for the actions and features of each *thing* until it can present its conclusions. These will be the names of one or more *things* whose properties are consistent with the information gathered during the consultation. An EXPLORES knowledge base *must* contain *things* and *questions*: the use of *rules* and *assignments* is optional.

Before giving the algorithm for the control strategy used by the EXPLORES inference engine, we will explain some of the terminology used. Each *thing* in a knowledge base has properties (features and actions) associated with it, and each property may have specific values which it must take (one of the possible states described above in Section 4.2.2). These are referred to below as **static values**. During a consultation, the inference engine tries to find a value for each property

in the knowledge base by asking the user to provide a value for it or by using the rules in the knowledge base to conclude a value for it. The values found during a consultation are referred to below as **dynamic values**. The static values of a property are said to be **inconsistent** if none of them matches the dynamic value found during a consultation, and are **consistent** if one of the values is the same as that found. If a *thing* has a property which is inconsistent, then the *thing* is **disregarded**, that is, it is no longer considered during the consultation. A *thing* is said to be **fully-instantiated** if a dynamic value has been found for all of its properties.

The control strategy used by EXPLORES during a consultation begins with the *thing* at the top of the hierarchy, and goes through the following steps:

1. For each property for which a dynamic value has not been established, try to find a dynamic value for it by:

   (a) asking the user, if there is a question which finds a value for that property.

   (b) firing all the rules which conclude a value for that property (using backward chaining, as described above for the CRESS inference engine). The truth of a rule's premise is established by following steps 1(a) and 1(b) as often as necessary. The final step in the reasoning chain must always be to ask the user.

2. Once a dynamic value has been found for all of the properties at the current level, move down to the next level of the hierarchy.

3. Disregard all *things* which have properties whose static values are inconsistent with the dynamic values found.

4. For the remaining things, find any properties for which dynamic values have not been established, and begin again at step 1.

A consultation ends when either:

- the bottom of the hierarchy is reached, there are one or more *things* with properties whose static values are consistent with the dynamic values found, and there are no more properties for which dynamic values need to be found. In this case, the *things* with fully-instantiated properties are presented as the result of the consultation.

- the bottom of the hierarchy is reached and there are no *things* which have properties whose static values are consistent with the dynamic values found. In this case, there are no fully-instantiated *things* in the knowledge base, and the system cannot produce a result for the consultation.

An example consultation

The EXPLORES control strategy is best illustrated by working through an example consultation using the knowledge base shown in Figures 4–2 and 4–3. This is a simplified version of the plants knowledge base described in Appendix B. (Note that the simplified knowledge base is incomplete, as the examples of *spiny cactus* are not shown). The steps in this consultation would be as follows:

1. Start with the *thing* called *spiny plants*, i.e. the *thing* at the top of the hierarchy.

2. Find a dynamic value for *stem protection* by asking the user *What kind of stem protection does your plant have?*. Assume user answers *spines*.

3. There are no outstanding properties for *spiny plants*, so move down to the next level of the hierarchy, to the examples of *spiny plants*, namely *spiny cactus* and *spiny euphorbia*. The properties of both *things* are consistent with the dynamic values found so far, so find dynamic values for any outstanding properties.

Figure 4–2: A simple knowledge base: things

NAME: *spiny plants*
Has: *stem protection = spines*
Relations: has member ⇒ *spiny cactus*
has member ⇒ *spiny euphorbia*

NAME: *spiny cactus*
Has: *milky sap in stem = no*
Relations: has member ⇒ *column cactus*
has member ⇒ *goats horn cactus*
has member ⇒ *peanut cactus*

NAME: *spiny euphorbia*
Has: *milky sap in stem = yes*
Relations: has member ⇒ *cactus spurge*
has member ⇒ *resinifera*

NAME: *cactus spurge*
Has: *spine type = long spines on branches*

NAME: *resinifera*
Has: *spine type = short spines on ribs*

Figure 4–3: A simple knowledge base: rules and questions

Question: *What kind of stem protection does your plant have*
finds a value for
Item: *stem protection*

Question: *Does your plant have milky sap in its stem*
finds a value for
Item: *milky sap in stem*

Question: *What type of stem does your plant have*
finds a value for
Item: *stem type*

Question: *What length of spines does your plant have*
finds a value for
Item: *spine length*

rule1:
IF *stem type = branched*
AND *spine length = long*
THEN *spine type = long spines on branches*

rule2:
IF *stem type = ribbed*
AND *spine length = short*
THEN *spine type = short spines on ribs*

4. Find a dynamic value for *milky sap in stem* by asking the user *Does your plant have milky sap in its stem?*. Assume user answers *yes*.

5. Disregard *spiny cactus* because its properties are inconsistent (*milky sap in stem* = *no*). Keep *spiny euphorbia*.

6. There are no outstanding properties for *spiny euphorbia* , so move down to the next level of the hierarchy, to the examples of *spiny euphorbia*, namely *cactus spurge* and *resinifera*. The properties of both *things* are consistent with the dynamic values found so far, so find dynamic values for outstanding properties.

7. Try to find a dynamic value for *spine type*. This cannot be done by asking the user, as there are no appropriate questions. Therefore, find all of the rules which conclude a value for *spine type* (*rule1* and *rule2*). A rule's conclusion is true if its premises are true, so try to establish which of the premises of *rule1* and *rule2* are true by finding a dynamic value for *stem type* and for *spine length*.

8. Try to find a dynamic value for *stem type* by asking the user *What type of stem does your plant have?*. Assume user answers *ribbed*.

9. Try to find a dynamic value for *spine length* by asking the user *What length of spines does your plant have?*. Assume user answers *short*.

10. Based on the answers given, *rule1* fails and *rule2* succeeds. Therefore, *spine type* takes the dynamic value *short spines on ribs*.

11. Disregard *cactus spurge* because its properties are inconsistent (*spine type* = *long spines on branches*).

12. *resinifera* has consistent properties, the bottom of the hierarchy has been reached, and there are no outstanding properties.

13. *resinifera* is the result of the consultation.

This is only one of several inferencing strategies which could be used on a hierarchical knowledge base. It uses a top-down, breadth-first method of traversing the hierarchy: that is, it begins with the root node and proceeds to the leaf nodes, analysing each level in full before moving down to the next. Many other search strategies would also be possible, such as depth-first search (where each branch of the hierarchy is searched in turn). One of the measures of the efficiency of a search strategy is that it visits fewer nodes to find a solution in comparison with other strategies. The structure of an EXPLORES hierarchy is such that there may be more than one solution, and thus all contending branches must be searched until a solution is found or it becomes apparent that one will not be found there. Thus, there would be nothing to be gained by using any other strategy: the same number of nodes would still have to be searched.

Another strategy often used to make classification more efficient is **query optimization**. This tries to ensure that a solution is found by asking the fewest number of questions possible, using heuristics such as 'ask about the most common property first' or 'ask about the least common property first'. Although there are many situations where such heuristics are valuable, in the EXPLORES shell they would do nothing to increase the efficiency of the inferencing algorithm. This can be illustrated using an example of two things which have all but one property in common: one has a ribbed stem and the other does not. Although establishing a dynamic value for 'ribbed stem' before all other properties would make it immediately obvious which of the two plants was not the solution, it would do little to confirm which was the solution. The system would still have to confirm dynamic values for all of the other properties, because the knowledge base may not contain a complete representation of the domain, and as such may not in fact contain a solution.

We will now discuss the potential benefits of a representation such as that used in the EXPLORES KRL, as well as its limitations with regard to the kind of knowledge which it can be used to represent.

4.2.4 Benefits and limitations

With a KRL based on production rules only, the knowledge base must contain terminating rules which conclude an answer to the consultation. This is not necessary in a KRL based on frames and production rules, since the information contained in a terminating rule is implicit in the representation of a bottom-level *thing* (the most specific instance of a class). For example, in Figure 4–1, we had the *thing*

NAME:	*goats horn cactus*
Has:	*spine type = curved*
	ribs = yes
Does:	*flowers in may = yes*

In addition, *goats horn cactus* also had the inherited properties *stem protection* with the value spines, and *milky sap in stem* with the value no. Thus, an EXPLORES knowledge base need only contain rules or questions to find values for all five of these properties.

If shell's KRL contained only production rules, then to represent the same knowledge it would be necessary to have a rule of the form:

IF *spine type = curved*
AND *ribs = yes*
AND flowers in may = yes
AND *stem protection = spines*
AND *milky sap in stem = no*
THEN *plant = goats horn cactus*

for every plant in the knowledge base. We would argue that for a large knowledge base, this type of representation is unnecessarily repetitive and ultimately time-consuming for the user, and that an object based representation where these rules are implicit in the structure of the object is a more intuitive and efficient representation.

Some rule-based shells (for example ADEX-Advisor [Briggs 87], which we described in Chapter 3) differentiate between top-level **advice** rules, which is how the one above would be classified, and **other** rules. Other rules do the

background work during a consultation, to find values for the premises of the advice rules. An example of such a rule would be:

IF *stem type = branched*
AND *spine length = long*
THEN *spine type = long spines on branches*

It is sometimes claimed that this distinction preserves the domain knowledge, in that it need not become compiled into the premises of a rule, but we would argue that this distinction is very much in the hands of the knowledge engineer. For example, the above rule appears to capture the essential properties of the plant *goats horn cactus* in much the same way as the *things* of the EXPLORES KRL. However, this knowledge could have been represented as many small rules of both kinds, and thus the explicit domain knowledge would have been lost. Even if care is taken to ensure that this does not happen, there still remain the problems of unnecessary repetition of premises, and the lack of any obvious relationship between the possible conclusions.

One of the criterion which we established during the initial evaluation was that the inferencing done by the system during a consultation should not be something which the user need consider when developing a knowledge base (for example, ensuring that rules appear in the order in which they will fire). The structure of the EXPLORES KRL and the algorithm used by its inference engine ensure that this is the case: the user need only describe objects in the domain in terms of their properties and their relationships, and provide a means by which values can be found for these properties during a consultation. (In addition, the user is helped in these tasks by the dedicated building environment and the knowledge checking tool: these are described in Chapter 6). This is possible because the inferencing algorithm exploits the inherent, hierarchical structure of a knowledge base, and thus the strategy used is implicit in the KRL. In addition, the algorithm ensures that the user does not have to learn a query language to consult a knowledge base, something which can be a problem in other frame-based shells (for example with FLEX, retrieval of information from the frames in a knowledge base can be very difficult).

The main benefit of a representation such as this is one which we will return to in the next chapter on explanation generation in expert systems and expert system shells. In Chapter 2 we discussed the distinction between procedural and declarative knowledge, and deep and surface representations. The EXPLORES KRL provides a means of explicitly representing a hierarchical model of the objects in a domain, their properties, and the relationships between them, and this declarative knowledge (represented using *things*) is separate from procedural knowledge for reasoning with it during a consultation (represented using *rules* and *questions*). In Chapter 3 we discussed how ideally the main use of the EXPLORES shell in an educational environment would be across a range of subjects, as a tool which could help the user learn about the domain of the knowledge base being consulted. For this to be possible, there must be an explicit representation of that domain in the knowledge base, and this is provided by the kind of representation described in this chapter. In the next chapter we show how the EXPLORES explanation tool makes use of this domain model to help promote the kind of learning described here.

The EXPLORES KRL is well suited to knowledge which is inherently hierarchical in structure, for example a biological taxonomy. In addition, there are other bodies of knowledge, which, although not naturally hierarchical, can be made to fit that kind of structure. However, there are many kinds of knowledge which cannot easily be represented using this type of formalism, such as causal knowledge (for example, the cause and effect relationships between the components in an electrical circuit). This is not regarded as a major disadvantage: it is unrealistic to expect to be able to represent all types of knowledge in any one KRL, regardless of the formalisms upon which it is based. Some restrictions must be imposed, and given the proposed use of the EXPLORES shell, it was decided to limit the representation to knowledge which could be represented in a hierarchical way.

4.3 Summary and Conclusions

In this chapter we described in detail the KRL designed for the EXPLORES shell, which is an extension of that provided by CRESS. The EXPLORES KRL has four knowledge structures: *rules*, *questions*, *assignments* and *things*, and an inference engine which combines backward chaining (using rules); query-the-user (using questions); and property instantiation (using things). The conclusions reached during a consultation are based on the static knowledge contained in the knowledge base and the dynamic knowledge which is both provided by the user at run-time and inferred from the rules in a knowledge base.

The EXPLORES KRL can be used to represent knowledge which fits naturally into a hierarchical or taxonomic structure or which can be made to fit such a representation. It cannot be used to represent causal knowledge.

Fikes and Kehler [Fikes & Kehler 85] list three criteria for a knowledge representation language: these are *expressive power* (can expert knowledge be represented effectively); *understandability*, (can experts understand what the system knows); and *accessibility* (the ease with which the system can use the knowledge that it has). Even with the limitations mentioned above, we would propose that a representation based on frames and production rules, as used in EXPLORES, comes closer to fulfilling all three of these criteria than a representation based solely in production rules.

The formalisms upon which the EXPLORES KRL is based allow a separation of domain knowledge (as represented using *things*) and control knowledge (as represented using *rules* and *questions*). As yet, we have said little about why such a separation might be desirable, other than to state that it reduces the need for terminating rules in a knowledge base, since these are implicit in the representation. In the next chapter, we discuss the generation of explanations in expert systems and expert system shells, in terms of an explanation framework. Within this framework, we demonstrate how the content of an explanation depends on, among other things, an appropriate knowledge representation formalism. We

then discuss in more detail the advantages of the EXPLORES KRL with regard to the provision of an explanation facility within the EXPLORES shell, which provides the means whereby the user can explore and learn about the domain of a knowledge base.

Chapter 5

Explanation in Expert Systems and Expert System Shells

In Chapter 3 we questioned why expert system shells should be used in classrooms, that is, what could we expect pupils to learn by using them. We concluded that if shells are confined to the Computing Science curriculum and presented to pupils as merely another piece of computer software (that is, they are in effect used to teach pupils about expert system shells) then existing shells are probably adequate for this purpose. However, we argued that this does not use shells to their full potential, and that the right shell could be used to help pupils learn about a variety of domains in a number of subject areas throughout the curriculum. One way of achieving this goal is to structure the shell such that pupils are given access to the domain knowledge contained in a knowledge base.

For this reason, the main focus of the research described in this thesis became the development of an explanation facility within an expert system shell, which would allow the user to ask questions about the domain of any knowledge base, and in response would be able to generate explanations using the domain knowledge represented there. In the previous chapter we described the KRL developed for the EXPLORES shell, which has has four knowledge structures (*rules, questions, assignments* and *things*). We also discussed the kind of knowledge which can best be represented in this KRL, and mentioned some of the advantages that a KRL which combines frames and production rules has over one which is based

solely on production rules. One of these was that it allows an explicit representation of the domain knowledge. In this chapter we will demonstrate how that can be used to good effect within an explanation facility.

Although many techniques have been developed for the generation of explanations from within expert systems, the explanation facilities of most expert system shells have remained basic. This can partly be explained by the fact that many of the techniques used in expert systems are highly domain-dependent, and as such cannot be used in an environment which must be able to deal with knowledge bases from a number of domains.

We begin this chapter by analysing the explanation facilities of a number of expert systems. This is done by constructing a framework which makes explicit the techniques used and the factors which can affect the type, structure and content of the generated explanation. We then move on to discuss the limited explanation facilities found in expert system shells, in terms of the framework constructed previously. Based on these analyses, we discuss the requirements for an explanation facility within an expert system shell intended for educational use, and then go on to describe the explanation facility developed as part of the EXPLORES shell. We finish with a summary and some conclusions regarding the function of explanation in an educational shell.

5.1 Explanation in Expert Systems

In recent years research on expert systems has focused not only on producing systems which perform quickly, efficiently and correctly, but which also have the ability to explain aspects of their performance to the user. There have been several different techniques developed for the generation of such explanations. An analysis of some of these shows that among the factors which affect the type, structure and content of a generated explanation are:

- the target of the explanation, that is who it is aimed at

- the **purpose** of providing the explanation
- the **type** of explanation required
- the **knowledge needed** to provide an explanation of the desired type
- the **interaction** between the user and the explanation facility, that is the ways in which the user can request an explanation, and in which the content of the generated explanation can be presented by the system.

In this section we will review some of the techniques used to generate explanations in expert systems, in terms of this framework. To do this, we will discuss each of the above factors in detail, illustrating the points made by describing one or more systems which have an explanation facility as one of their major parts.

5.1.1 Target and purpose

The target of any explanation generated by an expert system will almost always be someone who is consulting the system. However, that general description can cover a wide variety of people, and there will be a great deal of variance in the level of knowledge that each person will have about the domain, and in their reasons for requesting the explanation. Some examples of the possible targets for a generated explanation could be: someone who knows little or nothing about the knowledge domain; the expert from whom the knowledge in the system was acquired; or the knowledge engineer responsible for developing the system.

Potentially, each of these users will require the explanation for a different purpose: for example, the domain novice may wish to find out something about the domain of the knowledge in the system's knowledge base; the domain expert may want to ensure that the system contains a correct and accurate representation of his knowledge; and the knowledge engineer may wish to ensure that the system is reasoning accurately with the knowledge that it has.

Buchanan and Shortliffe [Buchanan & Shortliffe 84] propose five reasons why an expert system should be able to explain its performance. These are:

Understanding An explanation can help to give the user an understanding of the contents of a knowledge base, and of the line of reasoning used by the system to reach a conclusion during a consultation. This is regarded by Buchanan and Shortliffe as being the major goal of most of the work on explanation.

Debugging An explanation can also be useful for debugging a system, in that it provides a way to check that the knowledge base is complete and consistent, and to ensure that the system is using the knowledge which it has in an accurate way.

Education Educating the user is also listed as a reason for having an expert system which is able to explain its performance. Although linked to understanding, this is specifically aimed at a domain novice, with regard to educating them about the domain of the knowledge base being consulted. As the educational use of explanation is one of the topics of the research described in this thesis, we will discuss this point in more detail below.

Acceptance Acceptance is described as making a system acceptable to potential users by convincing them that its conclusions are reasonable. Again, an explanation of the line of reasoning and the knowledge used can help achieve this.

Persuasion Although listed as a separate reason by Buchanan and Shortliffe, persuasion is closely linked to acceptance: in fact, it is difficult to pinpoint a major difference between the two. They describe it as using an explanation to persuade users that the system's reasoning is correct, which seems to be the same as making the system acceptable to the user.

Although not all research on the generation of explanations cites all of these as its motivating reasons, many researchers generally propose at least a subset of

them (for example, see [Swartout 81, Aikins 83]). However, an explanation which is suitable for educating a domain-novice should be very different with regard to its content and structure from one intended to help a knowledge engineer to debug the system. This point is often overlooked by system developers, and the same multi-purpose explanation generated regardless. We would propose that the intended target of an explanation and the purpose of providing it should be a major consideration when deciding on the type, structure and content of the explanation to be generated. We will now discuss these factors further.

5.1.2 Type

We can identify two main types of explanation which can be generated by an expert system: **system-based** explanations and **domain-based** explanations. Examples of system-based explanations can be seen in many expert systems, such as MYCIN [Buchanan & Shortliffe 84], where the information given within the explanation outlines what has happened during the preceding consultation. This could be information about which rules have fired, which facts have been deduced, and what the steps were in these deductions. To generate a system-based explanation, a trace must be kept throughout the consultation of everything that has happened so far. This can then be retrieved, translated and presented as an explanation of the line of reasoning taken by the system.

A domain-based explanation is one which contains information about the represented domain knowledge. Some systems such as XPLAIN [Swartout 81] and CENTAUR [Aikins 83] use domain-based explanation to **justify** system-based explanations: that is, they attempt to explain not only *what* the system is doing but *why* it is doing it. Generation of domain-based explanations requires an explicit representation of the domain knowledge in a knowledge base: we return to this point in Section 5.1.3.

An explanation can also be categorised by whether or not it is tailored to the user in any way, such as taking into account his level of expertise in the domain, his current goals in the consultation, or his current requirements (for

example, there are occasions where a short summarised explanation may be preferred over a longer, more detailed one, and vice versa). Some systems always provide the same explanation regardless of external circumstances (for example MYCIN), but many systems have attempted to tailor their explanations to individual users (for example UMFE [Sleeman 85], ADVISOR [McKeown 88], GUMAC [Kass & Finin 88] and a prototype explanation system designed by Wallis and Shortliffe [Wallis & Shortliffe 84]). The explanations generated by these systems and the techniques used for doing so are discussed in detail in the next section, where we look at the knowledge needed to generate an explanation of the required type.

5.1.3 The knowledge needed

All expert systems have a knowledge base containing the knowledge for the particular domain to which they are dedicated. Within this knowledge base, there will be knowledge about the actual domain (declarative knowledge) as well as problem-solving knowledge for running the consultation (procedural knowledge). Both types of knowledge will be represented using a representation formalism such as production rules, frames, or a semantic network (some systems use more than one formalism). This was discussed in Chapter 2 with regard to knowledge representation and expert system shells.

Many early expert systems used production rules for knowledge representation, but as it became more important that a system should be able to generate coherent and informative explanations, it soon became clear that production rules alone were insufficient. If a system is to give a fuller, more complete explanation of what it is doing during a consultation, with regard to the problem solving-strategies used, then the function of the rules in the system must be represented, and there must also be an explicit representation of support knowledge (low level knowledge used for rule justification). If explanations are to be justified, that is, if systems are to explain what they are doing and why they are doing it, then the domain knowledge must be accessible to the system. This is

not the case if the system uses a representation based solely on production rules, which, as we described in Chapter 2, often contain only a compiled version of the domain knowledge. This compiled knowledge is represented in rules alongside problem-solving knowledge, and no distinction is made between the two: thus, there is no explicit representation of the domain available to the system. If an explanation is to be tailored in any way, then the system must additionally have knowledge about the user. We will now discuss these points more fully with reference to the explanation facilities of some existing expert systems.

Originally, the knowledge in MYCIN was represented as a set of production rules written in LISP. When an explanation was requested the system's explanation facility translated the current rule into English and displayed it. This translation constituted an explanation of what the system was currently doing. This facility was soon expanded by the inclusion of a Reasoning Status Checker (RSC) [Scott *et al* 84], which recorded the entire reasoning chain of the system during a consultation. This trace was then used to answer the questions WHY (entered by the user in response to a question from the system and interpreted as 'Why do you want to know that?'), and HOW (entered in response to a presented conclusion, and interpreted as 'How did you reach that conclusion?'). To answer WHY, the RSC presented the user with the reasoning chain being used to establish the top-level goal, and to answer HOW, a descent of this reasoning chain was presented. Figure 5–1 shows an example of the kind of explanation which could be generated in response to a WHY question during a consultation of a simple knowledge base containing rules for plant classification (the user's input is in italics, and the system's responses are in normal type). A HOW explanation would be more or less the reverse of this: an example can be seen in Figure 5–2.

Two of the design goals of MYCIN were that the expert rules in its knowledge base should be of use for purposes other than consultation, and that domain novices should be able to learn about its knowledge domain from the explanations generated during a consultation. However, when researchers attempted to use MYCIN for teaching purposes, it became obvious that its production rule

Figure 5–1: A WHY explanation

Does your plant have a ribbed stem?

⇒ *WHY*

It has been established that:
milky sap in stem = no (rule1)
spines = yes (rule4)
If I can establish that:
ribs = yes (rule5)
stem = one column (rule8)
spine type = brown (rule12)
Then I can conclude that:
plant = column cactus (rule93)

Figure 5–2: A HOW explanation

I have concluded that:
plant = column cactus

⇒ *HOW*

From:
rule1 I concluded that milky sap in stem = no
rule4 I concluded that spines = yes
:
:
Therefore I can conclude from rule93 that:
plant = column cactus

representation and hence the explanations which could be generated from it were insufficient for this task [Clancey 83].

GUIDON was a system which tried to use MYCIN's production rules to tutor students about the diagnosis of infectious diseases [Clancey 79]. This proved to be more difficult than anticipated: for example, there was no encoding of how the concepts in these rules were related to one another, or what the purpose of each concept was in the system. Some rules represented an explicit link between data and hypotheses, while others were there for their effect on the consultation, such as controlling which rules would fire. In addition, it was not possible for GUIDON to explain the expert's problem solving approach, as the problem search space and the strategy used within it were implicit in the ordering of the rule concepts. All of these problems resulted from the fact that production rules really only represent cause-effect relations in a body of knowledge. They state simply that if one or more events occur, then one or more other events will occur, and as such, mimic, often very poorly, the *dynamics* of a domain or of a reasoning process. Thus, a representation based on production rules ignores the fact that a knowledge base can contain many other types of information, such as classificatory knowledge, definitional knowledge, and contrastive knowledge. It is this variance which must be made apparent if a knowledge base is to be used to educate the user about a domain.

Recognising this problem in GUIDON, the researchers began to search for a representation which would make explicit the different types of knowledge found in a MYCIN knowledge base. This would then provide the basis for the generation of richer explanations, which would explain more effectively the problem-solving strategies used to make a diagnosis in the domain of infectious diseases, with a view to educating the user about that domain. The solution found was to enhance the standard rule-based representation by using meta-rules. These classified the rules in the MYCIN knowledge base according to their function in the system, and as such ensured explicit and separate representation of the system's **strategic** knowledge, its **structural** knowledge and its **support** knowledge [Clancey 83]. Strategic knowledge can be thought of as knowledge for in-

voking a rule: it is knowledge which plans how the goals and hypotheses will be ordered during problem solving ('determine *cause of infection* before *therapy to administer*'). Structural knowledge can be thought of as knowledge for **indexing** a rule: it is knowledge which abstracts from and hence indexes the domain knowledge in some way, such as classifying the causes of each disease into *common causes* and *unusual causes*. Support knowledge can be thought of as knowledge for **justifying** a rule: it is lower-level knowledge which explains the reasoning behind a rule, such as making clear why tetracycline should not be given to children.

The new system which was developed as a result of this was NEOMYCIN [Hasling *et al* 84]. Through having a re-representation of the knowledge in the MYCIN knowledge base using meta-rules, NEOMYCIN was able to generate system-based explanations which contained a description of the problem-solving strategies that it had used to determine the result of the consultation. Thus, it was able to generate abstract explanations of general problem-solving principles, which could be used to educate a domain novice.

Another problem with production rules as a representation formalism is that much of the knowledge about the domain is 'compiled' to become part of a rule, and it is this compiled knowledge which is represented: the knowledge which is used to construct the rule is no longer available once the knowledge base has been developed. This can be illustrated using a simplified example from the MYCIN knowledge base. The antibiotic tetracycline should not be given to children, as it can cause discoloration of teeth. However, a rule which makes use of this knowledge will compile it into the form:

IF patient < 9 ...
THEN not drug = tetracycline

This rule is perfectly correct, but has no encoding of the knowledge which caused it to be written.

Finally, the structure of production rules is such that that there is no separation of domain knowledge and problem-solving knowledge. All knowledge is

reduced to rule premises and conclusions, and thus there is no way of distinguishing between the two types. For example, a rule can exist of the form:

IF ...
THEN infection = meningitis
AND clear screen
AND display infection

where the first conclusion relates to the domain and the final two are there for 'housekeeping' purposes. Thus, with a rule-based representation, knowledge about the domain is not represented explicitly, and consequently is not available to the system.

For these reasons, many researchers began to look at representations which would allow explicit representation of a system's domain knowledge separate from its problem-solving knowledge, and which would allow a deeper, more complete representation of the domain. This strategy was used in CENTAUR [Aikins 83], an expert system in the domain of pulmanory function, where knowledge is represented using frames (prototypes) and production rules. Prototypes represent typical situations in the domain and provide a context for the more fine-grained reasoning of the rules. Production rules are used to represent the problem-solving knowledge needed to run the consultation, and are grouped according to their function in the system. With this representation, CENTAUR can give a very clear explanation of what it is doing and why, since the purpose of the knowledge in the system and the context in which it should be used are both made explicit.

This technique is also used in the XPLAIN system [Swartout 81], where knowledge is represented as a Domain Model and a set of Domain Principles, and the knowledge base is constructed from these using automatic programming techniques. This deals with the problem of knowledge compilation by ensuring that the knowledge used to construct the knowledge base is still around once it has been built, and so is still accessible to the system. Using this knowledge, XPLAIN can provide justified explanations.

It is argued by many that an explanation should be tailored to the user. This tailoring can take several forms: for example the system may take into account the user's level of expertise in the domain, and thus generate an explanation for a domain novice which does not contain too many technical terms. Alternatively, the user's current requirements in the consultation may be taken into account, and a short summarised explanation provided in preference to a long detailed one. For such tailoring to take place, the system must have knowledge about the user appropriate to the type of tailoring to be done. There are two ways in which this knowledge can be acquired: explicitly, for example by asking the user questions about the domain or about herself, to establish her level of expertise; and implicitly, for example by monitoring the users interaction with the system and building a model from that.

Examples of systems which use an explicitly-acquired user model are UMFE [Sleeman 85] and a prototype explanation system designed by Wallis and Shortliffe [Wallis & Shortliffe 84]. UMFE is a domain-independent User Modelling Front End subsystem, which acts a 'filter' between the user and an expert system (or any other system, such as an intelligent tutoring system or a database). It assumes that the user has asked the system a question, and the response generated is passed to UMFE before being presented to the user. UMFE acquires a model of the user by asking questions relating to the content of this response, and then presents a tailored version of the response to the user, using terms which it thinks will be understood. In this way, UMFE tailors the content of the generated explanation to a level which it thinks is appropriate, and constructs a model of the user's level of expertise in a domain. This model is dynamically updated each time an explanation is given, in an attempt to maintain an accurate picture of the user's current level of knowledge. This dynamic updating is based on the assumption that something is known by the user if it has been explained by the system, and although this is not necessarily a valid assumption, the model can be changed if the user later asks a question about something which he has been assumed to know.

The prototype expert system which was designed by Wallis and Shortliffe [Wallis & Shortliffe 84] acquires its user model in the first instance by asking the user to rate his level of expertise in the domain, on a scale of one to ten (where ten is expert). Internally, all of the inference rules and concepts in the knowledge base are assigned a numerical measure of complexity, and concepts are also assigned a measure of importance. These measures, together with the user's expertise rating, determine the content of the explanation produced, in terms of both the level at which it is aimed and the amount of detail which is given (more detail is available if requested by the user).

Two examples of systems which use an implicitly acquired user model are ADVISOR [McKeown 88] and GUMAC [Kass & Finin 88]. ADVISOR is an advice giving system for students which can provide information about available courses and can advise about whether a student can or should take a particular course. ADVISOR tailors its explanations to what it perceives to be the user's current goal during a consultation. This is derived by analysing the discourse segment between the user and the system immediately preceding the request for an explanation. The knowledge base for ADVISOR is divided up into several different viewpoints, and it is the relation of the user's goal to one of these viewpoints which determines the content of the generated explanation. Thus, ADVISOR can generate a different explanation in response to the same question if the user's goal is perceived to be different.

GUMAC acquires its model of the user's beliefs about the domain by 'eavesdropping' on the interaction between the user and the system. The information gathered from this is then used along with the current user model and the system's knowledge base to make inferences about the user's beliefs, using a set of implicit acquisition rules (developed from a real-life study of conversations between advice-seekers and a human expert). Kass and Finin conclude that from a practical viewpoint, a user model should be an explicit one in the first instance, and that gradually implicit techniques should be used to update the model during further interactions between the user and the system.

It is important to note at this point that there are situations where it may

be important to minimise the amount of work that the developer of a knowledge base has to do in order for the system to be able to generate an explanation from the knowledge in a knowledge base. By this, we mean the amount of knowledge which has to be represented over and above the domain knowledge and problem-solving knowledge in order that an explanation may be generated. For example, in NEOMYCIN the production rules in the knowledge base had to be categorised and meta-rules represented; in Wallis and Shortliffe's prototype system, each production rule had to be given an importance and a complexity rating, and in addition had to be classified and represented as nodes in a semantic network; in ADVISOR, the knowledge base had to be partitioned to allow the representation of different viewpoints. It is really only in XPLAIN that we see a representation which is sufficient to generate the desired type of explanation with *no additional representation necessary*. The significance of this point is explained in more detail in Section 5.2.1, when we discuss some criteria for the design of an explanation tool within an expert system shell intended for educational use.

5.1.4 Interaction

The last factor to be considered with regard to explanation generation is the interaction between the user and the system. By this we mean the method by which the user can ask for an explanation, and the way in which the content of that explanation is presented: in short, the input and output techniques used by the system.

As described above, the RSC in MYCIN required the user to request an explanation using the single word commands HOW and WHY. As output, the system presented an English translation of the reasoning trace which had been created during the consultation. However, a later addition to MYCIN was a General Question Answerer (GQA) which allowed the user to ask 'free-text' questions about the MYCIN knowledge base, and which used answer-templates to structure its responses. The GQA used keyword search rather than natural language processing techniques to interpret the user's questions and to generate the appropriate ex-

planation. This was possible because of the small, limited technical vocabulary of MYCIN's domain. Of the systems mentioned above, ADVISOR [McKeown 88] best illustrates the use of natural language input and output techniques within an explanation facility.

As we described above, ADVISOR is a system which can provide students with information about available courses, and can advise them as to what courses they can or should take. The user can converse with ADVISOR via a natural language interface which can deal with three types of questions. The first of these is information-type questions such as 'Is natural language offered this semester?' and 'Who teaches expert systems?'. To answer questions of this type, ADVISOR uses an underlying knowledge base, which has represented intersecting multiple hierarchies of course-related information. These provide the different perspectives on the knowledge base which were described above. The other two question types available in ADVISOR are ones which can require inferencing from an underlying expert system: these are **can** questions, such as 'Can I take natural language this semester?' and 'Can I take expert systems and robotics in the same semester?', and **should** questions such as 'Should I take data structures this semester?'. To answer questions of this type, a production system is used and the trace of rule invocations created is available to support the given explanation.

Once the content of an explanation has been derived, ADVISOR uses a Prolog surface generator to translate it into an English-like explanation. For example, in response to the question 'Should I take data structures this semester?' ADVISOR can produce a response such as:

> You should take data structures. I assume that you want to take courses in the normal sequence. Data structures is a first term sophomore course. You have taken the preceding courses.

As we mentioned above, ADVISOR can generate a different explanation in response to the same question if the user's goal is perceived to be different. This technique means that the response shown would not be the only possible answer to the question 'Should I take data structures this semester?'

BLAH [Weiner 80] is an expert system which interfaces to a Truth Maintenance System, and uses assertion and justification to record the reasoning used during a consultation. The main focus of this research was the structure and content of explanations, with a view to reducing their complexity and thus making them easier for the user to understand. BLAH reduced the complexity of its generated explanations by removing any assertions that the system assumed were already known by the user; reducing the number of details given; giving the remaining details incrementally; and adding structural information to the explanation in an attempt to make the underlying structure more apparent. Thus, BLAH can produce explanations which range from a single statement summarising the whole reasoning chain to one with a much more fine-grained level of detail. We would question Weiner's claim that the explanations provided by BLAH can be used successfully for the purposes of instruction, in that users can test their knowledge by asking the system questions and comparing the answers given with their own. However, the system does illustrate well the use of structuring strategies to render the content of an explanation more accessible and easier to understand.

Although the XPLAIN system could generate informative, justified explanations, the only user input available to request an explanation was the question WHY. The explanations given in response were generated using low-level phrase generators and higher-level answer generators. However an extension of XPLAIN, designed and implemented using the EES (Explainable Expert Systems) paradigm [Neches *et al* 85], can answer a broader range of questions, such as justification questions ('What is the significance of <result>?'); questions regarding timing or appropriateness ('When did the system consider/reject <goal, action or conclusion>?'); questions regarding definition or function ('What are the effects of <action>?'); and questions about the system's capabilities ('What does the system know about <concept>?'). The ability of a system to answer a broad range of questions seems to be the focus of much of the recent work on explanation in expert systems (for example [Kidd 85, Hughes 86]). A lot of this work is based on classifications of questions according to their type, with

procedures for answering these questions organised around the classification categories (for example [Lehnert 78, Hughes 86]). Conversely, in an attempt to deal with the ambiguous meaning of many questions, a classification has also been proposed which is based around answer types rather than question types [Gilbert 87]. These are derived from a categorisation of knowledge, where each answer type draws on a specified category for its content.

5.1.5 Summary

In this section we have described the explanation facilities of expert systems in terms of

- the **target** of the explanation
- the **purpose** of providing the explanation
- the **type** of explanation required
- the **knowledge needed** to provide an explanation of the desired type
- the **interaction** between the user and the explanation facility

We discussed how the target of an explanation can come from a broad range of users (such as a domain novice or expert, or a knowledge engineer) and that each of these may potentially require the explanation for a different purpose (such as debugging or education). We concluded that it is not enough to simply make 'providing an explanation' the goal of any system: the content and the structure of that explanation must be carefully considered if it is to achieve its desired purpose. To this end, we discussed two possible types of explanation (system-based and domain-based) as well as the desirability of having an explanation which is tailored to the user's level of knowledge in the domain or to his current goals and requirements in the consultation. Related to this, we described the kind of knowledge needed to produce an explanation of the desired type, and how for most cases it was necessary to use a knowledge representation over and

above production rules. Two examples of this were extending the use of rules by using meta-rules, and bringing in another representation formalism such as frames. We then moved on to discuss the input methods with which a user can ask for an explanation, and the output methods used by the system to provide an explanation (that is, the ways in which the content of the explanation can be presented to the user).

We will now briefly discuss the explanation facilities available in expert system shells, and will then use the factors discussed above to hypothesise about the requirements of an explanation facility as part of an expert system shell intended for use in education.

5.2 Explanation in Expert System Shells

In Chapter 2 we reviewed a number of expert system shells, in five different categories: small, medium and large rule-based shells; inductive shells; and hybrid shells. Most of these provide an explanation facility of some kind, but at best it is the kind of facility offered by the MYCIN RSC. By this, we mean that the shell will be able to produce a rule trace in response to the commands WHY and HOW, entered by the user during a consultation (as described above in Section 5.1.3). Many of the shells are limited to this type of explanation by the fact that they provide a KRL based only on production rules. However, even hybrid tools which make use of multiple representations such as frames, objects or schemas have an explanation facility of this type.

The explanation facilities of these shells are usually presented as a feature which allows the developer of a knowledge base to debug it, by checking on the knowledge used and the inferences made during a consultation. Thus, the target of the explanation is a knowledge engineer, and the purpose of providing it is for debugging. The generated explanation is system-based, with no attempt to tailor it to the user, and for this reason, the only knowledge needed to generate the explanation is a history of the current consultation. At most the user will

have the two commands WHY and HOW with which to request an explanation, and the output will be a rule trace as shown in Figures 5-1 and 5-2. (Some systems will 'structure' the rule trace by presenting only a small portion of it at a time, with the user able to move further into it if desired).

In addition to this, some shells also make use of canned text, which is linked to parts of the knowledge base and which can be displayed during a consultation when that type of explanation is requested. The intention is to provide explanations which are more domain-based, such as providing definitions of terms used in the knowledge base. As is the case with any system using canned text, this technique has many drawbacks. It is very difficult to maintain consistency if the knowledge base is altered in any way, and, as the same explanation is always given under all circumstances, it is rigid and inflexible. The use of canned text also involves advance anticipation of all possible questions which the user might want to ask, which is at best an unrealistic aim. This is related to the query language with which the user can request a canned explanation. However even if input is restricted to a one-word request such as EXPLAIN, as it is in many shells, the knowledge base developer must anticipate all possible points in all possible consultations where that request might appear, and hence provide appropriate text, if the system is to be complete.

This type of explanation facility is inadequate for the use which we described above, that is, as a tool which facilitates exploration of the domain of a knowledge base, with a view to educating the user about that domain. We will now look at how the explanation facility of an expert system shell can be designed to meet that requirement.

5.2.1 How can it be improved?

We established in Chapter 3 that if expert system shells are to be used in education, then their use should not be restricted to the Computer Science curriculum. Instead, shells should be available as classroom tools throughout the curriculum, to help pupils learn about the domain of any knowledge base being consulted.

This implies that users must be given access to the domain knowledge in a knowledge base, and we would propose that one way in which this access can be provided is to include within the shell an explanation tool designed to facilitate exploration of that knowledge. We will now look at the requirements for such a facility in terms of the five factors discussed above.

The first two factors we discussed were that it is essential to consider who the intended target of an explanation will be, and what the purpose is of providing it. If a shell is to be used throughout the curriculum as a tool which promotes learning, then the target of the explanations which it generates can be classified as a domain novice, and the purpose of generating that explanation is to try to educate that novice about the domain of the knowledge base from which the explanation is being generated. (This is not to say that these explanations would not also be of use to a domain expert or a knowledge engineer for the purposes of verification or debugging).

When we consider the type of explanation required, it is obvious that, if the explanation is to be used to educate the user about the domain, then it must be domain-based. In addition, given the environment in which the shell will be used and the potential variety of users, it also seems that the generated explanations should be tailored to the individual user in some way.

As we showed above, the knowledge needed for effective explanation generation must be more than can be represented within a knowledge base containing only production rules. We discussed the need for explicit representation of a domain's structural, strategic and support knowledge if the system is to explain its problem-solving strategies, and the need for a system to have access to an explicit representation of the domain knowledge if that knowledge is to be used in an explanation, for example for the purposes of justification. Thus, if a shell is to generate domain-based explanations, which is the goal of the explanation facility we are discussing here, then it must have access to the domain knowledge in a knowledge base. This can be provided by using a technique such as that used in the XPLAIN system, which ensures that a full model of the domain still exists after the knowledge base has been developed; or that used in CENTAUR,

which uses a representation based on frames and production rules to ensure a separation of domain knowledge and problem-solving knowledge.

When we discussed the extra knowledge needed by the system over and above production rules, we briefly commented on the possibility that an extra burden may be placed on the knowledge base developer by the representation method used, and that in certain circumstances it may be desirable to minimise this as far as possible. From informal observations of and discussions with users of small expert system shells, it is clear that many of them (and not only novices) find building a knowledge base difficult. Even if they have acquired all of the knowledge which they need to build the knowledge base, structuring and representing that knowledge can present many problems (this point is discussed in Chapter 3). In the kind of educational environment in which we envisage the EXPLORES shell being used, we expect that one of the two main uses of the shell will be the building of knowledge bases; that many of the users will be computer novices; and that time spent on the computer will often be limited by timetabling restrictions or machine availability. For these reasons we discussed, with regard to the development of the EXPLORES KRL, how it was necessary that the user should not need to consider the inferencing done during a consultation when developing a knowledge base. This also applies to the shell's explanation facility: it is desirable that if an explanation is to be generated from the knowledge in a knowledge base, then the user should have to do no knowledge representation over and above that which is necessary for consultation. Thus, the technique used to generate an explanation should be one which requires little or no additional knowledge representation by the user.

In deciding which methods we should use of those available to the user to request an explanation (such as one-word commands or natural language input) and those used by the system to output an explanation (such as natural language generation or structuring strategies) we must consider the fact that an expert system shell can, by definition, deal with knowledge bases from a number of different domains. Although for any one shell each knowledge base will contain knowledge represented in the shell's own KRL, in theory the domain of

that knowledge base can be any which can be made to fit the representation formalism used. This diversity of domains means that we cannot use a 'free-text' input technique which depends on the limited or predictable vocabulary of any one domain (such as that used by the MYCIN GQA). At the other extreme, one-word commands are not expressive enough for the user to properly explore the domain knowledge. Thus, the explanation facility requires a generic, domain-independent input technique which can be adapted to any knowledge base, and which allows the user to explore that knowledge base in an interesting and effective way. Similar problems also arise with regard to the output technique used by the system: it too must be independent of any one domain.

We have now outlined the requirements for an explanation facility as part of an expert system shell intended for use in education. This explanation facility provides the means whereby the shell can be used in variety of subject areas, with a view to educating the user about the domain of the knowledge base being consulted. For this to be possible, the user must be able to explore that domain knowledge in an interesting and informative way, using the explanation tool. Thus, the target of the explanations generated by the shell will be a domain novice, and their purpose will be the education of that novice. This implies that the shell must be able to generate domain-based explanations, which are tailored to the user in some way. To do this it must have access to an explicit representation of the domain knowledge, separate from its problem-solving knowledge, in a way which avoids 'over-burdening' users by asking them to represent 'extra' knowledge over and above that needed for consultation. The generic nature of shells dictates that the technique used to generate an explanation must be domain independent, and that the input and output routines must be ones which are not tied to any one domain and which are flexible and expressive enough to allow effective exploration.

Having outlined the requirements for an explanation facility for an educational expert system shell, we will now describe the explanation tool developed within the EXPLORES shell.

5.3 Explanation in the EXPLORES Shell

The EXPLORES shell is an expert system shell designed for educational use. It has environments for both building and consulting a knowledge base, and a number of tools which can be used to supplement these. One of these is an explanation tool, designed to be used from within the shell's consulting environment. We will now describe this tool in terms of the five factors discussed above and the requirements outlined in the previous section. A more detailed description of it can be found in the next chapter, where we outline the structure of the EXPLORES shell and describe in detail each of the facilities and the tools which it offers.

5.3.1 The EXPLORES explanation tool

The function of the explanation facility in the EXPLORES shell is to provide a means whereby the user can explore the domain knowledge of a knowledge base. This provides the potential for the user to learn something about that domain, and in the previous section, we outlined the requirements for such a facility. However before describing the realisation of those requirements in the EXPLORES explanation tool, it is important to point out some additional constraints which influenced the way in which this tool was designed and implemented. The most important of these is that, if the shell is to be used in an educational environment, then it has to be implemented on the kind of hardware which might realistically be available in classrooms. By this, we mean that the shell has to be micro-based, and this restriction means that any techniques which require a large amount of memory or computing power for the generation of explanations cannot be used. In addition, the practicalities of classroom use dictate that the techniques used must produce results in a relatively short space of time: users cannot be expected to endure a lengthy wait before receiving a requested explanation. These restrictions resulted in many compromises in the design of the EXPLORES explanation tool, and we will discuss these in more detail as they emerge in the ensuing description.

We proposed that since the intended target of an EXPLORES explanation is a domain novice, and that an explanation is generated with the purpose of educating that novice, then the shell must be able to generate domain-based explanations if that purpose is to be achieved. The generation of domain-based explanations requires an explicit representation of the domain, and from the methods by which this can be achieved, we decided that the most tractable was to design a KRL for the shell which would allow this kind of representation. Thus, the EXPLORES KRL is based on production rules and the simplified frames which we called *things* (we described this KRL in detail in the previous chapter), where the latter allow the user to describe the objects and classes of objects in a domain in terms of their properties and relationships. The decision to use this method was taken for two main reasons: it did not depend on a large amount of computing power (unlike a method using a technique such as automatic programming); and it did not place any additional burden on the user with regard to any extra representation over and above that needed for consultation (unlike a method using a technique such as meta-rules).

The requirements for the method by which a user can request an explanation stated that it should not be tied to any one domain and that it should be flexible enough to allow the user to explore the domain knowledge effectively. Ideally, this would mean that the interaction between the user and the system would be via a natural-language interface which would be able to interpret and respond to free-text questions about any aspect of any domain. Given the impracticality of trying to build such an interface for a small micro-based shell, it was decided that a reasonable compromise was to provide a number of question-templates, which allow the user to pose a variety of different questions about the domain knowledge. In addition, the content of the generated explanation is slotted into answer-templates, appropriate to the explanation to be given. Although this constrains the user to some extent, it is hoped that this can minimised by having a large and eclectic group of questions: the variety of questions available and the scope which they offer for exploration is one of the topics discussed in

Chapter 7, where we describe the evaluation of the EXPLORES shell and, more specifically, of the explanation tool.

The EXPLORES explanation tool can be used at any point during a consultation. In response to being asked a question by the shell, the user can type **explain**, and this command loads and runs the explanation tool. Initially, the user is presented with a menu of the four main explanation categories, namely **description, comparison, property analysis** and **relationship analysis.** If either the description category or the comparison category is selected from this menu, then a subsequent menu appears with appropriate question-templates. If either the property analysis or the relationship analysis category is chosen, then a menu of three sub-categories is presented. For property analysis these are general, features and actions, and for relationship analysis they are general, classification and component. One of these subcategories must be chosen before a menu of question-templates can be presented. Figure 5–3 shows a complete list of the question-templates which are available within each explanation category and sub-category. Each of these questions corresponds to different aspects of a *thing*, the knowledge structure available for the explicit representation of domain knowledge in the EXPLORES KRL. Thus, the question set in Figure 5–3 was developed because of the scope which it provides for exploration of that domain knowledge.

Once a question-template has been selected, prompts are given for the user to complete the template, that is, to replace the upper-case letters with, for example, the name of a *thing* or of a property. To eliminate the possibility that the user will ask a question which the system cannot answer, for example by completing a template with the name of a *thing* or a property which does not exist in the knowledge base, a menu of possible fillers is presented for each part of a question-template. For example, if the part of the question-template being completed relates to the *things* in a knowledge base, a menu of all possible *things* appears. If it concerns the features which a thing has or the actions which it performs, then all of the features or actions in the knowledge base are presented in a menu. The user can then select from the offered menu the filler to be used

Figure 5–3: Question-templates in the EXPLORES explanation tool

Description:
- Describe X

Comparison:
- Compare X and Y
- How do X and Y differ?
- How are X and Y similar?

Property analysis:

General:
- What properties does X have?

Features:
- What things have P?
- What things have P with value V?
- What things do not have P?
- What things do not have P with value V?
- What does X have?
- Does X have P?
- Does X have P with value V?

Actions:
- What things do A?
- What things do not do A?
- What does X do?
- Does X do P?

Relationship analysis:

General:
- How are X and Y related?

Classification:
- What is X an example of?
- What are examples of X?
- Is X a Y?

Component:
- What is X part of?
- What parts make up X?
- Is X part of Y?

Figure 5–4: Completing a question-template I

(menu of templates is presented)

User selects:	How are X and Y related?
System writes:	How are

(menu of things is presented)

User selects:	column cactus
System writes:	How are column cactus and

(menu of things is presented)

User selects:	resinifera
System writes:	How are column cactus and resinifera related?

in that part of the template. Figure 5–4 illustrates this process for a question where there are two fillers required and both of them must be the name of a *thing*: Figure 5–5 illustrates it for a template requiring the name of a *thing* and of a feature as fillers.

When the chosen question-template has been completed, the explanation tool will attempt to answer the question, that is, it will try to generate an explanation. The strategies used to find the content of an explanation depend on the type of question asked: these are described in detail in the next chapter. The structure of the presented explanation is determined by output routines which slot together a number of answer-templates. Figure 5–6 shows the how the explanation generated in response to a question of the type asked in Figure 5–4 would be constructed.

In our discussion of the requirements for an explanation facility such as this, we stated how the generated explanation should be tailored to the individual

Figure 5–5: Completing a question-template II

(menu of templates is presented)

User selects:	Does X have P?
System writes:	Does

(menu of things is presented)

User selects:	peanut cactus
System writes:	Does peanut cactus have

(menu of features is presented)

User selects:	milky sap in stem
System writes:	Does peanut cactus have milky sap in stem?

user in some way. We have already described some systems which do this, such as ADVISOR, UMFE and GUMAC. Given factors such as the computational cost of carrying out such user modelling; the lack of computing power in the machines on which the EXPLORES shell would be run; and the large number of potential users in any educational environment, we decided tried to reach a compromise by making it possible for different users to choose different answers for themselves. In the event, this took the form of follow-up questions, with which the user can ask the system to elaborate on many aspects of the generated explanation.

Once an explanation has been generated in response to a question, the user is presented with an elaboration menu. This contains all of the ways in which the system can elaborate on that explanation, using follow-up questions. The options which appear on the elaboration menu depend on the original question asked; the content of the explanation given; and the knowledge in the knowledge base. For example, follow-up questions would be available about each of the main *things* named in the explanation shown in Figure 5–6 (*column cactus*,

Figure 5–6: Constructing an answer using answer-templates

Question asked: How are *column cactus* and *resinifera* related?

System finds that:

(i) column cactus is a descendant of spiny cactus, which is a descendant of spiny plants
(ii) resinifera is a descendant of spiny euphorbia which is a descendant of spiny plants

Answer-templates used:

X is an example of Y
X and Y have Z as a common ancestor

Explanation given:

column cactus is an example of *spiny cactus*
spiny cactus is an example of *spiny plants*

resinifera is an example of *spiny euphorbia*
spiny euphorbia is an example of *spiny plants*

column cactus and *resinifera* have
spiny plants as a common ancestor

resinifera and *spiny plants*). Follow-up questions are not presented as question-templates, since they will only apply to one property or thing. If, as in the example just stated, there is more than one thing or property which can be followed-up, these are listed in a menu, and the user is asked which he wants to know more about. When one of these is selected, a menu of appropriate follow-up questions appears. Figure 5–7 lists all of the follow-up possibilities available in the EXPLORES explanation tool.

Follow-up questions are always questions which can be answered by the shell: that is, the system checks all possible questions and only offers those where the knowledge base contains knowledge of the type needed in the answer. Again, this eliminates the possibility that the user will pose a question which the system cannot answer. For example, for the three *things* mentioned previously (*column cactus*, *resinifera* and *spiny plants*), the follow-up question **Examples of it** would only be available for *spiny plants*, since there are no examples of *column cactus* or of *resinifera* in the knowledge base (they are at the bottom of the hierarchy). However, the question **What it is an example of** would be available for all three *things*.

Where there are two *things* in an answer, it is possible to follow-up the explanation in terms of 'both *things* together', for example by finding out how they are related. Figure 5–8 shows the follow-up possibilities offered for four of the question-templates from Figure 5–3 (the numbers in square brackets in Figure 5–8 refer to the number associated with each of the follow-up questions listed in Figure 5–7). Any number of follow-up questions can be asked, and the same follow-up question can be asked more than once. Using this facility, the user can control the length, content and direction of the explanation generated in response to a question. This technique cannot be classed as user modelling in the way that we described above. However, it can be thought of as a way in which users can individualise explanations for themselves, according to the information which they want to find out. Given the constraints mentioned above, it is a reasonable compromise.

The use of follow-up questions is of most benefit to users in situations where

Figure 5–7: Available follow-up questions

For one *thing*:

(1) What it has
(2) What it does
(3) Examples of it
(4) What it is an example of
(5) Parts of it
(6) What it is part of

For properties:

(7) Things which have this
(8) Things which do this
(9) Values for this

For two *things*:

(10) How they are related
(12) Unique properties they have
(12) Identical properties they have

Figure 5–8: Examples of possible follow-up questions

Describe X

Follow-up questions [2, 3, 4, 5, 6]

Apply to filler for X

How do X and Y differ?

Follow-up questions [1, 2, 3, 4, 5, 6]

Apply to either filler for X or filler for Y

Follow-up questions [10, 11, 12]

Apply to fillers for X and Y together

What properties does X have?

Follow-up questions [1, 2, 3, 4, 5, 6]

Apply to filler for X

Follow-up questions [7, 8, 9]

Apply to property selected from those in answer

How are X and Y related

Follow-up questions [1, 2, 3, 4, 5, 6]

Apply to either filler for X, filler for Y

or their common ancestor, given in answer

the original question asked has produced a negative response. For example, if the user asks a question such as 'Is *column cactus* a *thorny euphorbia*?', then the generated explanation will be 'No, *column cactus* is not an example of *thorny euphorbia*'. In this case, the use of follow-up questions can be seen as an attempt to correct a misconception that the user appears to have, and thus to provide a learning opportunity. This technique is demonstrated well in McCoy's work on misconceptions [McCoy 84], where she proposes a computational model for correcting users' misconceptions regarding the objects in a knowledge base, where these misconceptions are apparent from the questions being asked by the user about these objects. McCoy lists three types of **superordinate misconception**, that is, where an object is misclassified, and three types of **attribute misconception**, that is, where a property is wrongly attributed to an object. The strategies used for correcting these apparent misconceptions are based on assumptions about why the misconception may have occurred. For example, in one type of superordinate misconception, the misconception is assumed to have occurred because the object which is misclassified shares many properties with objects which are examples of the posited superordinate. An example question of this type would be 'Is a whale a fish?'. McCoys strategy for correcting a misconception such as this is to structure the generated explanation in the following way:

- Deny the posited superordinate and indicate the correct one ('No, a whale is a mammal')

- State attributes or properties that the object has in common with the posited superordinate ('Although it has fins and lives in the water ...')

- State defining attributes of the real superordinate ('... it's a mammal since it is warm blooded and feeds its young with milk).

This work provides great scope for enhancing the potential of EXPLORES as a learning tool, in that many of the questions which can be asked within the explanation tool are ones in which misconceptions of the type classified by

McCoy could be picked up and corrected using appropriate follow-up questions. The possibility of extending the explanation tool in this way is discussed further in Chapter 8, under 'Further Work'.

5.4 Summary and Conclusions

In analysing the explanation facilities of expert systems, we found a number of factors which were important for effective explanation generation. We proposed that the **target** and the **purpose** of an explanation must be considered, as different categories of user may require an explanation for very different reasons, and the structure and the content of the generated explanation must take this variance into account. We then discussed different **types** of explanation which could be generated, with regard to whether they are system-based or domain-based, and whether or not they are individually tailored in any way, and we analysed the type of **knowledge needed** to generate an explanation of the desired type. From this we concluded that production rules alone are insufficient for most types of explanation, and described other representation methods which have been used to improve the range and quality of the explanations produced. The final factor, **interaction**, was discussed in terms of methods by which a user can request an explanation (input) and by which the system decides on the structure and content of the generated explanation (output), and we described how the ability of a system to deal with natural language input is often dependent on there being a small, very well-defined domain.

From our discussion of these five factors, we constructed a set of criteria for an explanation facility as part of an expert system shell intended for educational use. The first requirement was that the explanation would be targeted at a domain novice, with the purpose of educating that novice about the domain of the knowledge base being consulted. To achieve this goal, the system must be able to generate domain-based explanations tailored to the needs of user in some way: this dictates that the knowledge in a knowledge base must be represented using

a formalism which separates domain knowledge and problem-solving knowledge. Time and skill constraints with regard to potential users implied that the representation used be one which requires no further representation of any knowledge over and above that needed for consultation. Finally, we imposed the need for input and output techniques which are domain-independent, flexible and expressive. An additional constraint was that the shell had to be implemented on a micro if it was to be used in schools. This meant that we could not afford to use any generation technique which was computationally costly. Having listed these requirements, we then described the explanation tool developed for the EXPLORES shell.

The EXPLORES explanation tool generates domain-based explanations: this is possible because the shell's knowledge representation language allows explicit representation of domain knowledge (using frame-like *things*) separate from its problem-solving knowledge (represented using *rules* and *questions*). The user interacts with the tool using question-templates from four main explanation categories, and the system uses answer-templates to structure the content of its responses. In addition to the generated explanation, supplementary, related information is available from an elaboration menu. This contains follow-up questions, the variety and subject of which is determined by the original question; the generated explanation; and the knowledge in the knowledge base. We also described how follow-up questions could be used to correct apparent misconceptions in the user's model of the domain. Although EXPLORES does not tailor its answer to the user in the normal sense, and does little to account for different levels of domain expertise, the elaboration menu does allow users to control the length, content and direction of an answer with respect to their current goals and requirements, and is a tractable compromise in the absence of sophisticated user modelling.

Given the appropriate hardware, potentially almost all of the currently available expert system shells could be used throughout the school curriculum for both the development and the consultation of knowledge bases. To develop a knowledge base it is necessary to acquire, organise, and represent a body of

knowledge about a domain, and in carrying out these tasks, it would be hoped that pupils would learn something about that domain. The other use of shells in classrooms is where pupils consult an existing knowledge base, and we discussed in Chapter 3 how this option is open to pupils with a wider range of ages and abilities than that of developing a knowledge base. During a consultation, the user provides the shell with information about the current situation, and receives in return a decision based on the given information and the knowledge in the knowledge base. The user need not think about how that decision was taken, for example what knowledge was used or what deductions were made. The provision of the standard shell explanation facility, where a trace of the preceding consultation is generated in response to HOW and WHY questions, can alleviate this problem to some extent: if desired, the user can be made party to the decisions taken during the consultation, and the knowledge upon which those decisions were based. However, if one of the purposes of a consultation is to try to educate the user about the domain of the knowledge base being consulted, then we would argue that this kind of explanation is insufficient to give the user an accurate picture of that domain, and that none of the currently available shells have any other facilities capable of achieving that goal. For this reason, we have included in the EXPLORES shell an explanation tool which allows the user to pose questions about the domain itself, such as asking for descriptions of the knowledge there; making comparisons between parts of that knowledge; or finding out how aspects of the domain are related.

The EXPLORES explanation tool has no tutoring strategies for specifically teaching the user about the domain of a knowledge base. It is an exploratory environment, and like all such environments, it is dependent on the motivation of the user. By this, we mean that the user is not coached and lead through the knowledge base: the shell provides the means by which the user can explore a knowledge base if he desires to do so. In an attempt to increase the user's motivation for carrying out such exploration, we have designed the environment in such a way that the task is made as simple as possible. Like the rest of the shell, the EXPLORES explanation tool is menu-driven: this means that the user is

presented with all possible choices at any one time, and so does not have to know himself what the options are. As before, each menu has a safety option which allows the user to return to the previous choice point with no other action taken. This extensive use of menus, and of question-templates with lists of all possible fillers, ensures that the user need have no prior knowledge about what there is to find out, or indeed what he wants to find out, from a knowledge base. In addition, the possibility of redundant questions which the system cannot answer is also eliminated. In Chapter 8, we discuss the development of teaching materials for the shell, which it is hoped will also encourage the user to explore a knowledge base.

Previously, we argued that expert system shells have some potential as classroom tools, in that the right shell could be used to help pupils learn about a number of domains in a variety of subject areas. In its explanation tool the EXPLORES shell provides a means whereby a motivated user can be educated about the domain of a knowledge base by asking questions about it. In return, the system generates explanations about that domain, and allows the user to find out more about different aspects of the generated explanation. Although it is not yet clear exactly what constitutes 'the right shell', we would propose that the explanation tool of the EXPLORES shell goes some way towards helping the classroom potential of shells be realised. In the next chapter we describe the structure of the rest of the shell. However we return to this point in Chapter 7, where we discuss the evaluation of the EXPLORES shell by experienced teachers.

Chapter 6

The EXPLORES Shell

In Chapter 3 we presented a list of criteria which we proposed should serve as the basis for the design of an expert system shell for use in education. These emerged from an initial evaluation which comprised interviews with teachers at the three participating institutions; informal discussions with and observations of these teachers using a variety of expert system shells; evaluation of those shells by the teachers using questionnaires; a case-study visit to a typical secondary school; and an evaluation of existing shells by the author. During the evaluation we also considered *who* in a classroom would use an expert system shell; *how* they could use it; and most importantly, *why* they should use it, that is, what would we expect the benefits of using one to be. The criteria presented were based upon the results of the evaluation and the answers to these questions.

In this chapter we look in greater detail at the expert system shell which has been designed and implemented as result of these analyses, that is, the EXPLORES (EXPLanation ORiented Expert System) shell. In previous chapters we described the knowledge representation language (KRL) developed for the shell (Chapter 4), and outlined the main features of the mechanisms used by it for explanation generation (Chapter 5). Here, we begin by describing the overall structure of EXPLORES, and follow this by taking each part of the shell in turn and outlining its main features; describing how it relates to other parts of the shell; and discussing any problems which were encountered during its implementation.

6.1 Overall Structure

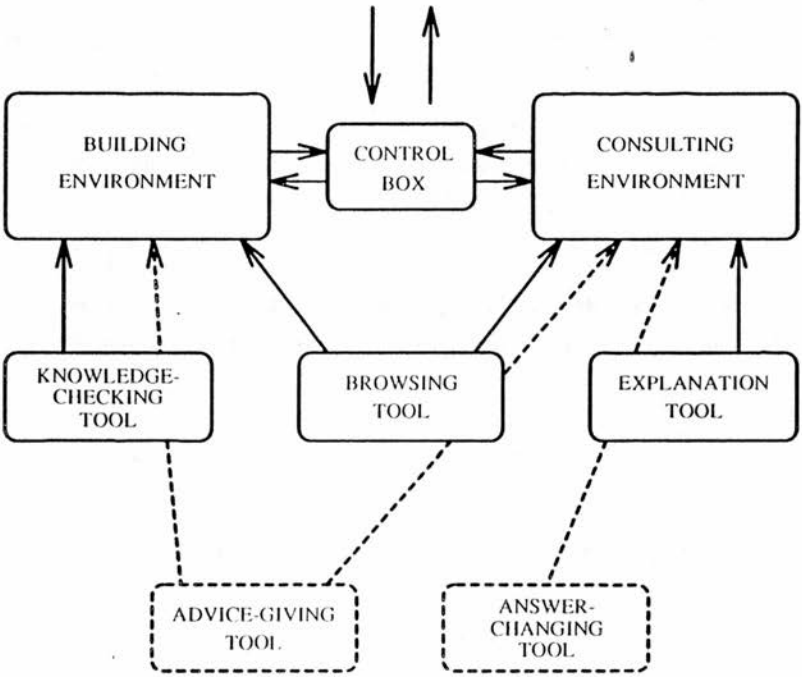
Previously, we proposed that one way to increase the potential benefits that pupils might gain from using an expert system shell would be to ensure that it has facilities for both building and consulting a knowledge base. For this reason, EXPLORES has two separate but linked environments for carrying out these tasks.

We also stated that the shell should, as far as possible, accommodate users with a wide range of computer skills and from a range of abilities and age groups. This implies that it should be adaptable in some way, with the ability to be tailored to the needs of any one individual, and for this reason EXPLORES has been designed as a toolkit shell. By this we mean that there are a number of tools which can be loaded into the building and consulting environments according to the user's requirements.

Figure 6–1 shows the structure of the EXPLORES shell. The entry point is the control box from which the user can move to either the building or the consulting environment. It is also possible to move from one environment to the other via the control box, without losing a loaded knowledge base. This is a particularly useful feature for tasks such as debugging, since a knowledge base can be developed in the building environment; consulted in the consulting environment; and then have necessary alterations made to it back in the building environment. As Figure 6–1 shows, there are five supplementary tools which can be loaded into one or both of these environments: a knowledge checking tool; a browsing tool; an explanation tool; an answer-changing tool; and an advisory tool. The first three of these have been implemented and are described in more detail below. Unfortunately, time constraints prevented the implementation of the answer-changing and advice-giving tools. However, their intended function in the shell is outlined below, along with a discussion of the need for such tools in an expert system shell.

The design of the EXPLORES interface was limited by the facilities available within Arity Prolog and the RML Nimbus. For this reason, it was not possible

Figure 6–1: The structure of the EXPLORES shell



to make use of a full WIMP environment: the shell has no facilities for using a mouse, and the simple windowing and menu facilities used were implemented by Dr Peter Ross [1] as C-routines linked to the Arity Prolog interpreter. Despite these restrictions, the EXPLORES user-interface has several features designed to make using the shell quick and easy. EXPLORES is almost exclusively menu-driven, with typing kept to a minimum: this greatly reduces the possibility of input-error, and ensures that users know exactly what they can do at any point. In addition, the first option on every menu is always a safety, 'do-nothing' option which returns the user to the previous menu with no other action taken. This ensures that errors resulting from accidental key-pressing are also minimised. Although somewhat trivial, features like these can become particularly important in a classroom where users do not want to spend a large proportion of the limited time that they have on error recovery.

We will now describe each part of the EXPLORES shell in turn, outlining its main features; how it relates to other parts of the shell; and, if it was implemented, the design decisions taken and the problems encountered during its implementation.

6.2 The EXPLORES Building Environment

Many small shells require that a knowledge base be built within a standard text editor. For novice users, this often entails learning how to use a text editor before being able to build a knowledge base. In addition, it means that building a knowledge base involves learning the syntax and semantics of the shell's KRL, and that the resulting knowledge base can contain simple syntax errors such as a missing full stop (these points are discussed in detail in Chapter 3 and Chapter 4). The provision of a building environment ensures that support is provided for building a knowledge base which is specific to the KRL of the shell; that it

[1] Department of Artificial Intelligence, University of Edinburgh

is only necessary for the knowledge base developer to be familiar with the basic structure of the KRL; and that syntax errors are kept to a minimum, since most of the processing and structuring of the entered knowledge is done internally by the shell [2] . Syntax and semantics errors can be further minimised by using the knowledge checking tool, which analyses a knowledge base and reports on any errors that it finds: this is described in more detail in Section 6.4.1.

Within the EXPLORES building environment, it is possible to:

- load a knowledge base
- unload a knowledge base
- edit a knowledge base
- see the knowledge in a knowledge base
- add knowledge to a new or existing knowledge base

It is also possible to leave the building environment and return to the control box.

On entering the building environment from the control box, the user is presented with a menu containing each of these options: this is referred to below as the main building menu. We will now briefly describe each of the features found in the EXPLORES building environment.

6.2.1 Loading a knowledge base

In the building environment it is possible to create a new knowledge base or to alter an existing one. If the latter is the case, then the knowledge base to

---

[2] As mentioned previously, it is not possible to build an EXPLORES knowledge base outwith the shell unless the user is prepared to structure it as a list of Prolog clauses. However, there is no reason why this could not be altered in a future version of EXPLORES

be altered can be loaded into the building environment by selecting the 'load' option from the main building menu. It is currently only possible to have one knowledge base loaded at any one time (that is, one file containing one hierarchy of *things*). Again, this is something which could be altered in future versions of EXPLORES if necessary. Because of this restriction, EXPLORES checks the building environment to see if there is any knowledge there already. If there is, the shell advises the user to save and unload that knowledge before attempting to load another knowledge base. Once the building environment is empty, a menu is presented which lists all of the knowledge bases in the current directory (all files which have the extension .KBS), from which the one to be loaded can be selected. If no knowledge bases can be found in the current directory, a message is given to that effect.

6.2.2 Unloading a knowledge base

The restriction of allowing only one knowledge base in the building environment at a time means that any knowledge currently residing there must be removed before a new knowledge base can be added. This can be done by selecting the 'unload' option from the main building menu. To prevent accidental loss of any new knowledge added, the option of saving that knowledge is given before the knowledge base is unloaded.

6.2.3 Adding knowledge

In the building environment it is possible to add knowledge to an existing knowledge base or to a new knowledge base. This can be done by selecting the 'adding' option from the main building menu (if knowledge is to be added to an existing knowledge base then it must first be loaded as described above).

When the adding option is selected, the adding menu is presented. This has options for adding *rules*, *questions*, *assignments* and *things*; for switching the knowledge checker on and off; and for checking the knowledge in a knowledge

base. Section 6.4.1 describes what happens when knowledge is added with the knowledge checking tool switched on, and how that tool can be used to check a knowledge base for syntax and semantics errors.

As discussed previously, the provision of a building environment within an expert system shell means that it can be dedicated to the shell's KRL. Thus, when knowledge is being added to a knowledge base within the EXPLORES building environment, prompts are given which are appropriate to the part of the knowledge structure being added. This can be illustrated by showing what happens when a rule is added to an EXPLORES knowledge base. As described in Chapter 4, *rules* in the EXPLORES KRL have the form

tag:
IF *premise*
THEN *conclusion*

Thus, when a rule is being added to a knowledge base, the user can be prompted for each part of it as follows:

Rule tag the tag of the rule being added can be entered. Each rule must have a unique tag: this allows it to be identified by name, for example when a knowledge base is being edited or when it is being checked by the knowledge checker.

if the premise of the rule being added can be entered. If it consists of more than one premise linked conjunctively and/or disjunctively, only the part up to the first link should be entered. When this has been done, a premise-menu appears with the three possible options for the next part of the rule, namely *and*, *or* and *then*. There are some restrictions about the structure that the premise of a rule can have (see Appendix A). If the entered premise does not adhere to these restrictions, it will not be accepted: an error message will appear, and the user will be asked to try again.

If *and* or *or* is selected from the premise-menu, the appropriate prompt

(and or or) appears and the next part of the premise can be entered, after which the premise-menu appears as before.

If *then* is selected, a **then** prompt appears and the conclusion of the rule can be entered.

then the conclusion of the rule being added can be entered. Again, if it contains more than one conclusion linked by conjunction (rule conclusions cannot be linked disjunctively [3]), it should only be entered up to that point, after which the conclusion-menu is presented. This contains the options *and* and *end of rule*, and if *and* is selected, the next part of the conclusion can be entered. As above, a conclusion which is not legal within the EXPLORES KRL will not be accepted.

The conclusion menu is presented until the *end of rule* option is chosen, at which point the complete rule is displayed and is added to the knowledge base.

The other three knowledge structures in the EXPLORES KRL are added in a similar way: prompts are given at all points for the next part of the structure to be entered. Thus, although a knowledge structure can, for example, contain spelling errors or can be logically incorrect within the knowledge base as a whole, the fact that the building environment is built around the EXPLORES KRL makes it difficult to enter a knowledge structure which is syntactically incorrect. This is not the case if a standard text editor is used, and, as stated above, means that the user's knowledge of the syntax of the EXPLORES KRL need only be minimal.

6.2.4 Viewing knowledge

It is possible to look at the knowledge currently in the building environment by selecting the 'viewing' option from the main building menu. This causes the

---

[3]This point is discussed in Chapter 4 with regard to the CRESS KRL

viewing-menu to be presented, which contains options for viewing all of the rules, questions, assignments or things in the knowledge base which is currently loaded (if there is one). When one of these options is chosen, all of the knowledge of the selected type is presented, one screen at a time, with a 'stop or continue' option at the end of each screen. This is a textual display, rather than a graphical display such as that described in Chapter 3 with regard to Expert Builder. Other options in the viewing menu allow the user to view all of the knowledge in the knowledge base; to load the browsing tool and browse through the knowledge in the knowledge base (see Section 6.4.2); and to return to the main building menu.

6.2.5 Editing knowledge

There are occasions when it is necessary to edit the knowledge in a knowledge base. Again, many shells insist that this is done using a text editor. However, EXPLORES has a facility which allows editing of a knowledge base from within the building environment. This can be accessed by selecting the 'edit' option from the main building menu.

The EXPLORES editor has two parts: a viewing window and an editing window. In the viewing window (the left half of the screen) the knowledge in the building environment can be viewed as described above in Section 6.2.4. This provides a way of seeing exactly which knowledge structures can be edited. In the editing window (the right half of the screen) the structure to be edited can be selected and the required editing carried out.

The editing menu offers the option of editing rules, questions, assignments or things. On selecting one of these, the unique identifier must be entered of the specific structure of that type which is to be edited (that is, the tag of a rule; the name of a thing; the variable-part of an assignment; and the variable part of a question). When this has been done, the chosen structure will be displayed in the viewing window. Currently, the editing operations available in EXPLORES are fairly simple, involving strategies such as replacement of whole lines or parts of knowledge structures.

For example, to change a rule from:

rule1:
IF *sky = blue*
THEN *colour = red*

to:

rule1:
IF *sky = blue*
THEN *colour = blue*

the user indicates that the conclusion is to be changed by choosing the appropriate option from the offered menu, after which a prompt appears for the new conclusion to be entered. The editing options which are available depend on the knowledge structure to be edited. Thus, when editing

rules the rule tag, the premise and the conclusion can all be changed or the entire rule can be deleted

questions the question-part and the variable-part can both be changed, or the entire question can be deleted

assignments the variable-part and the value-part can both be changed, or the entire assignment can be deleted

things the properties and relationships of a thing can be changed, or the entire thing can be deleted. If a property is to be altered, a menu is presented which contains options for adding a new property (either a feature or an action) and for altering the existing ones. If a new property is to be added, the appropriate prompts appear in the same way as in the adding environment. In addition, an existing property can be altered by changing its name and/or its values, or by deleting it. If the relationships of a thing are to be altered, the type of relationship in question can be chosen from a relationship menu, at which point another menu is presented which contains options for adding a new relationship of that type and for altering

an existing one. A new relationship can be added as described above: an existing relationship can only be deleted. It is not possible to change the name of a thing, as this could cause problems with regard to the consistency of the relationships within a hierarchy.

Overall, the EXPLORES editor is fairly limited: for example, it lacks any facilities which would allow character-by-character changes rather than whole-line replacement. However, although this can be time-consuming, for example for a rule which has a conclusion with many parts, the editor is still an improvement on that provided by other small shells, in that it allows the user to see the knowledge which can be edited; it is built around the specific KRL of the shell; it is menu-driven; and that the contents of the menus provided are largely dictated by the structure of the KRL *and* by the specific knowledge in the knowledge base. Some possible improvements to the EXPLORES editor are discussed in Chapter 8 under 'Further Work'.

6.2.6 Saving a knowledge base

If a new knowledge base has been created or an existing knowledge base has been altered, it can be saved by selecting the 'save' option from the main building menu. If an existing knowledge base is being saved, then this can be done under its current name or under a new name: if the knowledge base is a new one, then a name must be entered for it.

6.3 The EXPLORES Consulting Environment

An expert system shell must provide a means by which a knowledge base can be consulted: in the EXPLORES shell this can be done within the consulting environment. Like the building environment, the consulting environment can be accessed from the main control box (see Figure 6–1).

In the consulting environment it is possible to:

- load a knowledge base
- consult a knowledge base
- see the knowledge in a knowledge base
- unload a knowledge base

It is also possible to leave the consulting environment and return to the control box.

On entering the consulting environment from the control box, a menu is presented showing each of these options: this is referred to below as the main consulting menu. We will now briefly describe each of the options found in the EXPLORES consulting environment.

6.3.1 Loading a knowledge base

Before a knowledge base can be consulted, it must be loaded into the consulting environment. (The only exception to this is if it has been previously loaded into the building environment and the user has transferred from there to the consulting environment). This can be done by selecting the 'loading' option from the main consulting menu, and the process is the same as that described for the equivalent option in the building environment. Again it is only possible to load a single knowledge base.

6.3.2 Consulting a knowledge base

To consult a knowledge base in the EXPLORES consulting environment, it must first be loaded as described above, after which it can be consulted by selecting the 'consult' option from the main consulting menu. If this is the first time that the loaded knowledge base has been consulted during the current session,

or if there is a chance that it has been altered in any way since it was last consulted in the current session (for example, if the user has left the consulting environment), then the shell will take some time to analyse and summarise it, storing any information which might be needed during the consultation. This is stored in the form of Prolog lists, which will contain, for example, the tags of all of the rules which conclude a value for a particular premise, or the names of all of the things which have a particular feature. Although this analysis takes some time, it need only be done once, after which all of the information which might be needed during a consultation is in place. An alternative strategy would be to do any analysis as it was needed, but it was decided that this would ruin the flow of a consultation.

When the shell has finished analysing the knowledge in the knowledge base, it presents a menu containing the name of the top-level *thing* which it can investigate during the consultation (that is, the *thing* at the top of the hierarchy in the loaded knowledge base). If this is selected from the menu (the other option is the standard safety option), then the consultation begins.

During a consultation EXPLORES attempts to produce a decision based on the static knowledge in the loaded knowledge base and the dynamic knowledge which it acquires during the consultation from information provided by the user and from the inferences made from the rules in the knowledge base (the strategy used for this is described in detail in Chapter 4). When information is required from the user, the appropriate question is displayed on the screen and a list of suggested answers is given. These are all of the possible values for the *variable* for which the question finds a value, that is, all of the values which it has in the knowledge base when it appears in the premises of any *rules* or as a property of any *things*. These values are found and stored during the analysis which takes place at the beginning of the consultation.

When a list of suggested answers is presented, there are two possible courses of action. Pressing the space bar displays the first of the answers in the list as a possible answer to the question. If this is the required answer, pressing the carriage return key selects it: if not, pressing the space bar displays the next

answer in the list. This feature again reduces the potential for typing errors and enables the user to respond more quickly and accurately than would be possible by typing an answer. However, typing is also possible, and the typed answer need not be one which appears on the list. Typing can only be done *instead* of using the space-bar option: once the space-bar has been pressed the only possible answers which can be entered are those suggested by the system. Once an answer has been entered, it is then used by the shell as it tries to deduce the result of the consultation.

One of the most interesting things about the consulting environment is its interface to the explanation tool. At any point during a consultation, typing `explain` in response to a question from the shell (or choosing it from the suggested answers, where it is always given as the first option) will load and run the explanation tool. However, the user can return to the point where the consultation was interrupted at any time. The EXPLORES explanation tool is described in more detail in Section 6.4.3.

6.3.3 Viewing knowledge

We described in Section 6.2.4 how the knowledge currently residing in the building environment could be viewed. This is also possible in the consulting environment, by selecting the 'viewing' option from the main consulting menu, after which the process is the same as that outlined previously.

6.3.4 Unloading a knowledge base

As with the building environment, the restriction of allowing only one knowledge base means that any knowledge currently in the consulting environment must be removed before a new knowledge base can be added. This can be done by selecting the 'unload' option from the main consulting menu. Again, the option of saving the knowledge base is given before it is unloaded.

6.3.5 Additional features

Two other features were initially planned for the consulting environment. These were:

- saving a partial consultation
- loading a partial consultation

The reasons for providing these features arose out of our concern over the lack of time available in schools to carry out a substantial piece of work. The ability to save and then re-load a partially completed consultation would ensure that unfinished work could be returned to during the next session.

Unfortunately, lack of time prevented these features from being implemented. However, discussions with many users of small expert system shells has suggested that these are generally regarded as low priority features for a shell to have, and are rarely used in those shells which do have them

6.4 The Supplementary Tools

In addition to the two environments described above, EXPLORES has a number of supplementary tools. These can be loaded into one or both of these environments to provide a range of extra facilities. Initially five tools were envisaged for the shell. These were:

- a knowledge-checking tool, to provide feedback within the building environment on the correctness and completeness of the knowledge in a knowledge base
- a browsing tool, which allows the user to view the knowledge structures in a knowledge base in a much more flexible way than in the basic viewing facility

- an explanation tool, to provide access to the knowledge in a knowledge base by allowing the user to ask questions about it

- an answer changing tool, which would allow the user to backtrack and change an answer given earlier in a consultation (again, the motivation behind this was mainly concerned with time-saving aspects). The tool would also allow exploration of the possible paths through a consultation

- an advice tool, to act as a support tool for the user. This would provide advice at any point during building or consulting, with regard to the available options for the next stage of the task

Each these tools is discussed in more detail below, although time constraints meant that only the knowledge-checking tool, the browsing tool and the explanation tool were implemented. Of these, the explanation tool is regarded as the most important, and has been the one on which most of this research has focused. We discussed some of its features in the previous chapter: the rest are described below.

6.4.1 The knowledge checking tool

The knowledge-checking tool can be loaded into the building environment when knowledge is being added to a new or an existing knowledge base. Its main function is to provide feedback about the correctness and completeness of a knowledge base.

When new knowledge is being added to a knowledge base there are some errors which can be picked up immediately. We will refer to these as **immediate** errors. Immediate errors are mainly duplication errors, for example where the unique identifier of the knowledge structure being entered has already been used. Some examples of immediate errors would be two rules with the same rule tag; two things with the same name; and two questions which find a value for the same variable.

Thus, two rules of the form:

rule1:
IF *sky = blue*
THEN *colour = red*

rule1:
IF *grass = green*
THEN *colour = green*

would be picked up as an immediate error.

Another group of errors are those which can only be classed as errors when the knowledge base is analysed as a whole. We will refer to these as **potential** errors. Potential errors often take the form of missing knowledge structures, such as a reference to a thing which doesn't exist, or use of a variable name in the premise of a rule, where there is no provision for giving that variable a value during a consultation (that is, it does not appear in the conclusion of a rule and there is no question which finds a value for it). Other potential errors can be logical errors such as circular relationships between things (for example, where `thing1` is described as being an example of `thing2`, and `thing2` is described as being an example of `thing1`).

There are many arguments in the education and ITS literature about the appropriate time to give error feedback to pupils and users. The two extremes for this are that feedback should be given immediately the user strays from the correct path, and that it should only be offered when the user is hopelessly stuck. The former strategy makes no provision for self-realisation of the error made, perhaps followed by backtracking and self-correction: however, it does ensure that a correct solution is always achieved and the user is not allowed to become mired in an unresolvable mess. The latter strategy allows more freedom of movement with fewer constraints, but often does not offer enough support to novice users. This point can be illustrated using three Intelligent Tutoring Systems (ITSs) which use three different correction strategies.

PROUST [Johnson 88] is a system which diagnoses bugs in Pascal programs written by novices. When a complete but buggy Pascal program is submitted

to the system, it attempts to infer the user's intentions and classifies bugs as either faulty intentions or as failed attempts to implement correct intentions. By dealing only with complete programs, PROUST can provide no feedback while the program is being developed. This means that although students can work uninterrupted, and can try out many different solution paths, there is no help available if the student becomes stuck. At the other extreme, Anderson's LISP tutor [Reiser *et al* 85], an ITS for Lisp programming, corrects the student as soon as a deviation is made from the acceptable solution path for the program being developed. Although this ensures that students never become stuck, it also means that they are never given the opportunity to develop their own solutions. The Bridge ITS [Bonar & Cunningham 88] attempts to find a middle course between these two extremes. Bridge provides a tutorial environment for novice Pascal programmers. While using it, students can request feedback on complete and partial solutions. In addition, Bridge can interrupt a student if it perceives that an error is being made. However, unlike the LISP tutor, an interruption only occurs if the system decides that the error is relative to the model of novice programming operations that it perceives the student to be using. If the error is deemed to be unimportant for the student's current level of understanding, then it will be ignored, and the student will not be interrupted. Thus, students can develop their own solutions without constant interruption, but can also request help if necessary.

The feedback strategy preferred differs from user to user, and often depends on the level of skill achieved in the task being performed: experts may prefer to work uninterrupted, while novices may prefer the safety net of a tutor constantly keeping watch over them. As one of the design goals for EXPLORES was that it should be suitable for users with a range of skills and abilities, we decided that the knowledge-checker should not impose any one correction-strategy. Thus, when knowledge is being added in the building environment, there are two possible states for the knowledge checker:

Checker switched on This is the default state of the knowledge checker when knowledge is being added in the building environment. In this mode, the

knowledge being added is checked as it is entered, and any immediate errors are reported as they are found. If the knowledge structure being entered is identical in every respect to one already there, a message will be printed to that effect and the duplicate structure scrapped. If it is only the unique identifier of the structure which is duplicated, the error can be corrected immediately or left until a later point.

If the the error is to be corrected immediately, a menu of correction options is presented, from which the user can display the two duplicated knowledge structures; change the appropriate part of either structure by entering an alternative; or delete either structure. Changing the appropriate part of either structure is complicated slightly by the fact that the change made may duplicate the unique identifier of another structure in the knowledge base. Thus, the entered change is also checked and is disallowed if it has been used already.

Checker switched off The knowledge checker can be switched off by selecting the appropriate option from the adding menu. In this mode, knowledge can be entered without any interruption. However, any immediate errors which are discovered are collected and stored, and the user is told that the knowledge base has errors when the **Finished adding** option is selected from the adding menu. At that point, any necessary corrections can be made, or the knowledge base can be left as it is.

The entire knowledge base can be checked for both immediate and potential errors by selecting the 'Check for errors' option from the main adding menu, and any errors found can either be reported to the screen or saved to a named file. If the errors are to be reported to the screen, any immediate errors found are reported one by one, and the error can be corrected at that point if desired. Following this, potential errors are reported: however these cannot be corrected immediately, as many will involve the addition of new knowledge structures or the alteration of existing ones. The 'check for errors' option need not only be used when the knowledge base is judged to be complete: it can be used on a

partial knowledge base, for example as a means of checking which structures still need to be entered.

The following is a list of all of the potential errors currently found by the EXPLORES knowledge checker. Some of the errors which it finds are 'fatal' errors, that is, it would not be possible to consult the knowledge base while they were present in it: others are less serious. For each type of error, it lists the errors found of that type and where (for example, in which rules) they were found. The potential errors reported by the tool are:

- A list of *things* which are described in the knowledge base but which do not appear to be related to any other *things*.

- A list of *things* which are described as being related to other *things* in the knowledge base but which are not themselves described as *things*. Each *thing* reported has attached to it a list of the *things* to which it is said to be related.

- A list of variable names which have questions to find a value for them but which are not used anywhere else in the knowledge base.

- A list of variable names which are used in the premise of one or more rules but for which there is no way of finding a value (that is, they are not used in any rule conclusions and there are no questions which find a value for them). Each variable name reported has attached to it a list containing the rule tags of all of the rules which use it in a premise.

- A list of variable names which are the names of properties in one or more *things* but for which there is no way of finding a value (again, this means that they are not used in any rule conclusions and there are no questions which find a value for them). Each variable name reported has attached to it a list containing the names of all of the things which have it as the name of a property.

- A list of variable names which are used in the conclusion of one or more rules but which appear nowhere else in the knowledge base. Each variable name reported has attached to it a list containing the rule tags of all of the rules which use it in a conclusion.

- A list of pairs of *things* which have circular 'has member'/'is an example of' relationships, for example where `thing1` is described as being an example of `thing2`, and `thing2` is described as being an example of `thing1`.

- A list of *things* which are described as being examples of more than one *thing*. Each thing reported has attached to it a list containing the names of all of the things of which it is an example.

The strategy used to detect the above potential errors in a knowledge base is mainly list comparison. For example, to find all of the variable names which are used in the premise of one or more rules but for which there is no way of finding a value, the tool constructs and compares three lists:

- a list of all variable names used in rule premises, for example [V1, V2, V3, V4, V5, V6]

- a list of all variable names used in rule conclusions, for example [V1, V6]

- a list of all variable names for which a value can be found by a question, for example [V3, V4]

If all of the variable names in the second and third lists are removed from the first list, any which remain are those for which there is no way of finding a value (V2 and V5 in the above example). This technique is used in CRESS [Ross 89], but is extended here to cover more knowledge structures and more errors.

In all cases it is possible that the error found has occurred as a result of a typing error, for example, a relationship which mentions a thing called *catus* instead of *cactus*, where the latter is actually in the knowledge base. However, it is also possible that there are structures missing and therefore the knowledge base is incomplete.

The knowledge checking tool provides a way in which a knowledge base written in the EXPLORES KRL can be checked for errors. This can be done as knowledge is being added (by having the checker switched on) or at a later point (by choosing the 'check for errors' option from the adding menu). There are two types of errors which can be caught. Immediate errors are those which are immediately obvious when a knowledge structure is added to a knowledge base: these consist mainly of duplication errors, where the unique identifier of the structure being added is the same as one already there. Immediate errors can be repaired when they are caught or at a later stage. Potential errors only become obvious when a knowledge base is analysed as a whole, and comprise errors such as missing knowledge structures or illogical relationships: these cannot be repaired immediately.

The errors found by the EXPLORES knowledge checker in no way comprise all of the possible errors which a knowledge base could contain, and as such it may still be incorrect even if no errors are found by the knowledge checker. In addition to increasing the range of errors which it can find, there are many other possible extensions which could be made to the knowledge checker. For example, it could be made to recognise that errors may be the result of typing mistakes, and could try to find any names in the knowledge base which are similar to the mis-typed one. Another possible extension would be the ability to save the errors in the knowledge base to a file and then to load that file into the editor to be referred to while the knowledge base is being corrected. These and other options are discussed in Chapter 8 under 'Further Work'.

However, even with its current limitations, the knowledge checker can still provide useful feedback on the current state of a knowledge base, with regard to corrections and additions which must be made to it before it can be successfully consulted.

6.4.2 The browsing tool

The purpose of the browsing tool is to provide a more flexible means of examining the knowledge in a knowledge base, over and above the basic 'View knowledge' option available in the building and consulting environments. With that option, it is only possible to see every instance of each knowledge structure in a single, straightforward way. In the browsing tool, each type of structure can be viewed in many different ways, according to the particular aspects which it has. It is also possible to browse the knowledge base for structures containing specific keywords.

When designing the EXPLORES browsing tool, we were influenced by many of the criteria which had served as the basis for the design of browsing tools such as the ECO browser [Robertson *et al* 85] and of intelligent database assistants such as RABBIT [Tou *et al* 83]. These state that the user of such a tool cannot be expected to have more than a vague idea about the type of information which they want from the system, or indeed about what information is available. Thus, the interface must be constructed such that the user need not learn a query language before using the system; and that the kind of information available is made clear, but in a limited or structured way. In addition, the user must be able to move around the system freely, backtracking if necessary, and should have an indication of their position in the system at all times.

The EXPLORES browsing tool is available when the 'View knowledge' option is chosen from within either the building or the consulting environment. When it is first loaded, the browser takes some time to construct and store lists of the variable names used in each of the knowledge structure types in the knowledge base. It then states how many structures there are of each type, for example:

There are: 0 assignments 6 questions 3 rules 10 things

When this has been done, a menu is presented showing the two main browsing options: **Browse knowledge** and **Keyword search**. We will now discuss each of these options in more detail.

Browsing the knowledge in a knowledge base

This option allows browsing of a knowledge base from the general, top level aspects of a knowledge structure down to specific, bottom level details about the features of one particular knowledge structure of a chosen type. At every level, the user is free to move up or down, browsing in a more general or a more specific way as desired. The current level is always indicated in the bottom left-hand corner of the screen. The levels available for this part of the EXPLORES browsing tool are as follows:

Top level At the top level, the main browsing menu is presented, from which the 'Browse knowledge' option can be chosen

Second level a menu is presented offering the *type* of knowledge structure which can be browsed, that is, rules, assignments, questions and things. Only those types which exist in the knowledge base will appear in this menu: for example, if the knowledge base has no assignments, then 'assignments' will not be given as a menu option. The two choices available at this point are to select one of the structure types and move down to the third level, or to move back up to the top level.

Third level when a structure type has been selected, the parts of it which can be browsed are then presented as a menu. Thus, if the chosen type is *questions*, the menu options will be `All questions`, `Selected questions` and `Selected variables`: the options available for the other structure types are similar to this. If the `All` ... option is selected from this menu, then all of the knowledge structures of the appropriate type which are in the knowledge base will be displayed: this is the same as viewing each structure using the standard viewing option. Following this, the parts menu is presented again.

The two other choices available when the parts menu is presented are to select one of the other, more specific parts and move down to the fourth level, or to move back up to the second level.

Fourth level If a more specific part has been chosen, then a menu is presented offering all of the occurrences of that part in the knowledge base. Thus, if the `Selected questions` option is chosen from the question-parts menu, the question-parts of all of the questions in the knowledge base are offered as menu options. If the chosen structure had been rules and the specific part chosen had been `Selected premises`, then all of the variable names from all of the rules in the knowledge base would be offered. From here the user can return to the third level, or go down to the fifth level by selecting one of the instantiated parts from the menu.

Fifth level At this level, the complete knowledge structure containing the instantiated part selected from the fourth level menu is displayed. For most structure types, there will only be one knowledge structure which is appropriate. However, if the instantiated part is a variable name from a rule premise or a rule conclusion, then it is possible that it will appear in more than one rule. If this is not the case, the single rule in which it appears is displayed as before. If it is the case, then a menu will be presented containing the rule tags of all of the rules containing the instantiated part. The complete rule will be displayed if its tag is selected from the menu, and more than one tag can be selected.

The only option available from this level is to return to the fourth level.

Figures 6–2 and 6–3 illustrate the above descent of knowledge structures, showing two of the many possible paths which could be taken through a knowledge base.

Doing a keyword search

The second option available in the main browing menu allows the user to browse through all of the knowledge structures of any type which contain a specific keyword. There are two types of keyword search which are offered as menu options when the 'Keyword search' option is chosen from the main browsing menu: these are **enter keyword** and **look at dictionary**.

Figure 6–2: A possible path for browsing questions

Top level: *main browsing menu*
select `Browse knowledge`
Second level: *knowledge structures menu*
select `questions`
Third level : *parts menu*
select `selected variables`
Fourth level: *instantiated parts menu*
select `'stem type'`
Fifth level: *full structure displayed*

Question: *What type of stem does your plant have*
finds a value for
Item: *stem type*

Figure 6–3: A possible path for browsing rules

Top level: *main browsing menu*
select `Browse knowledge`
Second level: *knowledge structures menu*
select `rules`
Third level : *parts menu*
select `selected premises`
Fourth level: *instantiated parts menu*
select `spine length`
Fifth level: *rule tag menu*
select `rule1`
full structure displayed

rule1:
IF *stem type = branched*
AND *spine length = long*
THEN *spine type = long spines on branches*

If the 'enter keyword' option is selected then the user is prompted for a keyword. When this has been entered, a menu is offered of all of the structure types in the knowledge base, one or more of which can be selected: the selected types will be the only ones which will be searched for the keyword. If the selected type has no knowledge structures containing the keyword, a message will be displayed to this effect. If there is only one structure of that type containing it, such as there would be for questions and assignments, then the complete structure will be displayed. However, if there is more than one structure of the selected type containing the keyword (for example it may appear in the premises or conclusions of several rules) then a rule tag menu is presented as described above.

If the 'look at dictionary' option is selected from the keyword search menu, then an alphabetical menu is presented of all of the variable and property names which appear in all of the knowledge structures in the knowledge base. When one of these is chosen, all of the structures of each type which contain the chosen name are displayed in turn (again, a rule tag menu is used if appropriate).

Summary

The EXPLORES browsing tool offers a variety of ways in which the knowledge in a knowledge base can be browsed. There are two main categories of browsing, and within each category there are many options with regard to how browsing of that type can be carried out. The first category is a level by level examination of the knowledge from the top level (the general types of knowledge structure available) to the bottom level (the specific instantiation of one knowledge structure of the chosen type). There are many routes which this examination can take, and at any level the user can move back up to a higher level or down to a lower one. The second category of use is to browse through all of the knowledge structures which contain an entered keyword: this can be a keyword entered by the user or one chosen from the offered dictionary.

An important feature of the browsing tool is that it does not assume that the user has any prior knowledge of the knowledge base: this is mainly achieved by the fact that, like the rest of the shell, it is menu driven, and that the menus are often generated specifically from the knowledge in the knowledge base. Thus, the user is made aware of the options available at all times.

6.4.3 The explanation tool

The main function of the explanation tool in the EXPLORES shell is to provide access to a knowledge base in such a way that the knowledge in it can be explored, and that potentially, the user can learn about the knowledge domain. The access to a knowledge base which is provided by the explanation tool is very different from that provided by the browsing tool. The latter is concerned with how the knowledge in a knowledge base is structured, and with the actual components of each knowledge structure, and as such it provides many perspectives on the knowledge that is there. The explanation tool provides a view of the various aspects of the *things* described in the knowledge base, in terms of the properties which they have and do not have; the similarities between them; the differences between them; and the ways in which they are interrelated within the knowledge hierarchy. To do this, the explanation tool analyses and synthesises the knowledge that is there and presents it to the user in the form of structured explanations, generated in response to posed questions. To illustrate this point more fully, an analogy from the domain of computer programming would be the difference between looking at the code of an accountancy program in whatever programming language it was written, which is the role the browser plays, and having explained in English the procedures involved in carrying out some aspect of accounting, which is the role of the explanation tool.

As mentioned above, the explanation tool is interfaced to the consulting environment, and can be accessed during a consultation. Within the explanation tool, a knowledge base can be explored by posing questions about the knowledge contained in it, and as with the rest the shell, the tool is largely menu driven. In

addition to reducing input errors, this also means that again the user need have no prior knowledge of the contents of a knowledge base in order to ask questions about it.

On entering the explanation tool, a menu is presented which has as options the four main explanation categories: description, comparison, property analysis and relationship analysis. As discussed in the previous chapter, expert system shells are by nature generic and thus cannot use any input techniques which depend on the specific and limited vocabulary of any one domain, such as MYCIN's free-text questions [Buchanan & Shortliffe 84]. Nor is it feasible to consider having a natural language front-end which would (in theory) allow any question to be asked about the knowledge in a knowledge base. However, a reasonable compromise has been reached in the EXPLORES explanation tool by providing a number of question-templates within a variety of explanation categories, which can be completed by the user as appropriate. This provides quite a bit of flexibility with regard to the ways in which the knowledge base can be explored.

Thus, on selecting one of the explanation categories, a menu of question-templates is presented, containing all of the questions which are appropriate for that category (both the property and relationship categories have sub-category menus which must be selected from before the question-menu can be presented). A complete list of the questions which can be asked in the EXPLORES explanation tool is given in the previous chapter (Figure 5-3). In that chapter we also describe the ways in which a template can be completed and the facilities provided for doing this. When a question-template has been completed, the explanation tool tries to generate an explanation in response to the posed question: we will now describe the strategies for doing this within each of the available question categories.

Describing things

To describe a *thing* the explanation tool first assembles a list of all of the features which it has, both those that are unique to it and those which it inherits. This

Figure 6–4: Example explanation I - description

Describe thing1

thing1 has the properties
foo and
bar

---

foo has the value
blue
red or
yellow

bar has the value
three or
four

---

list is then presented to the user as a descriptive explanation. The structure of a description is such that it moves from the general to the specific, that is, the features that the *thing* has are listed first, and then, for each feature, the possible values for it are listed. Figure 6–4 illustrates the description which would be generated for a *thing* called thing1 which has the feature *foo* with the value *blue, red* or *yellow* and the feature *bar* with the value *three* or *four*.

Comparing things

Within this category, it is possible to compare two *things*, and, more specifically, to ask about the differences and the similarities between them. These questions refer to the features of the two things being compared, again to both those features which they inherit and those which are unique to them. There are three possibilities which can arise when the features of two *things* are compared, in that both things can have:

- the same feature with the same values. These are referred to as **identical features**

Figure 6–5: Example explanation II - comparison

Compare thing1 and thing2

thing1 and thing2 both have the property
foo with value
red

---

thing1 has the property bar
with value
three
thing2 has it with value
four

In addition, thing1 has the property
lah with value
large

thing2 has the property
fah with value
small

- the same feature but with different values. These are referred to as **similar features**

- features which the other *thing* does not have. These are referred to as **unique features**

To answer the question 'Compare X and Y?', a list of the features which each *thing* has is constructed (one list for each thing), these two lists are compared, and three further lists are constructed which correspond to each of the three possibilities described above. The generated explanation is structured such that it first describes any identical features which the two things have; it then describes any similar features; and lastly, it describes any unique features. Figure 6–5 shows the structure of an explanation which could be generated following a comparison between two *things*, namely `thing1` and `thing2`.

Allowances are made when generating the explanation if one or more of the three possibilities does not arise, for example if the two *things* have no identical features.

To answer the question 'How do X and Y differ?', the same strategy is used but the generated explanation contains only those features which are categorised as similar and as unique. Similarly, the explanation generated in response to the question 'How are X and Y similar?' contains only those features which are categorised as identical and as similar.

Properties of things

The sub-category menu for this category has the options 'all properties', 'what things have' and 'what things do'. Within the 'all properties' sub-category, the only question which can be asked is 'What properties does X have?', and the generated explanation lists the features which the *thing* has and the actions which it performs. The structure of that explanation is the same as that generated when a description of a *thing* is asked for, in that it moves from the general to the specific with regard to the properties and their possible values.

Within the 'what things have' sub-category, there are four questions, all of which refer to the features of things. These are 'What features does X have?', 'What things have X?' , 'What things do not have X?' and 'Does X have Y?'. The first of these is answered in the same way as the general question 'What properties does X have?', except that the answer lists only the features which the *thing* in question has, and not the actions which it performs.

The remaining three questions have both a general and a specific form, that is, it is possible to ask generally about a feature, and to ask about it with a specific value. The 'specific value' option is offered once the general part of the template has been completed, and if required a specific value for the property in question can be entered at that point. Thus, it is possible to ask not only questions such as 'What things have *foo*?', but also 'What things have *foo* with the value *red*?'.

In the question 'What things have X?' (either in its general or its specific form), the filler for X will be the name of one of the features which appear in the knowledge base, chosen from a menu listing all of the features there. The strategy used to answer this question involves the construction of a list of all of the things *on the bottom level of the hierarchy* which have the named feature, either as their own or as one which they have inherited, and the generated explanation displays this list. The reason for only listing *things* from the bottom level of the hierarchy is that, from an epistemological viewpoint, it is only those leaf nodes which will represent specific instances of an object rather than general cases or types. Each of these will represent the most definitive description of a *thing* on the particular branch of the hierarchy to which it belongs, and considering only leaf nodes ensures that the generated explanation is better structured, more easily understood, and more accurate. To illustrate this, consider a branch of a hierarchy which goes from `thing1` to the leaf node `thing4`, along the path:

thing1 → thing2 → thing3 → thing4

If `thing1` has the feature *foo* with the value *red*, then inheritance of properties down a hierarchy means that all of the *things* which are descendants of `thing1` will also have *foo* with the value *red*. Thus, if the explanation strategy for the question 'What things have X?' considered *all* the things in the hierarchy, the explanation would read:

thing1, thing2, thing3 and thing4 have foo with value red

Only considering the leaf nodes of the hierarchy gives a much more natural and intuitively-correct explanation, namely:

thing4 has foo with value red

This is particularly important when the knowledge base contains a large number of *things*, or when many of the *things* in it have a particular feature.

If all of the bottom-level things in a knowledge base have the named feature, then the explanation simply says 'All things ...' rather than listing each one by

name. It will never be the case that no things have the named feature: at least one *thing* must have it or it would not appear on the list of possible fillers.

The structure of the explanation generated in response to the question 'What things do not have X (with value Y)?' is similar to that of the previous question. However, in this case, the list displayed is a list of all of the *things* which do not have the named feature. As before, the answering strategy only deals with *things* at the bottom of the hierarchy. A *thing* can have a feature in one of two ways: it can inherit it from one of its ancestors, or it can have it as a unique feature of its own. Thus, a *thing* does not have a feature if none of its ancestor has it, and if the feature is not one of its own properties. It is possible that there will be no *things* which do not have the feature, that is, all *things* in the knowledge base do have it, and again allowances are made for this.

The question 'Does X have Y (with value Z)?' again refers to both the features which a *thing* has as its own and those which it inherits from its ancestors. The answer to this question will either be of the form 'Yes, `thing1` does have ...' or of the form 'No, `thing1` does not have ...': if the former is the case, then the answer will also state which *thing* the feature is inherited from, if appropriate.

In the sub-category menu the option 'what things do' is also offered. This sub-category contains many of the same questions as the 'what things have' sub-category just described: the main difference is that they refer to the actions which *things* perform instead of the features which they have.

Relationships between things

The sub-category menu for this category has the options 'how things are related', 'what things are', 'parts of things' and 'what things do. The first of these is a general sub-category: the last three all correspond to the possible relationships which *things* can have in a knowledge base.

Within the general sub-category, the only question which can be asked is 'How are X and Y related?', and currently this can only deal with relationships of the has member/is an example of kind. The EXPLORES KRL dictates that all

things in a knowledge base must be part of a hierarchy, and that each knowledge base must contain only one hierarchy. If this hierarchy is thought of as an inverted tree, with a single *thing* at the top from which all other *things* branch, then there are two main ways in which two *things* can be related:

- thing1 can be a direct descendant of thing2 or vice versa. That is, both things are on the same branch of the hierarchy, but one (the descendant) is further along that branch than the other.

- thing1 and thing2 are on different branches of the hierarchy, and have a common ancestor in thing3, that is, the point in the hierarchy at which their respective branches split. Thus, thing3 will be nearer the top of the hierarchy than either thing1 or thing2

The strategy for establishing how two things are related involves constructing a path from the *thing* at the top of the hierarchy, to each of the two *things* in question, and comparing the two paths. If these are identical to the point where one of them ends, then the *thing* with the longest path is a direct descendant of the other *thing*. For example if foo is the *thing* at the top of the hierarchy, and the two paths from it to thing1 and thing2 are:

foo → bar → thing1

foo → bar → thing1 → lah → fah → thing2

then thing2 must be a direct descendant of thing1. Figure 6-6 shows the structure of the explanation which will be generated if this is the case.

The other possibility is that the two paths are the same in the beginning, but then diverge, that is, the two *things* are on different branches of the hierarchy. In this case, the *thing* at the point before the divergence is the closest common answer of the two *things* being compared. For example, if again foo is the *thing* at the top of the hierarchy, and the two paths are :

foo → bar → lah → thing1

foo → bar → fah → thing2

Figure 6–6: Example explanation III - direct descendant

How are thing1 and thing2 related?

thing2	is an example of	fah
fah	is an example of	lah
lah	is an example of	thing1

So, thing2 is an example of thing1

Figure 6–7: Example explanation IV - common ancestor

How are thing1 and thing2 related?

thing1	is an example of	lah
lah	is an example of	bar
thing2	is an example of	fah
fah	is an example of	bar

thing1 and thing2 have bar as a common ancestor

then the two things have **bar** as a common ancestor. Figure 6–7 shows the structure of the explanation which will be generated if this is the case.

Within the sub-category 'what things are', there are questions which ask 'What X is an example of?', ' What are examples of X?', and 'Is X an example of Y?'. If the filler for X is assumed to be `thing1`, then generating an explanation for the first two questions involves straightforward look-up of the hierarchy. For the first question the strategy is to look for the *thing* immediately above `thing1` in the hierarchy with which it has an 'is an example of' (or conversely, a 'has member') relationship, and the generated explanation will list this (or will state that `thing1` is at the top of the hierarchy and is therefore an example of no

other *things*). To answer the second question, the strategy involves looking for all *things* immediately below thing1 in the hierarchy with which it has a 'has member' (or conversely, an 'is an example of') relationship, and the generated explanation will list these (or will state that thing1 is at the bottom of the hierarchy and there are therefore no examples of it).

Answering 'Is X an example of Y?' is more complicated: X can be said to be an example of Y if it is either immediately below Y in the hierarchy and the two *things* have a has member / is an example of relationship, *or* if it is an example of Z, where Z is a *thing* which is an example of Y. Thus, X can be descended from Y over several generations. If we assume that the filler for X is again thing1 and the filler for Y is thing2 then the strategy used to answer this question involves trying to find a path from thing2 down the hierarchy to thing1. If one can be found, then the two *things* are related, and the generated explanation outlines the complete path between them: if a path cannot be found, then the two *things* are not related, and the explanation will state this. Thus, if the path between the two things is :

$$\text{thing2} \rightarrow \text{foo} \rightarrow \text{bar} \rightarrow \text{thing1}$$

then the generated explanation will have the same form as that generated when two things are found to be related (see Figure 6–6).

The other two options in the sub-category menu for the 'relationships' category of explanations are 'parts of things' and 'what things do'. These refer to is part of / has part relationships and done to / done by relationships, respectively, and within each sub-category, the available question-templates and explanation strategies are similar to those described above for has member / is an example of relationships.

Follow-up questions

In the previous chapter we discussed why the EXPLORES explanation tool does not use any form of user modelling to try to establish the users level of skill

or current requirements, and as such it cannot tailor the explanations which it generates. As a compromise, it has a mechanism which allows the user to control to some extent, the length, content and direction of the generated explanation by means of follow-up questions. Every time an explanation is generated, a menu of possible topics for these follow-up questions is presented, and, if one of these is selected, the questions which can be asked within that topic appear, again as a menu. Any number of topics and questions can be chosen, and those which are made available depend on the original question asked; the explanation generated in response; and the existence of pertinent knowledge in the knowledge base. These mechanisms are described in detail in the previous chapter.

Summary

The explanation tool for the EXPLORES shell allows exploration of the knowledge in a knowledge base by providing a range of questions which can be asked about it. These come under four explanation categories (description, properties, comparison and relationships) and are initially presented to the user as question-templates. To help in the completion of these templates, menus of appropriate fillers are presented, and once the fillers have been selected, the explanation tool attempts to answer the posed question. For each explanation generated, a number of follow-up questions are available: the topic of these questions can be the *things* and the properties which appear in the original question, as well as those which appear in the generated explanation. The type of follow-up questions which can be asked depend on the knowledge in the knowledge base.

The explanation tool provides domain-based explanations using explanation techniques which are dependent on the structure of the knowledge in the knowledge base, but not on any one knowledge domain. Although it does not permit natural language input, this is compensated for to some extent by providing a wide range of question-templates, which allows flexible question-asking. The generated explanations are reasonably well-structured, and although not tailored to the user by means of user modelling, the provision of follow-up questions on a variety of topics allows the user to control their length, content and direction.

Thus, the explanation tool provides a means by which the user can learn about the domain of a knowledge base through question-asking and exploration.

6.4.4 The advice-giving tool

The advice-giving tool is one of the two tools which was not implemented due to lack of time. The main function of this tool in the shell would have been to provide the user with both general and specific advice about the part of the shell currently in use and about the task being carried out. For example, when adding knowledge, the user could change into 'advice' mode, and, using a menu-driven interface, could find advice about the four knowledge structures in the EXPLORES KRL. At any point, the user would also have access to more general information, such as the overall structure of the shell, the function of each tool, and so on.

In short, the advisory tool could be regarded as an on-line help system which would overlay the shell, and by means of hooks from the shell to the tool could provide access to both general information and information specific to the feature of the shell currently being used. The main problems foreseen with such a tool are related to the problems which are always found with canned text of any kind, such as consistency (any changes made to the system must be backed up by changes to the advice tool), but these are perhaps not insurmountable.

6.4.5 The answer-changing tool

This is the second of the two tools which there was no time to implement. Its main function in the shell would have been to allow the user to change an answer given earlier in the consultation, in response to being asked a question by the system. The main reasons for changing a previous answer would be if the question had been misunderstood, or if the answer given had produced unanticipated results. For example, if the system asked the question 'Do you want to find out about non-flowering plants?', where answering 'yes' would continue the

consultation and answering 'no' would finish it, the user may have answered 'no' thinking that there would be another question asking 'Do you want to find out about flowering plants?'.

At this point, the user would have been able to call upon the answer-changing tool, which would have provided a list of all of the questions asked during the consultation and the answers given. On selection of the one to be changed, a different answer could be entered, after which the tool would work forward through the rest of the consultation, re-interpreting the rules and asking any further questions where necessary. Thus, the answer-changing tool can also be viewed as a means by which the user could backtrack and proceed along another consultation path, and so could explore the knowledge in the knowledge base. This would also have been a useful feature for testing and debugging a knowledge base, allowing the tester to explore all of the possible reasoning paths taken by the system.

Of the five tools initially envisaged for the shell, this seemed the least important, and again, from informal observations of and discussions with users, seems to be something which is not seen as a very important aspect of an expert system shell in the classroom.

6.5 Summary and Conclusions

In this chapter we have described a shell which has been designed and implemented with a view towards educational use. This is the EXPLORES shell, and it provides two separate but linked environments: one for building and one for consulting a knowledge base. The building environment is built around the EXPLORES KRL, which means that knowledge can be added to a new or to an existing knowledge base by following the prompts given for each part of the four possible knowledge structures. In addition, the building environment also has facilities for editing, viewing, loading, unloading and saving a knowledge base. In the consulting environment, a knowledge base can be loaded, consulted, un-

loaded and viewed. During a consultation, the user is asked questions about the current situation, and the answers given are used with the knowledge in the knowledge base to provide the result of the consultation. The two environments can be supplemented by tools which provide facilities for checking the knowledge in a knowledge base for syntax and semantics errors; browsing through the knowledge in a knowledge base using a number of different methods and paths; and exploring the knowledge in a knowledge base by asking questions about it. Two other tools would have provided on-line help and allowed answers to be changed mid-consultation, but these were not implemented.

In Chapter 4 we discussed the extent to which the EXPLORES KRL lives up to the relevant criteria which were set out for it after the initial evaluation (described in Chapter 3). We will now do the same for the other features of the EXPLORES shell. Thus, the shell:

- provides facilities for both building *and* consulting a knowledge base, where the former is a highly structured environment built around the EXPLORES KRL. This allows knowledge to be entered quickly and easily into the appropriate template for each knowledge structure

- can provide feedback with regard to any errors, inconsistencies and incompleteness in the knowledge base being developed, by means of its knowledge checking tool

- has a structure which makes it suitable for users with a range of abilities, in that it provides tools with which the user can tailor each environment to meet his own needs. In addition, it has two very separate modes of use (building and consulting), which means that less able pupils could use the shell for consultation only, and that more able pupils could move on to knowledge base development

- has the potential for use throughout the curriculum as a tool which promotes learning. By this we mean that, in its explanation tool the shell has a facility which allows the user to explore a knowledge base by asking

questions about the domain knowledge contained there. In this way, a number of useful and informative explanations about that knowledge can be obtained, thus providing an environment where the user can learn about that domain

There are two features mentioned in the list of criteria which the EXPLORES shell does not have. The first is the ability to deal with uncertainty: the second is a graphics facility. CRESS [Ross 89], the shell upon which the EXPLORES shell is based, allows rules with certainty factors attached to them, and has facilities for, for example, combining these during a consultation. For reasons of space and time, this feature was not incorporated into the EXPLORES shell. However, there is no reason why it could not be added at a later date: we will return to this point in Chapter 8, under the heading of 'Further Work'. We will also discuss there the inclusion of the second feature: that of having a graphics facility within the shell which could be used for illustrative purposes.

The above are the ways in which we propose that the EXPLORES shell corresponds to the criteria which were previously laid out. In the next chapter we describe the evaluation of the shell by experienced teachers at one of the participating institutions, and discuss the results of that evaluation with regard to the accuracy of these proposals.

Chapter 7

Evaluation of the EXPLORES Shell

In previous chapters we have described the EXPLORES shell, in terms of the initial evaluation carried out to establish the criteria for its design (Chapter 3); its knowledge representation language (Chapter 4); its explanation facilities (Chapter 5); and its overall structure, with regard to the facilities it provides for building and consulting knowledge bases (Chapter 6).

In this chapter we describe the evaluation of the EXPLORES shell, in terms of the methodology used and the results obtained. After discussing the implications of these results, we outline some conclusions which can be drawn from this evaluation with regard to the potential of the EXPLORES shell for educational use.

7.1 The Evaluation

The EXPLORES shell is the shell which has been designed and implemented with regard to the fulfillment of the second of this project's aims, that is, the development of an appropriate expert system shell for use in education and training. The criteria which influenced its design emerged from an initial evaluation comprising an evaluation of existing shells by teachers at the three participating institutions, through questionnaires and interviews; an examination of the potential uses of shells in schools by the same teachers, again through interviews;

an evaluation of existing shells by the author, some through use for building and consulting knowledge bases, and others through published reviews; and a case-study visit to a typical Scottish secondary school. The results of this evaluation provided feedback about the features which were desirable in the new shell and about the possible uses of expert systems and expert system shells in education.

As described in the previous chapter, EXPLORES has an environment for building knowledge bases, one for consulting knowledge bases, and a number of tools with which these can be supplemented. These include a knowledge-checking tool with which the knowledge in a knowledge base can be checked for immediate and potential errors; a browsing tool, which allows flexible browsing of the structures in a knowledge base in a number of different ways; and an explanation tool. The explanation tool allows the user to ask a number of questions about the knowledge in a knowledge base. This is done through completion of pre-stored question-templates within a number of explanation categories, and in response to these the shell can generate explanations containing information about the *things* in a knowledge base, such as descriptions of their properties; comparisons of their properties; and the relationships which exist between them. Follow-up questions are also available, and these allow the user to find out more about specific aspects of the generated explanation. The follow-up questions provided are determined by the original question asked, the content of the generated explanation, and the knowledge in the knowledge base. It is hoped that this tool will provide the potential for users to learn about the domain of a knowledge base.

The purpose of this evaluation is to establish the extent to which users think that the EXPLORES shell fulfills the initial criteria outlined in Chapter 3. One general criterion was that the shell be made simple to use, and it is necessary to establish whether this holds true for each of the features provided. More specifically, it is important to get feedback from users with regard to the shell's:

KRL The initial criteria for the EXPLORES KRL were that it be flexible and expressive enough to allow a variety of types of knowledge to be represented.

In addition, we discussed in Chapter 4 how the KRL was designed in such a way that the user need not consider the inferencing carried out during a consultation when developing a knowledge base, and this should add to its ease of use. Thus, it is important to find out how easy it is for users to represent domain knowledge in the KRL; what they feel its limits are; and how they think it compares to a KRL based on production rules, such as is found in most shells of this type.

Building environment Having pupils either develop their own knowledge bases or add knowledge to an existing knowledge base were two of the three suggested uses for shells in the classroom which emerged from the initial evaluation. For this reason, the criterion was established that the new shell should have facilities for both building and consulting. The former is provided in the EXPLORES building environment, and we must establish whether users feel that this feature helps them to develop a knowledge base in any way.

Consulting environment Consulting a knowledge base is the third classroom use of an expert system shell suggested, and in the EXPLORES shell this can be done within the consulting environment. This has several features which were designed with the purpose of making this task as easy as possible for users, and we must establish whether this is in fact the case.

Knowledge checking tool One important criterion was that the shell should be able to provide feedback about the knowledge base being developed, with regard to any errors and inconsistencies which it contains, and whether or not it is complete. In the EXPLORES shell, this is the task of the knowledge checking tool, and it is important to find out whether users find the tool useful, and whether they understand the type of errors reported and the way in which they are reported.

Suitability for a range of users Given the range of ages, overall ability, and computing skills which can be found in a classroom, it was proposed that

the shell should be suitable for a variety of users. Thus, there needs to be feedback about who in schools would be able to use the shell and how it could be used by different categories of user.

Range of use The main criterion for the design of the EXPLORES shell was that it be suitable for use throughout the curriculum as a tool which would promote learning. We must therefore establish whether users agree with this aim, and consequently whether or not it has been achieved in the design and implementation of the shell.

Having established these aims for the evaluation of the shell, we must now consider the kind of users who would be suitable subjects for carrying out that evaluation. It seemed important that subjects have experience of classroom teaching, both because the shell is designed to be used in schools, and because of one the original aims of the program from which this project emerged (that is, that educational practitioners be involved in the design of educational software). However, some problems arose in the initial evaluation (described in Chapter 3) because the teachers were unable to differentiate between problems caused by the shells they were using and those caused by their own inexperience. For this reason, it was decided that subjects should also have some prior experience of using expert system shells, either for teaching or personal use. On a practical note, it was also necessary that the subjects had access to an RML Nimbus.

For the initial evaluation, we were able to call upon teachers at the three institutions who had agreed to participate in the project. Unfortunately this was not possible for the evaluation of the EXPLORES shell. Since the initial evaluation, contact had been maintained with only one of the institutions (Moray House College), and no further efforts had been made by course organisers at either St. Martin's College or at the Computer Based Learning Unit to introduce an expert system shells component into their Diploma courses. (In fact, the DASE Diploma course is no longer run at St. Martin's). At Moray House, the initial course on expert systems and expert system shells has been expanded since the initial evaluation, and is now a course on Artificial Intelligence, with

an expert system component. This means that there are still teachers at the college who are familiar with the concept of expert system shells, and it was decided that those teachers could participate in this evaluation. In the event, the timing of the evaluation was unfortunate, in that the College teaching term had just ended as the evaluation was beginning, meaning that there would be no teachers available for several months. However, we were able to contact several teachers who had attended past courses, who fulfilled the three criteria, and who were prepared to participate in the evaluation. In addition, we were also able to persuade colleagues who fulfilled the criteria to participate.

We will now describe the method used to evaluate the EXPLORES shell.

7.1.1 Methodology

There were seven subjects in all, with a wide range of teaching and computing experience. None of the subjects were currently teaching in a primary school: they were all currently teaching or had previously taught in mainstream secondary schools, sixth form colleges or Further Education colleges. Thus, the age of pupils taught ranged from secondary school age (11–18) to adults. In addition, one subject had taken part in projects concerned with the production of primary school software.

Between them, the subjects had used a wide range of computer software with their classes, including: word processing packages such as MS Word and WordWise; spreadsheets such as MS Excel; programming languages such as LPA Prolog, LOGO and COMAL (the Scottish Education Board's compulsory language for educational computing from 1986–1989); databases packages such as DATAEASE, QUEST, Masterfile and GRASS; desktop publishing software such as NEWSPAPER and Aldus Pagemaker; compilers and interpreters for languages such as Pascal, Basic and Assembler; and many other software packages for the BBC Micro and the Apple Macintosh. They also had experience of a wide range of small micro-based expert system shells. These included FLEX, ADEX-Advisor,

Expert Builder, ESIE, APES, REFINE, Crystal, I-1, Primex, ESP ADVISOR, ESP Frame-Engine and Xi.

For the evaluation an extensive User Guide was written for the EXPLORES shell, aimed at the teachers who would be evaluating it. This described the shell's knowledge representation language and its main features, and provided users with step-by-step instructions for using every one of it's facilities. At the beginning of the evaluation, the subjects were given a disc containing the shell and two sample knowledge bases (one for plant classification, which can be found in Appendix B, and a very simple one for shape classification), and a copy of the User Guide. Their instructions were to use the shell over a period of time, either on their own or with a class, and to keep extensive notes each time they used it, saying how long they had used it for; what they had done during each session; and any comments which they had. Although the subjects were not given any specific tasks to do, they were given a prioritised list of all the features of the shell which could be evaluated, on the understanding that if time was short, they should use and evaluate those features at the top of the list first. As it was the most novel feature, and the one which we are claiming provides the shell's potential for domain learning, the explanation tool was given the highest priority. This was followed by several aspects of the building environment, to establish how useful users found it. Thus, the features listed were (in order of priority with the highest priority feature first):

- the explanation tool (consulting environment)
- the 'add knowledge' feature (building environment)
- the knowledge checking tool (building environment)
- the 'edit knowledge' feature (building environment),
- any other features in either the building or the consulting environment.

After several weeks, the subjects were sent a questionnaire to complete, which had several sections, each with questions about a different aspect of the EX-

PLORES shell (a copy of this questionnaire can be found in Appendix D). The questions asked were grouped as follows:

The building environment Subjects who had used the EXPLORES building environment to develop a knowledge base of their own or to add knowledge to an existing knowledge base were asked questions about the EXPLORES KRL, on factors such as its ease of use, its expressiveness and its limitations; the built-in editing facilities; the facilities for viewing a knowledge base; and their general opinions about the need for a building environment within a shell.

The consulting environment Subjects who had used the EXPLORES consulting environment to consult a knowledge base were asked about the facility for answering questions using the space-bar to run through the suggested answers; whether they had consulted a knowledge base of their own, and if so, whether they had received any unexpected results; and their opinion of the facilities for viewing a knowledge base.

The explanation tool Here, subjects were asked about the utility and the scope of the available question-templates; the utility of being able to ask follow-up questions; features of the generated explanations, such as their clarity, complexity and usefulness; and their feelings about the potential use of the tool as a means of exploring a knowledge base to learn about its domain.

The knowledge checking tool Subjects who had used the knowledge checking tool were asked how helpful they had found it, and whether or not they had understood the errors reported.

General questions about the shell This section asked general questions about the shell's user-interface; its overall structure; and any features which users would have liked it to provide.

The educational use of shells Here, subjects were asked more general questions about the use of expert system shells in the classroom, with regard to how they could be used; what they had to offer; and how pupils could benefit from using them. Those who said that shells should not be used in the classroom were also asked why not, and if this applied to all shells or was simply a reflection of the fact that the 'right' shell just hasn't been designed yet. If the latter, they were asked what they would consider to be the right shell.

In addition, all of the above sections had questions about the subjects' overall opinion of, and general likes and dislikes about, the feature being discussed. There was also ample space for comments.

We will now discuss the results of this evaluation, with regard to the responses given by the subjects.

7.1.2 Results

We will discuss the results of this evaluation under each of the sections in which subjects were questioned, as outlined above. Although the time spent on the shell varied for each user, these results are based on an average use of around three hours per user.

The building environment

Some of the subjects used the building environment to develop a knowledge base of their own, while others used it to add knowledge to one of the sample knowledge bases provided. One or two had time to try both.

The EXPLORES KRL

EXPLORES provides a KRL based on frame-like structures called *things* and on production rules (it also has two other knowledge structures, questions and assignments). The subjects were asked their opinion of the KRL with regard to how difficult they had found it to represent knowledge about a domain; whether

they felt that it was more or less difficult to use than a language based solely on production rules; and what they felt its limits were.

All of the subjects found it relatively easy to represent their domain knowledge in the EXPLORES KRL, although a few of them commented that this might have been because they hadn't been very adventurous in their choice of domain: they had 'stuck to domains similar to the given one'. In addition, they all said that they had found the language easier to use than a straightforward production rule language. With regard to the limits of the KRL, most of the subjects commented that it would only be useful for domains which could fit into a hierarchical structure, saying that it 'might be hard for more esoteric domains', and that it reached its limits with 'flat domains'. One subject commented:

> As with most frame systems I would expect the KRL to do well when subject matter is easily organised as a hierarchy. Not too good otherwise.

Editing a knowledge base

Subjects were asked for their opinion of the simple editing facility provided by the EXPLORES building environment. Reactions to it were mixed: some subjects liked it and found it easy to use, while others found it more difficult. One subject commented that he 'would have preferred it to be part of the adding-knowledge feature'.

Viewing a knowledge base

The subjects were also asked for their opinion of the viewing facility (provided in both the building and the consulting environment). By and large this was thought to be 'fine', although one user commented that 'with a large knowledge base it would be tedious without some kind of search facility', and another thought that 'multi-windowing would be better'.

Overall

The final question in this section asked users whether they felt that the building environment helped them to build a knowledge base, or whether it was no better than using a standard text editor. They were also asked what it was about it that they found helpful and unhelpful.

Most of the subjects judged that the building environment was helpful, and was better than using a text editor to develop a knowledge base, although one commented that he 'would like the option of a text editor too, as I'd soon get fed up with the prompts'. One of its main benefits was seen as being that the user need not learn the shell's KRL, as all the information needed is provided in the form of prompts. In addition, some subjects particularly liked the fact that, by being built around the KRL, it was able to give guidance with regard to the knowledge already in the knowledge base. For example, when knowledge such as one of the features of a *thing* is being added, the user can ask to see a menu of all the other features which have been added previously. This feature was judged to be 'very helpful'.

The consulting environment

All of the subjects used the consulting environment to consult at least one of the sample knowledge bases, and some of them used it to consult a knowledge base which they had developed themselves in the building environment.

Space-bar option

The EXPLORES consulting environment has a facility whereby users can respond to the shell's questions by using the space-bar to run through the suggested answers, and the carriage return key to select one of these. Alternatively, answers can be typed in. The subjects were asked for their opinion of this feature, with regard to whether or not they found it useful, and what they liked and disliked about it.

Many of the subjects commented that they had found the feature difficult to use initially, for example because it wasn't 'obvious' what to do, but most found that it was 'useful once [they] got used to it'. One subject found it beneficial because she 'didn't have to type anything and it narrowed down options', and this feeling was also echoed by some of the others. However, one subject commented that he would 'prefer a mouse', while another said that the feature was 'better than free format but not as good as dialogue/scroll menu style'. The first item

on the suggested list of answers is the **explain** command, which places the user in the explanation tool, and some subjects found that they continually selected that response by mistake.

Consulting own knowledge base

If subjects had consulted a knowledge base of their own developing, they were asked if the answers received as a result were what they had expected, and, more importantly, were correct. They were also asked if the errors found resulted from mistakes which they had made, or from the inferences made by the shell. Almost all of the subjects in this category found no errors: the one subject whose knowledge base did contain errors said that they were due to mistakes he had made in the knowledge representation (he had not checked the knowledge base for errors using the knowledge checking tool).

Overall

Overall, the comments received about the consulting environment were positive, although several subjects thought that 'the input routine needs attention'. The environment was judged to be most useful 'for a well defined domain e.g. where the answers to questions are easy to provide', but rather less useful for domains where the user would have to respond to questions by guessing or by entering 'not known' (the shell has no facility for dealing with unknown data). Some subjects commented that one of its most useful features was that it allowed the user to consult a knowledge base without having to know how the knowledge was represented, or indeed what knowledge was represented.

The explanation tool

All of the subjects used the explanation tool to a greater or lesser extent (it was listed as the highest priority option for evaluation), both with the sample knowledge bases and with ones which they had developed.

Range of questions

Subjects were asked about the range of questions offered by the tool, with regard to whether or not the questions provided allowed them to explore a knowledge

base in a useful and informative way; and whether there were any questions which they would have liked to ask but couldn't. All of the subjects found the explanations useful and informative, and none had any additional questions which they would have liked to ask, although one commented that other questions might have emerged with 'more use and a bigger knowledge base'. Another thought that 'the template question approach is great for a hierarchical knowledge base', and an interesting point was raised with regard to whether or not teachers would be able to add to the list of templates.

Follow-up questions

Subjects were also asked about the utility of the follow-up questions provided, and again there was unanimous agreement that they were very useful and informative, and that the feature was a good one to have.

Explanations generated

In response to questions regarding the clarity and utility of the generated explanations, comments were mainly positive, saying that they were useful and reasonably clear. However, one subject stated that the explanations only really became clear 'once the structure of the knowledge base was understood', and that for this it had been 'useful to use the viewing option as well to give an overall picture'. The only really negative comment came from a subject who thought that the language used was not 'natural' enough, and that the generated explanations would be better if they contained 'more straightforward English'.

Learning

The main function of the EXPLORES explanation tool is to provide a means where users can explore and consequently learn about the domain of a knowledge base. Thus, subjects were asked if they thought that this was the case. All replied positively, saying, for example, that 'it has good potential for this', although again one had reservations with regard to the language used, saying that it would be successful 'only if the explanations are in better English'.

Overall

Subjects were asked to list the overall factors which they liked and disliked about

the explanation tool. One user disliked the fact that the tool had 'no way of getting how and why explanations', and another thought that the presentation of the generated explanations was a problem, in that they were 'sometimes ... poorly laid out'. On the positive side, subjects liked the explanation categories provided, and 'liked the idea of investigating the links and relationships in a knowledge base'. One liked the fact that it not only 'showed the kind of answers needed in a consultation' but implicitly showed *why* those answers were needed, although again this statement carried the caveat that 'it helped more if you had an overall picture of the knowledge base'. Another subject judged it to be 'a very powerful idea', and there was unanimous agreement with the main design principle behind the tool, that is, that it be used throughout the curriculum to promote domain learning.

The knowledge checking tool

All of the subjects who developed their own knowledge base or who tried to add knowledge to an existing knowledge base used the knowledge checking tool. They were asked whether or not they found it helpful; whether they understood the type of errors reported and the way in which they were reported; and generally what they liked and disliked about it.

Almost all of the subjects who used it found the knowledge checking tool to be very helpful, and one went so far as to say that she had found it 'invaluable'. (One subject said that he had used the tool only briefly , that it 'seemed OK', but that he had 'no strong feelings about it'). The errors reported were judged to be clear and easily understood: for example, one subject commented that the 'explanations of why there was insufficient data given and of what to give were very clear'.

Overall, subjects were very positive about the tool: for example, one 'liked it a lot' and another commented that it 'seemed essential'. On the negative side, one user disliked that fact that it wasn't possible to access the knowledge checker from within the editor.

General questions about the shell

The subjects were asked some general questions about the interface and the structure of the shell; about features which they would have liked the shell to have; and about their overall opinion of it, with regard to features that they liked and disliked.

Interface

Many subjects expressed dissatisfaction with the shells interface, citing several improvements which could be made, such as the inclusion of a full WIMP environment and more use of colour. One subject found the text to be 'too small'; another found the 'text layout too confusing sometimes'; and another had problems because 'the keyboard is not disabled while files are loading'. (This meant that any keys pressed accidently were processed once the files had been loaded). Positive comments about the interface were mainly related to the fact that like the rest of the shell it is largely menu-driven: for example, one subject liked it because she 'didn't have to type much', another because they were 'quick to use', and another liked the ease of recovery if a mistake had been made, saying 'recovery was easy - if I found I'd done the wrong thing I could get back easily and start again'.

Structure

The EXPLORES shell is structured such that developing and consulting a knowledge base are done in two separate but linked environments, and subjects disagreed strongly about the desirability of such a separation. For example, one said that she would 'prefer building and consulting in one', because it was 'time consuming going from one to the other', while another commented that 'it seems a good design', because

> you can isolate ... the consulting environment from the rest if you want to, depending on what you actually want the students to do. On the other hand, if you want them to ... get experience of consulting and follow that up by building up classification skills in a new domain or adding extra knowledge, then it is easy to go from one to the other.

Other features

Subjects were asked about other features that they would have liked the shell to have, with regard to features which they had seen on other shells, and to ones which were part of their personal wish-list but which they had not necessarily seen implemented elsewhere. Several subjects said that they couldn't think of anything, but others responded with features such as HOW and WHY explanations; an override facility for inherited properties in a knowledge base; a graphical representation of the KRL; having the explanation tool offer to show the user relevant parts of the KRL relevant to the generated explanations; and the provision of a Tutorial Guide as well as a User Guide, with specific exercises for the novice user to work through.

Overall

The feature of the shell which was disliked by most users was its interface (see above), for example one subject said that it

> would have been marginally acceptable in the mid-80's. Schools wouldn't tolerate it nowadays. I've tried not to let my judgement of the important concepts be coloured by my dislike of it.

Many subjects also disliked the occasional slowness of the shell, commenting that 'it can be time-consuming to load tools and change between environments'.

Features which subjects particularly liked were the KRL; the knowledge checker; and the explanation tool. With regard to the latter, one user commented that it 'provides a very powerful way of investigating a knowledge base', and another said that 'if [its] presentation and answers could be improved [it] would have some potential for use in computing and other subjects'.

Although some subjects thought that the shell would need to be made more robust if it is to be used in the classroom, overall many were positive about it, with comments such as 'I think EXPLORES has potential despite the interface'

The educational use of shells

In the final part of the questionnaire, subjects were asked whether or not they consider that there is a place for expert system shells in the classroom. Those who responded positively to this question were asked what they see as the role of an expert system shell in the classroom; how they think shells could be used; whether they think shells have anything to offer which can't be found elsewhere, or are simply a way of automating something which can be done by another method; and how they think pupils could benefit from using them. Subjects who reponded negatively would have been asked why, and whether they felt that this was the case for all shells or that the 'right' shell just hasn't been designed yet. However, in the event all of the subjects responded positively to the original question.

Subjects saw the shells in several different roles, such as tools for teaching knowledge organising; for teaching logic; for developing communication; for research; for essay writing; for project work; and for problem-solving. They were seen as being useful for teachers, who could 'use them diagnostically for testing how well concepts have been understood', and for 'supporting other ways of learning about domains which lend themselves to this kind of treatment'. With regard to their use by pupils, all subjects regarded consultation as presenting few problems, but a few expressed doubts about whether pupils could develop their own knowledge base from the beginning: many thought that 'adding to existing knowledge would be worthwhile' as a compromise.

Expert system shells were seen as having much to offer in the classroom, such as a means of 'making children think about information in a novel way'; a way of 'making children think on a meta-level'; providing the motivation to 'research. identify and organise knowledge'; giving a way of testing solutions; providing good possibilities for group work; providing 'an additional way of finding out about things and of developing problem solving classification skills'; and providing 'runnable representations of knowledge with concrete feedback on gaps and inconsistencies'. However, one subject commented that 'I don't think we have

yet thought of all the possible educational opportunities. This will only come from classroom trials and experience'.

In addition to the benefits implicit in many of the above, subjects also thought that shells were useful to pupils as 'another resource', in that they 'provide variety'. This was seen as important, because 'the more ways children can come at something the better', and so 'variety of media is a good thing'. One subject though that they 'might enthuse children about finding out in a way books don't', with the proviso that 'it would be good if they were then led to books as well'.

Finally, one subject commented that although 'the right shell hasn't been designed yet', he felt that 'the EXPLORES shell hopefully takes us a step on'.

7.1.3 Discussion

Before going on to discuss the results of this evaluation, it is necessary to comment on certain aspects of the methodology. Before any claims can be made about it, ideally a piece of educational software such as this should be evaluated in the classroom, with data collected on its use by teachers and pupils in a number of subject areas over a long period of time. In addition, this also allows the possibility of incremental development, with on-site testing. However, such an evaluation involves a tremendous amount of time and effort. With regard to this project, it would have involved the development (by the researcher) of several knowledge bases in different domains, linked to the material being presented by the teachers involved (there is no room in a school timetable for any additional material, and teachers cannot be expected to develop their own knowledge bases). Related to this, teachers would have to be equipped with other teaching materials, and given time to learn how to use the shell (for example on one or more InService courses). For pupils, another User Guide would have had to be developed, aimed at their level of comprehension and containing material such as worksheets and tutorial exercises. In addition, the shell would have had to be made robust enough to withstand extensive use by novices. The time constraints of this project were such that it was impossible to carry out such an extensive

evaluation, and a compromise had to be reached with the methodology described above.

It also proved difficult to persuade teachers to act as subjects. Initially there were another four teachers involved, and although they were keen in the beginning, they all returned the shell after a few weeks, saying that they could not find the time to evaluate it. This was not unreasonable, as they were already having to cope with a full teaching load, and for the most part did not have access to a Nimbus outside of school hours. In the event we were left with seven subjects, and although this is not a sufficient number of people upon which to base any strong conclusions about the shell, the subjects involved were all experienced enough in the field to be able to give a general picture of how the shell would be likely to be perceived by others. Their responses can also provide a basis for further development of and a more extensive evaluation of the EXPLORES shell. These topics are discussed in more detail in the next chapter under 'Further Work'.

Another problem which must be mentioned is the fact that not all of the supplementary tools which were developed could be evaluated. At the time the evaluation was being carried out, the browsing tool was unavailable, as earlier changes to other parts of the shell had affected the running of the program. As the evaluation could not be delayed until the browsing tool was again available, it was necessary to evaluate the shell without it. Thus, we have no feedback from teachers about that tool. However, this was not regarded as a major problem, as the browsing tool is one of the lower priority features of the EXPLORES shell. It is very much a peripheral tool, and its absence in this evaluation does not affect the overall conclusions about the educational use of the shell.

The feedback received about the EXPLORES building environment suggests that overall subjects were positive about having an environment based on the specific KRL of the shell, and found it easier to develop a knowledge base than they would have using a text editor. Users liked the fact that it can provide guidance about what to do next and to the knowledge which is already present in the knowledge base. However, it is possible that more experienced users

would actually prefer to develop a knowledge base using a text editor (such as the subject who felt that he would be irritated by the prompts after a while), and it would be possible to have this an option in future versions of the shell. We will return to this point in the next chapter.

The KRL was judged to be easier to use for knowledge representation than one based on production rules, although it is not clear how much of this is due to the KRL itself or to the fact that the provision of a building environment means that the user need not learn the exact syntax of the KRL, and is prompted for each part of the knowledge structure being added. One user suggested the inclusion of an override facility, but we have already discussed in Chapter 2 how this can lead to problems such as logical inconsistency [Brachman 85].

Overall, subjects judged it to be good for domains which were inherently hierarchical in nature, but felt that it was limited to those domains. This is not an unreasonable constraint, in that no representation formalism is suitable for all types of knowledge. The problem could be solved in part by providing more formalisms and knowledge structures, which would allow other types of information such as causal knowledge to be represented. However, this only adds another level of complexity to what is already a difficult task, and we would propose that the EXPLORES KRL is adequate for the type of use proposed. In addition, many of the subjects taught in school curricula are ones where at least some aspects can be made to fit a hierarchical representation.

The EXPLORES editor is fairly basic in that it lacks facilities such as character-by-character editing, having instead whole-line replacement. From the feedback given, it is not possible to state specifically that subjects did, or did not like it: it received positive comments from several, who found it easy to use, and negative comments from others, who found it difficult. It is perhaps useful to restate here some conclusions stated in the previous chapter. Despite its faults, the EXPLORES editor is at least an improvement on that provided by other small shells, in that it allows the user to view the knowledge which can be edited; it is built around the specific KRL of the shell; it is menu-driven; and the contents of the menus provided are largely dictated by the structure of the KRL *and* by

the specific knowledge in the knowledge base. However, in light of the negative comments received, we will discuss possible improvements to the editor in the next chapter.

Little of note was said by subjects about the viewing facility offered in both the building and the consulting environment. However, the comment regarding the fact that the facility could prove to be tedious for a large knowledge base with many different structures is an important one. It suggests that there is a need for a more structured and selective way of looking at the knowledge in a knowledge base, such as that provided by the browsing tool. Unfortunately, this tool was not evaluated for the reasons outlined above.

By and large subjects were satisfied with the consulting environment. However the mainly negative comments regarding the space-bar option for selecting one of the suggested answers during a consultation, suggest that this is not a successful feature. Instructions on how to use the facility are given each time the user is asked a question, in the form of three lines of text at the top of the screen. These instruct the user to press the space-bar to see the suggested answers, and to press the carriage return key to select an answer. Despite this, many subjects had problems with the facility, perhaps because it is not one which appears anywhere else in the shell, and is not immediately obvious in the way that a menu- or a mouse-based selection method would be.

This suggests that it would be better to alter this facility, perhaps by providing menus, which would still provide many of the benefits cited by subjects for the space-bar option (such as the narrowing down of options and the fact that it saves the user typing what can be long answers). Including a safety option as the first item on the menu (as is the case throughout the shell) would also eliminate a common mistake made by many users: that of ending up in the explanation tool through hitting a key in error. Subjects were satisfied with the answers given when they consulted a knowledge base of their own, which implies that the system is making the correct inferences from its static and dynamic knowledge, although this would have to be tested more extensively using larger, more complex knowledge bases.

In the previous chapter we cited some of the literature on the design of browsing tools and intelligent database assistants, with regard to their design criteria. One of these was that the user could not be expected to know what she wanted to find out, or indeed what there was to find out, before using such a system [Robertson *et al* 85, Tou *et al* 83]. Although this was discussed with reference to the design of the EXPLORES browsing tool, it is a criterion which holds good for the rest of the shell, and is one which users commented on specifically with regard to the consulting environment: they liked the fact that they did not need to know what knowledge was represented or how it was represented before being able to consult a knowledge base.

The shell has no facility for dealing with missing data, and this was commented on by two subjects, who suggested that it might be useful for the user to be able to reply that the information requested is 'unknown', when asked a question during a consultation. This would allow the system to produce answers which were suggestions with provisos attached, such as

> The answer might be peanut cactus. However, peanut cactus has the property 'red flowers' with the value 'yes', and a value could not be established for this property during the consultation.

In addition, EXPLORES also has no facility for dealing with uncertainty, although the shell upon which it was based (CRESS) allowed certainty factors to be attached to rules, and had facilities for combining these during a consultation to establish an overall certainty for a conclusion. Although uncertainty handling was established during the initial evaluation as a desirable feature, we judged that it was less important in a KRL of this type than in one based on production rules, because it is designed to solve classification problems based on specific facts about the objects in a domain. Both of these features are ones which will be discused further in the next chapter, as possible additions to the shell.

There were two tools available for subjects to evaluate, and there were many positive comments about both of them. The explanation tool was listed as the highest priority feature for the evaluation, and feedback was received from all of the subjects on various aspects of it. Comments about the range of the question-

templates were all positive, and subjects were happy with the scope that these provided for exploring a knowledge base, although again this requires further testing with a larger and more complex knowledge base. The suggestion that teachers be allowed to tailor the shell to their own needs by adding question templates of their own is an interesting one. However, it is hard to see how this would be possible, given that each template has attached to it a strategy for generating the appropriate explanation, some of which are fairly complex (several of these strategies are described in the previous chapter). The extra scope for exploration provided by the follow-up questions also received many positive comments. In Chapter 4 we suggested that follow-up questions provide the potential for extending the shell's role to that of a 'tutor' which could attempt to correct apparent misconceptions in the user's model of the domain [McCoy 84]. Although overall they were regarded by subjects as a very useful feature in their present format, we would propose that such an extension would greatly enhance their utility in the shell.

One comment concerned the fact that the explanations generated by the shell became more clear once the structure of the knowledge base was understood. For this, the subject had used the viewing facility. However, these and other comments suggest that the browsing tool could be a useful complement to the explanation tool, in that it provides better access to the structure of a knowledge base than the standard 'viewing' facility, and thus can serve to give a better picture of the structure of the hierarchy of *things* contained there. The browsing tool could also be a solution to the suggestion that the explanation tool be able to show relevant parts of the knowledge base after generating an explanation. Although much of the information that this would provide is available through follow-up questions, the browsing tool would provide a way for users to see the specific knowledge structures involved instead of a translation of these.

Almost all of the subjects felt that the generated explanations were useful and reasonably clear, and that from them pupils could indeed learn about a knowledge domain. However, one subject was more negative, saying that they would only be of use to pupils if the language and presentation were both im-

proved. Despite the fact that this was a minority opinion, it cannot be ignored, particularly given the small number of subjects involved. The utility of the EXPLORES explanations and the potential which they provide for learning is something which will only become clear if the shell is used in the kind of 'ideal' evaluation described above, where pupils could be tested in some way on what they had learned about a domain as a result of using the shell.

The shell has no facility for generating HOW and WHY explanations during a consultation, and this was an addition which one user mentioned. CRESS, the shell upon which the inference engine and the KRL of EXPLORES are based, has a facility for generating these during a consultation which was removed when EXPLORES was developed. This was mainly done to save memory, as it is computationally quite expensive, in that a record must be kept of the entire consultation to date. It was also regarded as a low priority feature in the first instance, as it is one which is available in almost every other shell. However, the code for the CRESS facility could be easily adapted to take into account the inferencing strategy and additional knowledge representation formalism of EXPLORES.

Overall, subjects agreed with the principles behind the design of the explanation tool, and were very positive about the idea of using an explanation facility such as this to help promote learning, even if they had reservations about some aspects of its implementation.

The second tool evaluated was the knowledge checking tool. This was developed mainly as a result of the observation that most users of expert system shells have problems developing a knowledge base. We argued in Chapter 3 that if building is to be one of the two main uses of the shell, then it should provide facilities which help make that task easier. The knowledge checking tool is one of these facilities, and judging by the subjects' responses, it is a relatively successful one. The the tool cannot be accessed from within the editor, and this is something which must be considered for future versions of EXPLORES, as is the fact that the editor cannot be accessed when knowledge is being added to a

knowledge base using the 'add knowledge' option. We will return to this point below, when we discuss possible changes to the structure of the EXPLORES shell.

For reasons such as the elimination of typing errors and the increase in speed of use provided, EXPLORES is largely menu-driven. This is almost the only feature of the shell's interface which received positive comments from subjects. The negative comments received concerned the fact that EXPLORES has no facility for using a mouse or icons; it has no graphics facilities; it makes no use of colour; and although it uses windows, these are really only text boxes and not windows such as would be found in a system with a full WIMP environment (scrollable windows with features such as move, resize, open, close and shrink). Unfortunately this is something over which we had no control, as we were limited by the facilities provided by the hardware (an RML Nimbus running MS-DOS) and the programming language (Arity Prolog). Even then, the menus and windows used by the shell had to be implemented as C-routines and linked to the Arity Prolog interpreter [1] .

There is a Microsoft windowing package for the RML Nimbus, but the version available when EXPLORES was being implemented had limited facilities, and would not have been a substantial improvement on the ones used. One drastic solution would be to reimplement the basic ideas of the shell (which subjects were positive about) on another machine offering a full WIMP environment, such as the Apple Macintosh (an idea suggested by at least one of the subjects). In addition, it would also be possible to make use of the graphics facilities provided by such a machine. This could be used to provide illustrations for the user during a consultation, related to both the questions being asked and the conclusions being presented. In addition, it would also be possible to present the contents of a knowledge base graphically, and to allow the user to add knowledge using simple graphics tools. This was one of the initial criteria laid out in Chapter 3, and is one which we were not able to fulfill in the implementation of EXPLORES.

---

[1] This was done by Dr Peter Ross in the Department of Artificial Intelligence at the University of Edinburgh.

In that chapter we also described a rather unsuccessful implementation of such a facility in Expert Builder. However it is a feature which, if implemented properly, could be very useful in an educational shell.

One complaint concerned the fact that the keyboard is not disabled while files are loading, and this goes back to a point made earlier in this discussion, concerning the need to make the shell robust enough for classroom use. By this, we mean making it less sensitive to 'incorrect' use by implementing safety measures such as that suggested. Although this can be done to some extent, the exact measures needed is again something which would only become clear after extensive classroom use, where it would be possible to observe what users are likely to do. A related issue concerns that fact that the shell could not be used in schools without first developing teaching and support materials for both teachers and pupils, such as the Tutorial Guide described above.

Many of the negative comments about the structure of the shell seem to have stemmed from its slowness: it takes some time to load any of the supplementary tools; to move from one environment to the other (via the control box); and to move between different tasks in an environment (such as from adding to editing). This is something which would have to be improved if the shell is to be used in the classroom, where time is limited and users cannot be expected to endure lengthy waits. The slowness of the shell is partly due to the limitations of the hardware. However, its speed could be improved to some extent by optimising the Prolog code, using strategies such as the implementation of repeat-fail loops instead of recursive loops (which are computationally costly in Arity Prolog). In addition, it would also be possible to compile much more, if not all of the code: currently, only a small part of it is compiled, and many files are loaded at run time. These and other measures would make the shell run quite a bit faster, and would perhaps make its structure more acceptable.

Some subjects were more positive about the shell's structure, and liked the fact that the tasks of building and consulting are carried out in two separate environments. This seems particularly important in light of the fact that many subjects both in this and in the initial evaluation expressed reservations about the

ability of younger or less able pupils to build their own knowledge bases, but were confident that such pupils would benefit from consulting pre-built knowledge bases. Thus, it seems important that the two tasks be kept apart, which suggests that this aspect of the shell's structure should remain as it is. However, other subjects complained that it was not possible to use the editor while adding knowledge, without leaving the adding environment and going into the editing environment. Neither is it possible to use the knowledge checker from within the editing environment. The first problem could again be improved slightly by making it quicker to change between tasks such as editing and adding: at the moment the user must go through the main building menu, and this can take some time. However, both of these problems suggest that we should give serious consideration to how related parts of the shell link together, and how this can be improved.

Some of the findings from our initial evaluation were confirmed, in that subjects were very positive about the potential use of expert system shells in education, suggesting many possible roles for them. It was also widely agreed that they have a lot to offer as a learning resource, and could be used by pupils for both building and consulting knowledge bases (although there were reservations about building for certain pupils). On the more specific issue of whether or not EXPLORES has potential as a useful classroom resource, subjects responded positively overall, particularly with regard to the ideas behind the design of the shell, if not to some aspects of its implementation, and we would suggest that this is as much as can be expected in a project of this type, given the time available and the number of people involved.

7.2 Summary and Conclusions

In this chapter we have described the evaluation of the EXPLORES shell by seven people, each of whom had teaching experience in schools and colleges and experience of expert system shells. As with the initial evaluation described in Chapter 3, this evaluation was formative in nature. Subjects were given a copy of the shell, a User Guide, and a list of features in order of their evaluation priority. They were asked to use the shell as much as possible, and to keep a log of what they had used it for and for how long. After some time, the subjects were given a questionnaire, with questions on many aspects of the shell, such as the building environment, the consulting environment, the knowledge checking tool and the explanation tool.

Despite the problems mentioned at the beginning of Section 7.1.3, overall the results of the evaluation were encouraging, and have led us to reach the following conclusions about the features of the EXPLORES shell and its potential for use a tool to promote learning in a variety of subject areas throughout the school curriculum:

- the EXPLORES KRL provides a simple way of representing domains which are inherently hierarchical or which could be made to fit a hierarchical framework. For these domains, it is better then a KRL based on production rules: users find it easier to represent domain knowledge than they would with a production rule formalism. However ...

- ... this could in part be due to the EXPLORES building environment, which provides support specific to the shell's KRL in a way that a text editor cannot. Although some aspects of this environment (such as the editor) require improvement, overall it is a worthwhile inclusion in the shell, and it removes at least some of the burden of knowledge representation from the user. For users who do not require this, there should be a facility which would allow a knowledge base to be developed using a text editor, outwith

the shell. For this, it would be necessary to develop a more formal syntax for the EXPLORES KRL.

- the consulting environment appears to produce correct conclusions from the knowledge which it has during a consultation. However, a better method should be found for allowing the user to enter information during a consultation. Rather than the current space-bar option, this should probably take the form of menus, to be consistent with the rest of the shell.

- the knowledge checking tool provides invaluable support for the development of knowledge bases, by providing clear and informative feedback about any errors and inconsistencies found, and about its completeness. In view of the positive comments received, we suggest that it should be expanded to cover more errors and thus provide even better feedback (currently it covers only a subset of all of the possible errors in a knowledge base).

- the explanation tool has no facility for generating system-based explanations, and this should be added in future implementations. However, it does provide a range of questions whose scope permits the effective exploration of the domain knowledge in a knowledge base. Although the presentation of the explanations generated in response to these questions perhaps needs to be improved, the tool has the potential for promoting learning in a variety of domains.

- the provision of a building and a consulting environment means that the shell is suitable for all three of the uses suggested: pupils developing a knowledge base from the beginning; pupils adding knowledge to an existing knowledge base; and pupils consulting a pre-built knowledge base.

- the separation of the consulting and building environments provides a means whereby users with a range of abilities can use the shell for different purposes. However, the transition between these environments should be improved, not least by speeding it up.

- the speed and robustness of the shell must be improved before it can be used in schools: it is currently too slow and too dependent on 'correct' use to be any good in the classroom.

- the interface of the shell has several problems, some of which could be solved in this implementation and others which could only be solved by reimplementing EXPLORES on a machine with a more user-friendly WIMP environment. However, there is no doubt that the shell would be improved by the inclusion of features such as colour, graphics, an input device such as a mouse, better windows and clickable icons.

- expert system shells have a place in the classroom as a learning resource, and there are many possible roles which they could assume there. In addition, they offer much which cannot be found in other resources. However, it is as yet unclear what kind of shell would be ideal for educational use.

Even if EXPLORES is not the ideal shell for effective classroom use, it does not seem unreasonable to claim that it is a step in the right direction, that is, towards a shell for use in education whose design is based on a serious consideration of educational issues and practicalities.

Chapter 8

Conclusions and Further Work

In this chapter we state the conclusions which can be reached about the research described in this thesis, and describe further related research which could be carried out. The aims of of this research project were to:

- explore the potential of expert systems in education, the characteristics of existing expert system shells, and the value of these tools as perceived by experienced teachers

- define, and where necessary develop, appropriate expert tools (software and documentation) for use in education and training

To assess the extent to which we have achieved those aims, we begin by restating the criteria which were established for the design and implementation of the new shell during the initial evaluation, and by listing some additional practical constraints involved. We follow this with a brief description of the main features of the EXPLORES (EXPLanation ORiented Expert System) shell, the expert system shell developed during the project, and our conclusions about the extent to which this shell fulfills these criteria. We end this chapter with a discussion of some further research which could be carried out, with regard to changes which could be made to the shell; to the introduction of the shell into schools; and to the further development of facilities for the generation of domain-based explanations in expert system shells.

8.1 The Design Criteria

From the initial evaluation there emerged some criteria which we proposed should influence the design of the new shell. These were that the new shell should:

- have a knowledge representation language which is flexible and expressive enough to allow a variety of types of knowledge to be represented
- be designed such that the inferencing done by the system during a consultation does not affect the structure of a knowledge base, in that it should not be a factor which users need to consider when representing knowledge
- have facilities for both building and consulting a knowledge base, where the former should be highly structured such that knowledge can be entered easily
- provide good feedback about the knowledge base being developed, with regard to any errors and inconsistencies which it contains, and to whether or not it is complete
- have the ability to deal with uncertainty
- have or provide access to graphics facilities, which can be used to illustrate the questions asked of the user and the conclusions presented by the system
- be suitable for users from a range of age-groups, and more importantly for a range of abilities, both specifically in the area of computing and more generally in other areas
- be able to run on the kind of hardware which will be available in schools, which realistically means a small micro.
- be one which can be used throughout the curriculum. This carries with it the implication that it will be used as a tool to promote learning in those areas in which it is used.

We described in Chapter 2 how there are many features which an expert system shell can have, and, that realistically, it is not possible for any one shell to provide everything. In deciding which features to provide there are many factors which must be considered. For the EXPLORES shell, the above criteria had to be taken into account, as had the constraints resulting from the memory and power limitations of the kind of machine on which it had to be implemented. For these reasons, there were many features which were not included.

There are several different representations which allow more efficient inferencing during a consultation, such as rule sets; when daemons; choice of forward or backward chaining; and agenda control. As the inferencing done by the system was something which we proposed users should not have to consider during knowledge base development, we could not include any features of this type. In addition, there are many features which although desirable, are intractable on the kind of hardware currently used in schools, due to the fact that they are computationally very expensive, requiring a substantial amount of memory and processing power. These include sophisticated user-modelling; access to low-level programming languages; uncertainty handling; truth maintenance; graphics facilities; and access to database or spreadsheet software. Thus, the memory and speed constraints of the available hardware meant that none of these features are available in EXPLORES.

8.2 The EXPLORES Shell

Given the above criteria and practical constraints, the EXPLORES shell was designed and implemented. The shell has two separate but linked environments (one for building and one for consulting a knowledge base); three supplementary tools (a knowledge checking tool, a browsing tool, and an explanation tool); and a knowledge representation language (KRL) with four basic knowledge structures (*rules, questions, assignments* and *things*). A knowledge base can be developed in the building environment, which is built around the KRL. This has facilities

for adding to, editing, viewing, loading, unloading and saving a knowledge base. In the consulting environment a knowledge base can be loaded, consulted, unloaded and viewed. During a consultation, the user is asked questions about the current situation, and the system uses the answers given and the inferences which can be made from them (dynamic knowledge), along with the knowledge in the knowledge base (static knowledge), to deduce the result of the consultation.

The shell's three supplementary tools can be used to check a knowledge base for syntax and semantics errors (the knowledge checking tool); to browse through a knowledge base using a number of different methods and paths (the browsing tool); and to explore a knowledge base by asking questions about the domain knowledge contained there (the explanation tool).

The frame-like *things* in the EXPLORES KRL provide a means whereby users can represent objects and classes of objects, in terms of their properties and of the relationships between them. The *rules* and *questions* of the KRL are used to provide dynamic values for the properties of *things* during a consultation: *questions* to ask the user for information, and *rules* to conclude values based on other known information. *Assignments* are used to present text on the screen. The EXPLORES KRL and inferencing strategy are based on that of CRESS [Ross 89].

The shell is almost fully menu driven: this cuts down on problems caused by mistyping; ensures that the user knows the options available at every point; and improves the speed at which users can interact with the shell. In addition, the inclusion of a safety, 'do nothing' option as the first item on each menu allows the user to return from that menu with no action taken.

8.3 Conclusions

We would propose the following conclusions about the extent to which the EXPLORES shell fulfills the initial criteria:

- the EXPLORES KRL provides a simple way of representing knowledge domains which are hierarchical in nature, or which can be made to fit a hierarchical framework.

- the EXPLORES KRL is better then a KRL based on production rules: users find it easier to represent knowledge than they would with a production rule formalism. This could in part be due to the fact that it allows explicit representation of domain knowledge (in *things*) separate from problem-solving knowledge (in *rules, questions* and *assignments*). In addition to giving the user a clear picture of how the knowledge in a domain fits together when a knowledge base is being developed, this separation also means that a knowledge base can be used more effectively for tasks other than consultation, such as tutoring and explanation generation.

- the EXPLORES building environment can provide support which is specific to the shell's KRL, in a way that a text editor cannot. The development of a knowledge base was one of the most difficult tasks identified in the initial evaluation, and the EXPLORES building environment goes some way towards making it easier.

- the editing facilities provided within the building environment are better than those available in many other shells, in that the they can be related to both the KRL and to the specific knowledge in the knowledge base. However, they require some improvement before the shell can be used in the classroom.

- the shell's inferencing strategy produces conclusions during a consultation by using an algorithm which makes use of the hierarchical structure of

a knowledge base. It uses a top down, breadth-first strategy which can consider several branches of the hierarchy in parallel, and can present more than one possible answer. Use by subjects and by the author indicate that this algorithm always produces a 'correct' result, although as is always the case, this is dependent on the correctness and completeness of the knowledge in the knowledge base. As the inferencing strategy is implicit in the structure of the knowledge base, it is something which need not be taken into account by the user while developing a knowledge base: this is a strong point in favour of both the inferencing strategy and of the KRL.

- although it saves typing (and consequently is quick and reduces input errors), the space-bar facility with which information can be entered during a consultation is confusing to novice users. In addition, it provides little by way of a safety net (every key press causes some action to be taken). It should therefore be changed to a more consistent and obvious menu-driven method, which provides the same benefits and is less prone to error.

- the knowledge checker is an invaluable tool for users (particularly novices) who are trying to develop a knowledge base. It provides clear and informative feedback about many of the immediate and potential errors in a knowledge base, and consequently is another feature of the shell which helps users with the difficult task of knowledge base development. However, it should be extended to cover a wider range of errors.

- the range of questions provided by the explanation tool is broad enough to allow the effective exploration of the domain knowledge in a knowledge base, through the generation of domain-based explanations in response to these questions. However, although none were suggested during the evaluation of the tool, we would propose that some additional questions could be added.

- the use of follow-up questions in the explanation tool, which are tied specifically to the content of the original question, to the generated explanation,

and to the knowledge in the knowledge base, provide a means by which users can investigate a topic further. In the absence of user modelling techniques which could tailor the generated explanations to the user's knowledge of the domain or to their current requirements in the consultation, follow-up questions provide a means whereby the user has control over the length, content and direction of a generated explanation.

- the strategies used to generate explanations from the domain knowledge are ones which place no additional burden on the user: they make use of the inherent structure of the EXPLORES KRL, and the user need represent no knowledge over and above that needed for the consultation of a knowledge base.

- we have no formal evidence to support the claim that the domain-based explanations generated in response to original and follow-up questions help pupils to learn about a knowledge domain. However, informal feedback from experienced teachers suggests that they would be good for this purpose (in some cases with slight alterations to the presentation and language used). There are no other expert system shells which provide such a facility.

- the explanation tool has no facilities for requesting and generating system-based explanations, which would be useful for tasks such as debugging. This is not seen as a major failing: the main goal in the development of EXPLORES was that it provide a means whereby users can learn about the domain of a knowledge base, and system-based explanations are less useful for this purpose than domain-based ones.

- expert system shells potentially have three main uses in the classroom. The first is where pupils develop their own knowledge bases; the second is where pupils add knowledge to an existing knowledge base; and the third is having pupils consult pre-built knowledge bases. There are many different learning opportunities in all three of these uses, both specifically with regard to the domain of the knowledge being acquired, organised and represented,

and more generally with regard to the development higher-order thinking skills related to issues such as problem solving. The provision of both a building and a consulting environment in the EXPLORES shell means that it is suitable for all three types of use, and consequently for providing pupils with the benefits of those uses.

- in both evaluations, teachers expressed doubts about the ability of younger or less able pupils to build their own knowledge bases, but were confident that almost all pupils in secondary schools would be able to consult a knowledge base. The separation of the EXPLORES consulting and building environments means that the shell can be used for either purpose with little or no overlap. Thus, the two tasks can be regarded as conceptually different by different categories of user. In addition, the supplementary tools provide a means of support for novice users which need not be used by more experienced users, and as such allow the shell to be tailored to individual requirements to some extent.

- the speed, robustness and interface of the shell must all be improved before it can be used in schools. The interface lacks any of the facilities found in most educational software currently being produced, such as use of a full WIMP ENVIRONMENT, colour, and graphics. In addition, the shell is too slow and too prone to failure in the face of user error to be any good in the classroom. Although these factors are important, and the success of the shell as en educational tool depends on their being improved, it is still possible to be positive about the ideas presented in the shell in spite of them.

We are in no doubt that expert system shells have a place in the classroom as a learning resource. There are many possible roles which they could assume there, and they offer much to both teachers and pupils that cannot be found in other media. We would argue strongly that for their full potential to be realised, they should be used not only in Computer Studies departments, but throughout the school curriculum, to help pupils learn about any number of

subjects. Although not ideal, the EXPLORES shell offers one possibility for the kind of shell which could be used in this way.

In the next section we will discuss some changes which could be made to the EXPLORES shell to improve its chances of success as a classroom tool. However, it is important to restate here that the problem is a wider one. If computers are to be used successfully in schools in this country, then there must be a major increase in the resources available for hardware, software, and training, with well thought out policies for issues such as software development. One of the main aims of the program from which this research project developed was to encourage collaboration between industry, researchers and educators in the development of educational software. Although this is an aim with which we are in strong agreement, its realisation, at least in this project, was flawed because teachers and course organisers were expected to contribute *in addition to* their normal, day-to-day work. At best this is unrealistic given the heavy demands currently made on staff by schools and colleges. For this type of collaboration to be successful, there must be available the resources for all parties to work full-time on the project; for the provision of hardware in schools; for incremental testing and development of the software in schools; for a large-scale evaluation of the software; and for teacher training in the use of the software. It is only then that we will begin to see the successful design, implementation and use of effective software for schools based on sound educational principles and the practicalities of classroom use.

8.4 Further Work

In this section we will discuss further work which could be carried out with regard to the research described in this thesis. This will cover three main areas: improvements and extensions to the EXPLORES shell; the development of further materials to support the use of the shell in the classroom; and further research in the area of explanation generation in expert system shells.

8.4.1 The EXPLORES shell

There are several improvements which could be made to the EXPLORES shell, both in light of the evaluation described in the previous chapter and as a result of extensive use of the shell by the author for both building and consulting knowledge bases.

The editing facilities of the shell are a feature which a few of the subjects said should be improved. Currently, they are very basic: for example, the premise of a rule can only be edited by replacement of the entire premise. This is slightly better then replacement of the entire rule, which is necessary in some in-built editors (for example Xi), but can still be time-consuming. One way in which this could be improved would be to provide a hook to an underlying text editor such as micro-Emacs, which would allow character by character editing, but there are two main reasons why this is not desirable. The first is that in many ways the EXPLORES editor is structured around the KRL, meaning that, for example, prompts can be given for the part of the structure being edited, or that all of the possible relationships between *things* can be presented. This tailoring would be lost if a standard text editor were to be substituted. The second reason concerns the design of the EXPLORES building environment, one of the main strengths of the shell. Having such a highly structured, dedicated environment means that the user need not be concerned to any great extent with the syntax of the KRL, and that the possibility of a knowledge base containing syntax errors is greatly reduced. Again, this feature would be lost if a standard text editor were to be used.

One solution could be to reduce the granularity of the current editor. For example, if a rule exists of the form

if sky = blue
and temp > 65
and ...
then suggested activity = picnic

then currently any alterations which need to be made to any part of the rule's premise involve the retyping of every part of it. A substantial improvement to

this would be to have a series of menus where the user can first of all select that the rule's premise is to be changed; then which clause of the premise is to be changed; and finally whether or not the whole clause or just a part of it is to be changed (such as the operator, the variable or the value). At that point, the alteration to be made could then by typed in. Although this is still not character-by-character editing, it is á solution which would preserve the benefits of the current editor (such as the links to the KRL and the knowledge in the knowledge base being edited), and would remove some of its disadvantages (such as the need for a large amount of typing if a large knowledge structure is to be edited). An additional improvement would be to make the editor accessible to the user when knowledge is being added without the need to go through the main building menu. However, this might be less necessary if the transition time between the two tasks could be improved (see below).

In suggesting improvements to the editor we have so far assumed that the shell's interface remains menu-driven. However, there are many additional improvements which could be made if the interface was altered such that the use of a mouse became possible. For example, this would allow the user to select the exact part of the knowledge structure to be edited, which could be anything from a single character to several lines of text (such as the entire premise of a rule). This would provide a much more flexible and faster way of editing than any of the methods possible using menus only.

In order to satisfy those users who would prefer to develop a knowledge base using a text editor outwith the shell, it is necessary to develop a a more formal syntax for the EXPLORES KRL, such as that outlined for the CRESS KRL in Appendix A. The syntax for the EXPLORES rules, questions and assignments is the same as that described for CRESS, but there is as yet no way in which *things* can be represented in this way. This would be a relatively simple matter, involving the declaration of reserved words and symbols as Prolog operators, and the form of the syntax would be similar to that used by the shell when displaying *things* on the screen (see Appendix B).

Although it is relatively successful as it is, with regard to the provision of

building support, the knowledge checking tool could be expanded to cover more errors. An example of such an extension would be checking that there are no errors in the representation of inherited properties between the *things* in a knowledge base. Properties can be inherited by *things* from other *things* which are above them on the same branch of the hierarchy. This can be full inheritance, where the property and all of its stated values are inherited, or partial inheritance, where the property and a subsection of its stated values are inherited. It is not possible for an inherited property to have any values other than those already stated. Thus, the knowledge checking tool could be expanded such that it picks up and reports on this and any other errors which it currently misses.

During a consultation, the user is asked to provide information about the current situation, on which the system's conclusions can be based. For reasons of speed and accuracy, it is desirable that the user should not have to type in this response, unless an answer is to be entered which is not one of the suggested answers. The current method for entering one of the suggested answers involves using the space-bar to move through those answers and the carriage return key to select one of them. This should be changed such that the suggested answers are presented as a menu, with additional options for entering the explanation tool and choosing to type an answer instead. Almost all of the menus in the EXPLORES shell have a safety option as the first item on the menu, which allows the user to return to the previous point with no action taken. During a consultation, selection of this option would effectively mean that information needed by the system would not be entered, and for this reason it might be useful to include a further safety-net asking 'Are you sure you don't want to answer this question?'.

This is linked to further alterations which could be made to the shell: currently, there is no provision for dealing with uncertain or missing data. A simple method for doing this would be to allow the user to indicate that the information requested by the system is not known. Some slight alterations to the shell's inferencing strategy would then allow the presentation of best-fit answers, saying that the values concluded for the properties for which information is available

suggest that a stated *thing* might be the answer, but that there is not information available to conclude values for all of the relevant properties. Another solution would be the use of certainty factors in a way similar to that used by CRESS but extended to incorporate *things*. These would allow the user to provide a numerical measure of the certainty of the entered response, and could be combined during a consultation to provide an overall certainty for the suggested conclusions.

Although the range of questions available in the EXPLORES explanation tool was judged to be broad enough to allow effective exploration of a knowledge base, there are still some questions which could be added. For example, the question 'Does X have P?' also has a more specific form, which is 'Does X have P with value V?'. A possible extension to this would be to allow the user to add more than one value ('Does X have P with value V and value W but not value Z ...'); or more than one property ('Does X have P with value V and R with value W ...'). There are several extensions such as this which could be made to the range of questions available, but as we discussed previously, these are directly linked to and therefore limited by the syntax of the EXPLORES KRL.

The EXPLORES explanation tool has no facility for generating system-based explanations. However, the inclusion of such a facility would be a simple addition to make using the appropriate code from CRESS for keeping a trace of the consultation, and incorporating any changes necessitated by the additional knowledge structure in the EXPLORES KRL. This would allow the user to ask WHY in response to being asked a question during a consultation (meaning 'Why is that information needed'), and to ask HOW when presented with the result of the consultation (meaning 'How was that conclusion reached?'). The appropriate rule-trace could then be presented in response.

The overall slowness of the shell is a problem which could be solved by measures such as re-writing parts of the Prolog code so that it uses more repeat-fail loops and less recursion (recursion is computationally very expensive in Arity Prolog, a fact which was not discovered until some way into the EXPLORES implementation). In addition, the structure of the shell could be changed: currently,

a small part of it is compiled and the remaining files are loaded at run-time as required. This is in keeping with the 'toolkit' aspect of EXPLORES described previously. However, the shell could perhaps be made to run faster by compiling all of it. Although this would mean that the toolkit aspect would be lost internally, there would be no change to the shell's external presentation to the user. In addition, the shell must be made more robust, by taking actions such as disabling user input where appropriate, or flushing input buffers to get rid of any characters entered in error.

We described in Chapter 6 how the original design for EXPLORES had five supplementary tools, but that in the event only three of these were implemented. The remaining two tools, an answer-changing tool and an advice-giving tool, were both features which emerged from the initial evaluation as desirable ones to have, although they were not as high a priority as the other tools. We would argue that these tools would still enhance the profile of the shell in the classroom, and that any further work on the shell should include their development. Of the two, the advice-giving tool is of a higher priority.

8.4.2 The use of EXPLORES in schools

The major barrier to the use of the EXPLORES shell in schools in its current form is the lack of hardware upon which to run it. At the beginning of this project, it appeared that the RML Nimbus would become one of the main microcomputers used in schools. However, until now most schools in Scotland have used either Amstrad or BBC micros, and many are now moving in the direction of the Apple Macintosh. Very few schools use the Nimbus. In England and Wales, the picture is slightly different: there, the Nimbus is more popular. Assuming that this situation is unlikely to change in the near future, then the lack of suitable hardware is a problem for the widespread use of the EXPLORES shell. To this end, it would seem that the shell should be re-implemented in a language suitable for other hardware currently present in schools, although this would be a major undertaking if the language were to be anything other than another dialect of

Prolog (such as LPA Prolog on the Macintosh). Even then, it would involve a substantial amount of re-programming if, for example, the WIMP features of the Apple Macintosh were to be included.

For those schools which do have appropriate hardware, there is still some more work which would have to be done on the documentation for the EXPLORES shell. For the shell to be evaluated, a User Guide was produced, which was aimed at teachers with some prior knowledge of expert system shells. If EXPLORES is ever to be used in classrooms, it is essential that a set of teaching and support materials be developed for both teachers and pupils. These would include resources such as appropriate knowledge bases, worksheets, and tutorials, and ideally would be based on material currently being taught in schools, and developed in conjunction with the teachers who would be using them.

A problem mentioned several times elsewhere in this thesis is the lack of time for teachers to familiarise themselves with new software, and the lack of computer-training for teachers in general. The successful introduction of the EXPLORES shell into schools would be greatly enhanced by giving teachers an intensive training course using the shell, away from the classroom, to the point where they felt they could use it confidently, and, more importantly, could teach pupils how to use it. Only then would it be possible for teachers outwith computer studies departments to realise the potential of the shell as a tool within their classroom.

8.4.3 The generation of domain-based explanations

In previous chapters we have discussed the point that by and large the explanation facilities of expert system shells are basic: they can generate system-based explanations in response to 'how' and 'why' questions, and their domain-based explanations are limited to canned text. For this reason, one of the major contributions of this thesis is the development of a domain-independent technique for generating domain-based explanations in an expert system shell. We have already discussed the utility of these explanations as an educational medium: in

this section we will describe two extensions to the generation technique which would enhance this. In practical terms, it would not currently be possible to carry out these extensions and to still be able to run the shell on the kind of machine found in schools. However, the extensions are interesting and worthwhile ones in spite of this, and hopefully the development of more powerful micros means that the problem will not always exist.

In Chapter 5 we described some strategies for the correction of apparent user-misconceptions about the domain of a knowledge base in a question-answering system [McCoy 84]. In the EXPLORES shell, these strategies could be used both to shape the content of some the explanations generated by the EXPLORES explanation tool, and to shape the choice of follow-up questions available after an explanation has been generated. The use of such strategies is only possible for questions which can be answered by confirmation or denial of a posited fact (such as 'Is thing1 and example of thing2?') rather than those requesting information (such as 'What properties does thing1 have?'). McCoy lists two main categories in which misconceptions can occur: superordinate misconceptions (where an object is classified wrongly) and attribute misconceptions (where a property is wrongly attributed to an object). In the EXPLORES explanation tool, the correction strategies for superordinate misconceptions would be appropriate for questions of the form:

Is X a Y?
Is X part of Y?

and those for attribute misconceptions for the questions:

Does X have P?
Does X have P with value V?
Does X do P?

Two other questions would be suitable in the second category if altered slightly: 'How are X and Y similar?' could become 'Are X and Y similar?', and similarly for 'How do X and Y differ?'

The EXPLORES explanation tool has no capacity for user-modelling, although, as described previously, the user can control the length, content and direction of the generated explanations by the judicious use of follow-up questions. A possible enhancement to the shell would therefore be the inclusion of some user-modelling facilities which would structure the generated explanation according to previous questions asked. For example, if the user asks the question:

What is cactus spurge an example of?

then the system currently responds with:

cactus spurge is an example of spiny euphorbia

If at a later point the question:

What is resinifera an example of?

is asked, then the system will respond with a very similar answer:

resinifera is an example of spiny euphorbia

The inclusion of a user modelling facility which, during a single session, would create and update a simple model of the user based on the questions asked and the explanations generated, would enable tailored explanations to be generated, such as:

You already know that cactus spurge is an
example of spiny euphorbia.

resinifera is also an example of spiny euphorbia .

This would then allow comparisons to be made between cactus spurge and resinifera using follow-up questions.

Both of these enhancements to the EXPLORES explanation tool would enable it to actively tutor the user about the domain of a knowledge base, albeit in a relatively simple way, and would help the shell to be used more effectively as a tool to promote learning throughout the school curriculum.

Bibliography

[Adams 84] J. Barclay Adams. Probabilistic Reasoning and Certainty Factors. In Bruce B. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, pages 263–271, Addison-Wesley, Reading, MA, 1984.

[Aikins 83] Janice S. Aikins. Prototypical Knowledge for Expert Systems. *Artificial Intelligence*, 20:163–210, 1983.

[Allwood *et al* 85] R.J. Allwood, D.J. Stewart, C. Hinde, and B. Negus. *Report on expert system shells evaluation for construction industry applications.* Technical Report, University of Technology, Loughborough, 1985.

[anon 85] anon. Directory of microcomputer-based software for expert systems work. *Expert Systems*, 2(4):222–227, October 1985.

[Belzer *et al* 80] Robert Belzer, Lee Erman, Philip London, and Chuck Williams. HEARSAY-III: A Domain-Independent Framework for Expert Systems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 108–110, 1980.

[Bennett & Englemore 84] James S. Bennett and Robert S. Englemore. Experience using EMYCIN. In Bruce B. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, pages 314–328, Addison-Wesley, Reading, MA, 1984.

[Bloomfield 86] Brian P. Bloomfield. Capturing Expertise by Rule Induction. *The Knowledge Engineering Review*, 1(4):30–36, 1986.

[Bobrow & Winograd 77] Daniel G. Bobrow and Terry Winograd. An Overview of KRL, a Knowledge Representation Language. *Cognitive Science*, 1:3–46, 1977.

[Bonar & Cunningham 88] Jeffrey Bonar and Robert Cunningham. Bridge: an intelligent tutor for thinking about programming. In John Self, editor, *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, chapter 24, pages 391–409, Chapman and Hall, London, 1988.

[Brachman & Levesque 85] Ronald J. Brachman and Hector J. Levesque, editors. *Readings in Knowledge Representation.* Morgan Kauffman, 1985.

[Brachman 83] Ronald J. Brachman. *What ISA Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks.* Technical Report 638, Fairchild Laboratory for Artificial Intelligence Research, May 1983.

[Brachman 85] Ronald J. Brachman. "I Lied about the Trees" Or, Defaults and Definitions in Knowledge Representation. *The AI Magazine*, 80–93, Fall 1985.

[Briggs 87] Jonathon H. Briggs. *Applications of Expert Systems in Further Education: A Report to the Further Education Unit.* Pre-publication version 1.0, 1987.

[Brown 87] A. Brown. Metacognition, Executive Control, Self-Regulation and Other More Mysterious Mechanisms. In F.E. Weinert and R.H. Kluwe, editors, *Metacognition, Motivation and Understanding*, pages 65–115, Lawrence Erlbaum, Hillsdale, NJ, 1987.

[Buchanan & Shortliffe 84] Bruce G. Buchanan and Edward H. Shortliffe, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, MA, 1984.

[Christian-Carter 87] J. Christian-Carter. Q-VITAMINS: Using PROLOG in Nutrition Education. In *Proceedings of the Knowledge Based Systems in Education Conference (PEG '87)*, July 1987.

[Clancey 79] W. J. Clancey. Tutoring rules for guiding a case method dialogue. *IJMMS*, 11:25–49, 1979.

[Clancey 83] William J. Clancey. The Epistemology of a Rule-Based Expert System - a Framework for Explanation. *Artificial Intelligence*, 20:215–251, 1983.

[Cohen *et al* 85] Paul Cohen, Alvah Davis, David Day, Michael Greenberg, Rick Kjeldsen, Susan Lander, and Cynthia Loiselle. Representativeness and Uncertainty in Classification Systems. *The AI Magazine*, 136–149, Fall 1985.

[Conlon & Cope 89] Tom Conlon and Peter Cope, editors. *Computing in Scottish Education: The first decade and beyond. Edinburgh Education and Society*, Edinburgh University Press, 1989.

[CRI 86] CRI. *The CRI Directory of Expert Systems.* Learned Information Ltd, Oxford and New Jersey, 1986.

[Curran 86] Susan Curran. CRYSTAL: Not Brainy But Beautiful. *Practical Computing*, 76–77, November 1986.

[Davis 80] M. Davis. The mathematics of non-monotonic reasoning. *Artificial Intelligence*, 13:73–80, 1980.

[diSessa 82] A. A. diSessa. *A Principled Design for an Integrated Computing Environment.* Technical Report, Laboratory for Computer Science, Massachusetts Institute of Technology, 1982.

[Duda 80] R.O. Duda. *The PROSPECTOR System for Mineral Exploration.* Technical Report, SRI International, 1980.

[Eisenstadt & Brayshaw 88] M. Eisenstadt and M. Brayshaw. The Transparent Prolog Machine (TPM): an execution model and graphical debugger for logic programming. *Journal of Logic Programming*, 5(4):277–342, 1988.

[Epp *et al* 88] Helmut Epp, Martin Kalin, and David Miller. PC Software for Artificial Intelligence. *Science*, 240:824–830, 6th May 1988.

[ESP85] *ESP ADVISOR User Guide and Reference Manual.* Expert Systems International Ltd, 9 West Way, Oxford OX2 0JB, issue 2 edition, July 1985.

[ESP86] *ESP Frame-Engine User Guide.* Expert Systems International Ltd, 9 West Way, Oxford OX2 0JB, October 1986.

[Exp85] *Expertech Xi User Guide and Reference Manual.* Expertech Ltd, 1985.

[Exp89] *Expert Builder: Learning through Structuring and Exploring Ideas.* Integrated Modelling Project, October 1989.

[Fikes & Kehler 85] Richard Fikes and Tom Kehler. The Role of Frame-based Representation in Reasoning. *Communications of the ACM*, 28(9):904–920, September 1985.

[Forgy 82] C.L. Forgy. Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[Forsyth 86] Richard Forsyth. Software review: Personal Consultant Plus. *Expert Systems*, 3(4):244–246, October 1986.

[Forsyth 87] Richard Forsyth. Software review: Expertech Xi Plus. *Expert Systems*, 4(1):48–51, February 1987.

[Gilbert 87] G. Nigel Gilbert. Question and answer types. In D. S. Moralee, editor, *Research and Development in Expert Systems VI*, pages 162–172, Proceedings of Expert Systems '87, Cambridge University Press, Cambridge, 1987.

[Goldberg & Robson 83] A. Goldberg and D. Robson. *Smalltalk-80: The language and its implementation.* Addison-Wesley, Reading, MA, 1983.

[Goldberg 81] A. Goldberg. Introducing the Smalltalk-80 system. *BYTE*, 6(8), 1981.

[Gordon & Shortliffe 84] Jean Gordon and Edward H. Shortliffe. The Dempster-Shafer Theory of Evidence. In Bruce B. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, pages 263–271, Addison-Wesley, Reading, MA, 1984.

[Hammond & Sergot 85] *apes: Augmented PROLOG for Expert Systems.* Logic Based Systems Ltd, Richmond, Surrey, 2nd edition, 1985.

[Harmon *et al* 88] Paul Harmon, Rex Maus, and William Morrisey. *Expert Systems: Tools & Applications.* John Wiley & Sons, Inc, 1988.

[Harrington] R.J. Harrington. KES - An expert system development tool. Circulated report.

[Hasling *et al* 84] Diane Warner Hasling, William J. Clancey, and Glenn Rennels. Strategic explanations for a diagnostic consultation system. *International Journal of Man Machine Studies*, 20:3–19, 1984.

[Hassell 87] D. Hassell. Using ADEX-Advisor as a Tool for Qualitative Modelling: Some experiences from secondary-school trials. In *Proceedings of the Knowledge Based Systems in Education Conference (PEG '87)*, July 1987.

[Hughes 86] Sheila Hughes. Question classification in rule-based systems. In M. A. Bramer, editor, *Research and Development in Expert Systems III*, pages 123–131, Proceedings of Expert Systems '86, Cambridge University Press, Cambridge, 1986.

[Jackson 90] Peter Jackson. *Introduction to Expert Systems.* Addison-Wesley, Reading, MA, 2nd edition, 1990.

[Johnson 88] W. Lewis Johnson. Modelling programmer's intentions. In John Self, editor, *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, chapter 23, pages 374–390, Chapman and Hall, London, 1988.

[Kass & Finin 88] Robert Kass and Tim Finin. The Need for User Models in Generating Expert System Explanations. *International Journal of Expert Systems*, 1(4):345–375, 1988.

[Kidd 85] Alison L Kidd. What do users ask? - some thoughts on diagnostic advice. In M. Merry, editor, *Expert Systems '85*, pages 9–19, Cambridge University Press, 1985.

[Laurent *et al* 86] Jean-Pierre Laurent, Jacqueline Ayel, Franck Thome, and Danielle Zieblin. Comparative Evaluation of Three Expert System Development Tools: KEE, Knowledge Craft, ART. *Knowledge Engineering Review*, 4(1):18–29, 1986.

[Lehner & Barth 85] Paul E. Lehner and Stephen W. Barth. Expert systems on microcomputers. *Expert Systems*, 2(4):198–208, October 1985.

[Lehnert 78] Wendy G. Lehnert. *The Process of Question Answering.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1978.

[Leo89] *Leonardo: An Evaluation Guide.* Creative Logic Limited, Uxbridge, Middlesex, 1989.

[Lewis 86] R. Lewis. *Research in Progress.* ESRC-ITE programme, May 1986.

[Mandl & Lesgold 88] H. Mandl and A. Lesgold, editors. *Learning Issues for Intelligent Tutoring Systems.* Springer-Verlag, New York, 1988.

[McCoy 84] Kathleen McCoy. Correcting Object-Related Misconceptions: How Should The System Respond? In *Proceedings of COLING '84*, pages 444–447, 1984.

[McDermott & Doyle 80] D. McDermott and J. Doyle. Non monotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.

[McKeown 88] Kathleen R. McKeown. Generating Goal-Oriented Explanations. *International Journal of Expert Systems*, 1(4):377–395, 1988.

[Minsky 75] Marvin Minsky. A framework for representing knowledge. In Patrick H. Winston, editor, *The psychology of computer vision*, McGraw-Hill, 1975.

[Neches *et al* 85] R. Neches, William R. Swartout, and J. Moore. Explainable (and maintainable) expert systems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 382–389, 1985.

[PAL86] *I-1 User Guide and Reference Manual.* PAL Software Ltd, PO Box 24, Ashley Rd, Hale, Altringham, Cheshire, WA15 9QT, 1986.

[Quinlan 79] J. R. Quinlan. Discovering Rules By Induction From Large Collections of Examples. In Donald Michie, editor, *Expert Systems in the Microelectronic Age*, pages 168–201, Edinburgh University Press, 1979.

[REF87] *REFINE User Guide and Reference Manual.* Office Workstations Ltd, 144 Broughton Rd, Edinburgh, EH7, 1987.

[Reiser *et al* 85] B.J. Reiser, J.R. Anderson, and R.G. Farrell. Dynamic student modelling in an intelligent tutor for lisp programming. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 8–14, 1985.

[Reiter 80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 81–132, 1980.

[Rich 83] Elaine Rich. *Artificial Intelligence.* McGraw-Hill, 1983.

[Robertson *et al* 85] David Robertson, Robert Muetzelfeldt, Dave Plummer, Mike Uschold, and Alan Bundy. The ECO Browser. In M. Merry, editor, *Expert Systems '85*, pages 143–155, Cambridge University Press, 1985.

[Ross 89] Peter M. Ross. *Advanced Prolog - Techniques and Examples.* Addison-Wesley, 1989.

[Schefe 80] Peter Schefe. On foundations of reasoning with uncertain facts and vague concepts. *International Journal of Man-Machine Studies*, 12:35–62, 1980.

[Scherz *et al* 88a] Z. Scherz, B. Weinberg, M. Pontch, and D. Goldberg. EESS - Educational Expert System Shell. In *Proceedings of PICKET Education Group Conference (PEG '88)*, July 1988.

[Scherz *et al* 88b] Z. Scherz, B. Weinberg, M. Pontch, and D. Goldberg. The PROLOG in Education Project. *PEG-BOARD*, 3(1), summer 1988.

[Scott *et al* 84] A Carlisle Scott, William J. Clancey, Randall Davis, and Edward H. Shortliffe. Methods for Generating Explanations. In Bruce G. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, chapter 18, pages 338–362, Addison-Wesley, Reading, MA, 1984.

[Shortliffe & Buchanan 84] Edward H. Shortliffe and Bruce B. Buchanan. A Model of Inexact Reasoning in Medicine. In Bruce B. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, pages 233–262, Addison-Wesley, Reading, MA, 1984.

[Sleeman 85] D. Sleeman. UMFE: A User Modelling Front-End subsystem. *International Journal of Man Machine Studies*, 23:71–88, 1985.

[Steels 84] L. Steels. Second-Generation Expert Systems. *Future Generation Computer Systems*, 4(1):213–221, 1984.

[Steels 90] Luc Steels. Components of Expertise. *AI Magazine*, 11(2):28–49, 1990.

[Stefik 78] Mark Stefik. Inferring DNA structures from segmentation data. *Artificial Intelligence*, 11:85–114, 1978.

[Stefik *et al* 82] Mark Stefik, Jan Aikins, Robert Balzer, John Benoit, Lawrence Birnbaum, Frederick Hayes, and Earl Sacerdoti. The Organization of Expert Systems, A Tutorial. *Artificial Intelligence*, 18:135–173, 1982.

[Storrie 87] P. Storrie. Computers in history - practice into theory. In *Proceedings of the Knowledge Based Systems in Education Conference (PEG '87)*, July 1987.

[Swartout 81] William R. Swartout. *Producing Explanations and Justifications of Expert Consulting Programs*. Technical Report MIT/LCS/TR-251, Massachusetts Institute of Technology, January 1981.

[Tou *et al* 83] Frederick N. Tou, Michael D. Williams, Richard Fikes, Austin Henderson, and Thomas Malone. RABBIT: An Intelligent Database Assistant. In *Proceedings of the American Association for Artificial Intelligence*, pages 314–318, 1983.

[vanMelle 79] William van Melle. A Domain-Independent Production-Rule System for Consultation Programs. In *Proceedings of the 6th International Joint Conference in Artificial Intelligence*, pages 923–925, 1979.

[vanMelle *et al* 84] William van Melle, Edward H. Shortliffe, and Bruce G. Buchanan. EMYCIN: A Knowledge Engineer's Tool for Constructing Rule-Based Expert Systems. In Bruce B. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems: The*

MYCIN Experiments of the Stanford Heuristic Programming Project, pages 302–313, Addison-Wesley, Reading, MA, 1984.

[Wallis & Shortliffe 84] Jerold W. Wallis and Edward H. Shortliffe. Customized Explanation Using Causal Knowledge. In Bruce G. Buchanan and Edward H. Shortliffe, editors, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, chapter 20, pages 371–388, Addison-Wesley, Reading, MA, 1984.

[Webb 87] M. Webb. Towards an Expert System Shell as a tool for conceptual and qualitative modelling in secondary education. In *Proceedings of the Knowledge Based Systems in Education Conference (PEG '87)*, July 1987.

[Webb 88] M. Webb. Knowledge engineering in secondary education - What tools are needed. In *Proceedings of the PICKET Education Group Conference (PEG '88)*, July 1988.

[Weiner 80] J. L. Weiner. BLAH, A System Which Explains its Reasoning. *Artificial Intelligence*, 15:19—48, 1980.

[Weinert & Kluwe 87] F.E. Weinert and R.H. Kluwe, editors. *Metacognition, Motivation and Understanding.* Lawrence Erlbaum, Hillsdale, NJ, 1987.

[Weiss & Kulikowski 79] Sholom M. Weiss and Casimir A. Kulikowski. EXPERT: A system for developing consultation models. In *Proceedings of the 6th International Joint Conference in Artificial Intelligence*, pages 942–947, 1979.

[Whalen & Schott 83] Thomas Whalen and Brian Schott. Issues in Fuzzy Production Systems. *International Journal of Man-Machine Studies*, 19:57–71, 1983.

[Winston 84] Patrick H. Winston. *Artificial Intelligence.* Addison-Wesley: Reading, MA, 2nd edition, 1984.

[Zadeh 76] L.A. Zadeh. A fuzzy-algorithmic approach to the definition of complex or imprecise concepts. *International Journal of Man Machine Studies*, 8:249–293, 1976.

Appendix A

The CRESS KRL

The knowledge representation language and inferencing strategy used in the EXPLORES shell were based on that of CRESS, a rule-based shell written in Edinburgh Prolog. This appendix outlines the syntax for the CRESS KRL, adapted from [Ross 89]. Not all of this KRL is used in EXPLORES (for example, there is no provision for using certainty factors). A CRESS knowledge base must be developed outside the shell using a text editor, and therefore its KRL has a definite syntax which must be adhered to. The inclusion of a building environment in EXPLORES made it possible to remove some of these restrictions (for example, parameter names are more flexible, and there is no need to end each knowledge structure with a full stop).

CRESS has three basic knowledge structures: *rules, assignments* and *questions*. Figures A–1 and A–2 show the syntax of each of these: in both figures, the name in bold type on the left hand side refers to the part being defined; italicised words in the definitions refer to parts which are themselves defined; words in normal type indicate reserved words and operators in the CRESS KRL; and parenthesised text outlines in English the syntax of the named part. The following points should be noted about the CRESS syntax:

- **actions** are carried out in the order in which they appear in a conclusion.
- **cf** stands for certainty factor, or confidence factor.
- a parameter can be declared to be single-valued by including an entry of the form

  parameter is single_valued.

  somewhere in the knowledge base. When the system tries to find values for the named parameter, it will stop as soon as a rule that concludes a value for it succeeds.

Figure A–1: The syntax of the CRESS KRL (I)

assignment	→	*parameter* = *text* .
question	→	*text* finds *parameter*.
rule	→	*rulename* : if *premise* then *conclusion*.
rulename	→	*parameter*
premise	→	*premise_bit*
	→	*premise_bit* and *premise*
	→	*premise_bit* or *premise*
premise_bit	→	*relation*
	→	*status*
relation	→	*parameter* = *value*
	→	*parameter* > *value*
	→	*parameter* < *value*
status	→	*parameter* is known
	→	*parameter* is unknown
	→	*parameter* is investigated
	→	*parameter* is askable
conclusion	→	*conclusion_bit*
	→	*conclusion_bit* and *conclusion*

Figure A–2: The syntax of the CRESS KRL (II)

conclusion_bit	→	*relation*
	→	*relation* cf *cf_number*
	→	*action*
action	→	clear_screen
	→	present *parameter*
	→	await_return
value	→	*parameter*
	→	*variable*
parameter	→	(a word, beginning with a lower case letter and consisting of any of the characters a-z, A-Z, 0-9, underscore)
	→	(a positive number)
	→	(any sequence of these strung together by the connective 'of')
variable	→	(a single word beginning with an upper-case letter and made from a-z, A-Z, 0-9, underscore)
text	→	(one or more characters enclosed in single quotes)
cf_number	→	(a number from 0 to 100 inclusive)

A.1 Examples from a CRESS knowledge base

The following are some examples of knowledge represented in the CRESS KRL:

text_23 = 'We are going to copy the colour of the walls or windows'.

'What price is the paint' finds price of paint.

rule of st_nugent:

if	colour of door of house is unknown
and	colour of walls of house = X
or	colour of windows of house = X
and	price of paint < 5.99
then	colour_to_buy = X cf 65
and	colouration of house = uniform
and	clear_screen
and	present text_23
and	await_return.

The reader should note the use of the variable X in the example rule. It stands for the same value throughout the rule, and has no connection with any other X in the knowledge base.

Appendix B

An EXPLORES Knowledge Base

In this appendix we list an example knowledge base for the EXPLORES shell. The domain of this knowledge base is plants, specifically varieties of cactus and euphorbia. Knowledge is added to an EXPLORES knowledge base in the shell's building environment, where prompts are given for each part of the knowledge structure being added. That knowledge is stored in the shell as a series of Prolog clauses. Thus, the EXPLORES KRL does not have an official syntax. The knowledge presented below is a translation of the stored Prolog clauses, and is how the knowledge in an EXPLORES knowledge base will appear if it is looked at using the 'viewing' option in either the building or the consulting environment. These examples show one way in which this domain knowledge could be be organised and represented, but there are many others.

B.1 Things

NAME: *plants*
Has: *stem protection = thorns, spines or hair*
Relations: has member ⇒ *thorny plants*
has member ⇒ *spiny plants*
has member ⇒ *hairy plants*

NAME: *spiny plants*
Has: *stem protection = spines*
Relations: has member ⇒ *spiny cactus*
has member ⇒ *spiny euphorbia*

NAME: *thorny plants*
Has: *stem protection = thorns*
Relations: has member ⇒ *thorny euphorbia*

NAME: *hairy plants*
Has: *stem protection = hair*
Relations: has member ⇒ *hairy cactus*

NAME: *spiny cactus*
Has: *milky sap in stem = no*
Relations: has member ⇒ *column cactus*
has member ⇒ *goats horn cactus*
has member ⇒ *rats tail cactus*
has member ⇒ *peanut cactus*

NAME: *spiny euphorbia*
Has: *milky sap in stem = yes*
Relations: has member ⇒ *cactus spurge*
has member ⇒ *resinifera*

NAME: *hairy cactus*
Has: *milky sap in stem = no*
Relations: has member ⇒ *old man cactus*

NAME: *thorny euphorbia*
Has: *milky sap in stem = yes*
Relations: has member ⇒ *crown of thorns*

NAME: *column cactus*
Has: *stem = one column*
spine type = brown
ribs = yes

NAME: *goats horn cactus*
Has: *stem = fat and globular*
spine type = curved
ribs = yes

NAME: *rats tail cactus*
Has: *stem = several finger like stems*
spine type = brown
ribs = no
pink flowers = yes

NAME: *peanut cactus*
Has: *stem = several finger like stems*
spine type = white
red flowers = yes

NAME: *cactus spurge*
Has: *stem = tall and branched*
spine type = long spines on branches

NAME: *resinifera*
Has: *stem = fat and ribbed*
spine type = short spines on ribs

NAME: *old man cactus*
Has: *stem = one column*
hair type = long and silvery

NAME: *crown of thorns*
Has: *stem = grooved branches*
thorn type = sharp thorns on stems

B.2 Questions

Question: *What kind of stem does your plant have?*
finds a value for
Item: *stem*

Question: *What kind of stem protection does your plant have?*
finds a value for
Item: *stem protection*

Question: *Does your plant have milky sap in its stem?*
finds a value for
Item: *milky sap in stem*

Question: *Is the stem of your plant ribbed?*
finds a value for
Item: *ribs*

Question: *What type of hair does your plant have?*
finds a value for
Item: *hair type*

Question: *What type of thorns does your plant have?*
finds a value for
Item: *thorn type*

Question: *What colour of spines does your plant have?*
finds a value for
Item: *spine colour*

Question: *What shape of spines does your plant have?*
finds a value for
Item: *spine shape*

Question: *What length of spines does your plant have?*
finds a value for
Item: *spine length*

Question: *Does your plant have flowers?*
finds a value for
Item: *flowers present*

Question: *What colour of flowers does your plant have?*
finds a value for
Item: *flower colour*

B.3 Rules

rule1:
IF *flowers present = yes*
AND *flower colour = red*
THEN *red flowers = yes*

rule2:
IF *flowers present = yes*
AND *flower colour = pink*
THEN *pink flowers = yes*

rule3:
IF *stem = fat and ribbed*
AND *spine length = short*
THEN *spine type = short spines on ribs*

rule4:
IF *stem = tall and branched*
AND *spine length = long*
THEN *spine type = long spines on branches*

rule5:
IF *spine colour = white*
THEN *spine type = white*

rule6:
IF *spine colour = brown*
THEN *spine type = brown*

rule7:
IF *spine shape = curved*
THEN *spine type = curved*

Appendix C

A Questionnaire for Evaluating Expert System Shells

This appendix contains a copy of the questionnaire used during the initial evaluation described in Chapter 3. Teachers participating in this evaluation were given a questionnaire to complete for each expert system shell which they had used. In this way we were able to get feedback on all five of the available shells.

Expert System Shells - General Questionnaire

The purpose of this questionnaire is to get some 'on-the-spot' reactions to the expert system shell which you have just been using. Most of the questions can be answered by circling the appropriate number and adding any comments which you feel are pertinent. Don't spend too much long on it - Charles will be grilling you in more detail at the end of the course. Comments of any nature are more than welcome, so feel free to be critical /enthusiastic (or whatever!).

1. Basic details:

   (a) Shell name:

   (b) Hardware:

   (c) What previous experience do you have of using shells?

1	2	3	4	5	6	7	8	9	10
complete novice									very experienced

   Comments:

2. Using the shell for consultation:

(a) Knowledge base(s) used:

(b) How easy was it to start up the consultation?

1	2	3	4	5	6	7	8	9	10
very difficult									very easy

Comments:

(c) How helpful were the prompts?

1	2	3	4	5	6	7	8	9	10
very unhelpful									very helpful

Comments:

(d) How easy was it to get on-line help on how to use the system?

1	2	3	4	5	6	7	8	9	10
very difficult									very easy

Comments:

(e) How useful was the on-line help?

1	2	3	4	5	6	7	8	9	10
not useful									very useful

Comments:

(f) How easy was it to move around the system?

1	2	3	4	5	6	7	8	9	10
very difficult									very easy

Comments:

(g) How easy was it to get stuck?

1	2	3	4	5	6	7	8	9	10
very difficult									very easy

Comments:

(h) When the system explained its conclusions (in response to 'how' and 'why' commands), were the explanations clear?

1	2	3	4	5	6	7	8	9	10
very unclear									very clear

Comments:

(i) Were they useful?

1	2	3	4	5	6	7	8	9	10
not useful									very useful

Comments:

3. Using the shell to build a knowledge base:

(a) Give a short description of the knowledge base you tried to build (size, domain etc).

(b) How easy was it to understand the features of the proposed knowledge base?

1	2	3	4	5	6	7	8	9	10
very difficult									very easy

Comments:

(c) How easy was it to then build one?

1	2	3	4	5	6	7	8	9	10
very difficult									very easy

Comments:

(d) How easy was it to represent the knowledge?

1	2	3	4	5	6	7	8	9	10
very difficult									very easy

Comments:

(e) How easy was it to alter the knowledge base (e.g. to edit, add to, or remove the stored knowledge)?

1	2	3	4	5	6	7	8	9	10
very difficult									very easy

Comments:

(f) What were your most common errors?

(g) If you didn't get it right first time, how easy was it to debug the knowledge base?

1	2	3	4	5	6	7	8	9	10
very									very
difficult									easy

Comments:

(h) Comments on any constraints, limitations etc that the shell imposed:

4. General comments:

(a) Overall, did you find the system friendly to use?

1	2	3	4	5	6	7	8	9	10
very unfriendly									very friendly

Comments:

(b) Comments on screen-layout, use of colour, etc?

(c) General likes, dislikes, comments, suggestions:

(d) It may be useful for me to ask you for more details on some of your comments. If you have no objection to this, please write your name below - otherwise you can remain anonymous!
Name:

Thanks for your help!

Appendix D

The EXPLORES Evaluation Questionnaire

This appendix contains a copy of the questionnaire used for the evaluation of the EXPLORES shell, described in Chapter 7. Initially, each of the participants was given a copy of EXPLORES and a User Guide, with instructions to use the shell as much as possible (and a prioritised list of all of the features which could be evaluated). After several weeks, each subject was given a questionnaire to complete, which asked questions about each feature of EXPLORES which they could have used; about the shell's potential as a classroom tool; and about the educational use of expert system shells in general.

The EXPLORES shell - evaluation questionnaire

D.1 Background

1. Name:
2. Date:
3. School address:
4. What kind of school do you teach in (mainstream primary / secondary, special needs etc).
5. What age of children do you normally teach?

6. What computer software, if any, do you use with your classes?

7. Have you used any other expert system shells, for teaching or personal use, either in school or elsewhere? If so, which ones have you used, what did you use them for, and what is your opinion of them?

D.2 The EXPLORES Shell

In this section you will find specific questions about EXPLORES, with regard to how you used it and what you liked and disliked about it. As mentioned above, the shell is not really ready for use in schools; what I am interested in are your feelings about the design of the shell and the facilities that it offers.

Some of you may have kept notes as you were using the shell; since I would like copies of these as well as your completed questionnaires, feel free to write 'refer to notes' rather than re-iterating information which you have already given.

D.2.1 The building environment

When this project began, very few small expert system shells provided a facility for building a knowledge base; most required that a knowledge base be built outside the shell, using a standard text editor. Even now, few small shells have that facility.

When designing EXPLORES we decided that it was important that a building environment based on the Knowledge Representation Language (KRL) should be an integral part of the shell, and that this should be separate from the consulting environment, since the two tasks were conceptually different and provided different learning opportunities.

This section contains questions about the basic building environment and the EXPLORES KRL. A later section will ask you about the knowledge checker, one of the supplementary tools which can be used in the building environment.

1. Did you use the EXPLORES building environment? If so, was this to build a new knowledge base or to add to an existing one?

2. What was your opinion of the EXPLORES KRL?

   (a) How easy or difficult was it to represent your domain knowledge in it?

   (b) Did you feel that it was more or less difficult to use than a language based solely on production rules?

   (c) What do you feel are its limits?

   (d) Any other comments about it?

3. Within the building environment there is a simple editor for editing your knowledge base. Did you use this? If so, what was your opinion of it?

4. Within the building environment there is a facility for viewing the knowledge in your knowledge base. Did you use this? If so, what was your opinion of it?

5. Do you feel that the building environment helped you when building a knowledge base, or was it no better than building one using a text editor? Can you say why, and what it was about it that you felt was helpful/unhelpful?

D.2.2 The consulting environment

The EXPLORES consulting environment allows a user to consult a pre-built knowledge base. The knowledge within a knowledge base is analysed by the shell so that, among other things, it can suggest possible answers whenever the user has to be asked for information. It also has a feature which uses the space bar to run through the list of possible answers and the carriage return key to select one of them; this enables the user to enter an answer without typing it, thus ensuring that typing errors cannot occur. Users are also free to enter their own answers.

This section contains questions about the basic consulting environment. A later section will ask you about the explanation tool, one of the supplementary tools which can be used in the consulting environment.

1. Did you use the consulting environment to consult a knowledge base?

2. Did you use the space-bar / carriage return facility to run through and select one of the suggested answers? Did you find this feature useful? What did you like and/or dislike about it?

3. Did you consult a knowledge base that you had built? If so, were the answers you received from the consultation what you expected, and, more importantly, were they correct? If not, was the error yours or the shell's?

4. **Within the consulting environment there is a facility for viewing the knowledge in your knowledge base. Did you use this? If so, what was your opinion of it? (If you have already commented on the viewing facility for the building environment the ignore this question).**

5. **Overall, what did you like and dislike about the consulting environment?**

D.2.3 The explanation tool

The explanation tool is one of the main features of the shell. It is a central argument of my thesis that if shells are to be used in classrooms then they should be used to enable pupils to learn about a subject, rather than about expert system shells. One way of doing this is to provide a facility which allows exploration of the domain knowledge in a knowledge base by asking questions about it; this is the rationale behind the EXPLORES explanation tool.

1. Did you use the explanation tool?

2. Do you think that the questions templates provided allowed you to explore a knowledge base in a useful and informative way? Can you think of any questions that you wanted to ask about the knowledge base, but couldn't?

3. Did you find the follow-up questions useful?

4. How did you feel about the explanations given in response to your questions? Were they clear enough or were they too complex? Were they useful?

5. Do you think that this tool provides a means by which users could explore and learn about the domain of a knowledge base?

6. Overall, what did you like and dislike about the explanation tool?

D.2.4 The knowledge checker

The knowledge checker is built around the EXPLORES KRL, and will check a knowledge base for incorrect, duplicated or missing knowledge. The errors that the knowledge checker reports are only a subset of possible errors that could occur, although there is no reason (other than lack of time) why most, if not all errors could be picked up.

1. Did you use the knowledge checker? If so, did you find it helpful?

2. Did you understand the *type* of errors reported and the *way* in which they were reported?

3. Overall, what did you like and dislike about the knowledge checker?

D.2.5 General questions

The interface of the shell is very much restricted by the limited facilities of the Nimbus and Arity Prolog; the windows and menus that are used are the result of several hours C hacking by my supervisor, Peter Ross.

1. What did you like and dislike about the interface?

2. How could it have been improved?

3. How did you feel about the structure of the shell, that is, an environment for building, a separate environment for consulting, and tools with which these environments could be supplemented (the original design for the shell had several other tools, which had to be abandoned due to lack of time)?

4. Are there other features, either ones that you have seen on other shells or ones that are part of your personal wish-list, that you would have liked the shell to have?

5. Overall, what was your opinion of the EXPLORES shell. What did you like and dislike about it?

D.3 Expert System Shells in Education

Several years ago, it seemed that expert system shells would be a great boon to the classroom. This has not happened, and there seem to be several reasons why, not least of which is that until recently small micro-based shells have been, by and large, rather poor. It is also the case that few, if any, have been designed for educational use. In my thesis, I will argue that if shells are to be used in classrooms, then they should be used as tools with which pupils can learn about a variety of subjects, rather than simply used to teach pupils about expert system shells. Consequently they need to be designed with this criterion in mind.

1. Do you think that there is a place for expert system shells in the classroom?

   If so:

   (a) Ideally, what do you see as their role in the classroom?

   (b) How do you think they should be used (e.g. pupils consulting pre-built knowledge bases, pupils building own knowledge bases etc)?

   (c) Do you think shells have anything to offer which can't be found elsewhere, or is are they just a way of automating something which could be done by another method? What do you think they have to offer, if anything?

(d) In what ways, if any, do you think pupils could benefit from using shells?

If not:

(a) Why not?

(b) Do you feel this is the case for all shells, regardless, or is it that the 'right' shell hasn't been designed yet? If the latter, what would you regard as the right shell?