THE UNIVERSITY *of* EDINBURGH

Beyond Rules: The Development and Evaluation of Knowledge Acquisition Systems for Educational Knowledge-based Modelling

Thomas H. Conlon

Ph.D.

The University of Edinburgh

1997

Abstract

The technology of knowledge-based systems undoubtedly offers potential for educational modelling, yet its practical impact on today's school classrooms is very limited. To an extent this is because the tools presently used in schools are EMYCIN-type expert system shells. The main argument of this thesis is that these shells make knowledge-based modelling unnecessarily difficult and that tools which exploit knowledge acquisition technologies empower learners to build better models. We describe how such tools can be designed. To evaluate their usability a model-building course was conducted in five secondary schools. During the course pupils built hundreds of models in a common range of domains. Some of the models were built with an EMYCIN-type shell whilst others were built with a variety of knowledge acquisition systems. The knowledge acquisition systems emerged as superior in important respects. We offer some explanations for these results and argue that although problems remain, such as in teacher education, design of classroom practice, and assessment of learning outcomes, it is clear that knowledge acquisition systems offer considerable potential to develop improved forms of educational knowledge-based modelling.

Acknowledgements

During the period of this research, I have been fortunate in having two academic 'homes' — the Department of Artificial Intelligence at Edinburgh University and Moray House Institute of Education. Not many people have the privilege of attending an AI workshop in the morning and then spending the afternoon in a school with a group of PGCE students. The opportunity to do this kind of thing on a regular basis has influenced the way I think about computing in education. I'm very grateful to colleagues in both institutions for tolerating and even encouraging my double life.

For advice, support and collaboration I owe many debts: to my AI supervisors, Helen Pain and Dave Robertson; to the teachers who made the research possible, especially Norman Bowman, Ann Cole, Robert Grant, Dorothy Johnstone, Luci Jones, Neil Livesey, John Mason, Andy Pendry, and Doug Urquhart; to the numerous students and children who tested the new Primex tools; to the programmers who developed the systems (especially Apple Macintosh OS v7.5, Microsoft Word v4, LPA MacProlog32 v1.0 and Prolog++ v1.0, and SPSS Inc. SPSS v6.1) without which this work could hardly have been done — thank you all.

Closer to home, Jean Casey always had time to listen to my latest tale of wonder or woe. For days in Crete and Skye, and all the other good times we had during the years of this research, thanks Jean. This thesis is dedicated to you.

Contents

List of Figures

List of Tables

List of Main Abbreviations

AI	Artificial Intelligence
AIED	Artificial Intelligence and Education
IT	Information Technology
ITS	Intelligent Tutoring Systems
KBS	Knowledge-based Systems
PCM	Persistent Collaboration Methodology
PCT	Primex Classification Tree
PDT	Primex Decision Tree
PFT	Primex Factor Table
PKRL	Primex Knowledge Representation Language
PMA	Primex Model Analyser
PMC	Primex Model Comparator

Chapter 1 Introduction

This chapter briefly reviews the background to the research. The research hypothesis and the questions that will be investigated are previewed and the main contributions of the research are summarised. Finally, the structure of the thesis is described by outlining the contents of each chapter.

1.1 Background to the research

Since schooling is much concerned with knowledge, there is at least a *prima facie* case that the technology of knowledge-based systems (KBS) should have something to contribute to the classroom. Indeed, research in Artificial Intelligence and Education (AIED) has already shown that this is so. Intelligent Tutoring Systems (Polson & Richardson 1988), perhaps the best known products of work in AIED, provide one illustration of how KBS can contribute to teaching and learning. Knowledge-based modelling, the subject of this thesis, provides another.

Intelligent Tutoring Systems (ITS) and systems for knowledge-based modelling share certain characteristic features: for example, both use explicit and symbolic representations of knowledge. However, the educational functions of the two kinds of system are very different. An ITS comes with a preconstructed knowledge base representing an authoritative model of the domain that is to be learned. In knowledge-based modelling on the other hand, the learner is seen as the source of knowledge and the computer is a means of expression. The model that matters is the knowledge base created by the learner.

The idea that building a computer knowledge base might be a productive learning activity is still quite new. Happily, but not coincidentally, in the late twentieth century a new type of professional has emerged who provides inspirational support for the idea. The knowledge engineer — a (typically) professional expert system builder whose task it is to analyse and model the problem-solving capabilities of an expert in some specialist domain — is an impressive learner. At the outset of a new project knowledge engineers characteristically know very little about the

domain. To produce the computer system they can and do become 'near expert' (O'Keefe & O'Leary 1989). Thus knowledge engineering can be seen as a kind of self-directed, constructive, active-learning activity (Clancey 1988). For Nydahl (1991) and other researchers in knowledge-based modelling, knowledge engineers provide a direct role model for classroom learners.

Of course, the analogy between children and knowledge engineers has limits. A knowledge engineer brings to each new domain an already highly developed set of skills in acquiring, structuring, representing, and interpreting knowledge. These skills, which many educationalists would value highly, are much less highly developed in children. Research is needed to discover whether, and how, skills of this kind can be promoted by knowledge-based modelling. Another difference is that the knowledge engineer is in command of sophisticated computer tools. The crucial role of tools is suggested by the fact that much of the research in KBS has centred on them. For educational knowledge-based modelling, the development of tools is likewise a central area of research.

The tools for knowledge-based modelling that are most widely used in schools at present are rule-based expert system shells. These systems, which have their ancestry in EMYCIN (van Melle 1979), are now considered to be rather limited and inflexible (e.g. Jackson 1990). If the domain knowledge can be adequately represented in the shell's rule language, an expert system shell may be useful: but a shell itself provides little or no help in the development of the knowledge base. We argue that since children lack knowledge engineering skills, such tools make knowledge-based modelling unnecessarily difficult. More usable tools will be ones which provide explicit support for acquiring, structuring, and representing knowledge.

Producing new designs is almost invariably a formidable task. Fortunately, a field of KBS research — in automated knowledge acquisition — has already produced a body of helpful ideas. This field has tackled a problem that is related to ours: how to enable domain experts to communicate their knowledge to a computer system directly, without the need for intervention by a knowledge engineer. In a restricted range of tasks and domains,

the field has had some success. This provides a source of techniques which can potentially give children some of the support they need. Much of this thesis is an attempt to describe how tools can be designed that exploit these techniques in order to enable children to build models.

1.2 Main hypothesis and research questions

The main hypothesis of the research is that knowledge acquisition systems will enable more successful model building by children than has been possible with the EMYCIN-type shells. We test this hypothesis by designing and building a set of knowledge-based modelling tools that use knowledge acquisition techniques. We then compare children's performance with these tools to their performance with an established rule-based shell.

The research confronts us with a range of problems. For example, there is no universally accepted research methodology for this kind of project, so we need to explicate one. Knowledge acquisition systems are typically task-specific or domain-specific, so we must select a suitable task or domain. The comparison of children's performance requires criteria for successful model building, and so we need to define these. Following a review and appraisal of the context of the research in Chapter 2, we discuss these problems and others in Chapter 3. This leads to the formulation of the specific research questions which we investigate in subsequent chapters.

In preview, the research questions are as follows:

1. Which application area (task or domain) is suitable?
2. What representation formalisms and knowledge acquisition techniques are available for the selected application area?
3. What should be the approach to the development of tools?
4. What main principles and issues should be addressed in the design of tools?
5. How much time is required by children to build a model?
6. What problems do children have in using the tools?
7. How correct, efficient, and concise are the models?

8. Do children enjoy and feel confident in using the modelling tools?
9. Do teachers regard the modelling tools as successful?
10. Do children learn the representation formalisms that are embedded in the tools?

In the course of Chapters 4 to 8 we find answers to all ten questions. Classification is selected as the application area towards which the new modelling tools are aimed. We design three new modelling tools, based on different representation formalisms and knowledge acquisition techniques for classification, and these are evaluated by a combination of large-scale school trials with focussed studies of individual children. We provide extensive data which enables the effectiveness of the new modelling tools to be compared with that of the established rule-based shell.

In Chapter 9 the findings for each question are summarised and the limitations of these findings are described. We conclude that under certain conditions, the research hypothesis is valid. Although problems remain, it is clear that knowledge acquisition systems offer considerable potential to develop improved forms of educational knowledge-based modelling.

1.3 Main contributions of the research

The main original contributions of this research are as follows:

1. To investigate the contribution that can be made by knowledge acquisition technology to the design of educational tools for knowledge-based modelling.
2. To develop three specific designs for modelling tools which use knowledge acquisition technology.
3. To describe a set of measures for evaluating the extent of success of children's modelling.
4. To define a set of measures for analysing the quality of children's models.

5. To compare the extent of success of children's modelling using the knowledge acquisition tools with that using a rule-based shell.
6. To explicate a research methodology for projects in applied AIED.
7. To demonstrate that the analysis of models built with the tools can be performed semi-automatically.
8. To investigate applications for automated or semi-automated analysis of models.

1.4 Structure of the thesis

The remainder of the thesis consists of chapters which are summarised below.

Chapter 2: Context and appraisal

This chapter reviews the background to the present work, drawing on and linking together research in modelling, education, and knowledge-based systems. Although the technology of knowledge-based systems potentially has much to contribute to the development of new forms of educational modelling, our appraisal is that the tools currently in use have been successful only to a very limited extent. This is related to their poor usability. It is argued that tools are needed that give children much greater support in the model building process. Research in knowledge acquisition is identified that seems to represent a promising direction for the design of such tools.

Chapter 3: Questions and methods

This chapter describes the research strategy. The research hypothesis is that knowledge acquisition technology will enable better modelling. To test this hypothesis it was decided to design, implement, and evaluate new classroom modelling tools. An important issue concerns the methodology for developing tools: it is necessary to ensure a proper balance between the 'technology push' and the 'learning pull'. It is claimed that the Persistent Collaboration Methodology, a hybrid of user-centred design with action research, provides such a balance. The chapter explains

why classification tasks are an appropriate choice of application area and describes a strategy for implementation that will provide the new tools as software extensions to an existing expert system shell.

Chapter 4: Foundations for design

This chapter establishes foundations for the design of classification modelling tools. We discuss the place of classification in education and conclude that appropriate representations of classification for children's modelling may include decision trees, classification hierarchies, and factor tables. From the KBS literature we distinguish three inference methods used in classification and we explain why the methods of simple classification and systematic refinement are considered to be more suitable for educational tools than the method of heuristic classification. We survey some of the knowledge acquisition literature and identify a range of potentially useful techniques, including 'contrived' techniques such as laddering, knowledge editors, and induction. These considerations together lead to outline specifications for three tools, each of which is claimed to have the potential to be usable and useful. We explain why, although the three specifications are diverse, they should adhere to a set of overarching design principles relating to the user interface, compositionality, and runtime interpretation.

Chapter 5: The modelling tools

This chapter describes three new classification-oriented knowledge acquisition tools, all implemented as extensions to the Primex shell. The tools are: PDT, based on decision tree representations; PFT, based on factor table representations; and PCT, based on classification tree representations. We explain how each tool manifests the overarching design principles that were discussed previously. We describe some of the main design issues that were encountered in the development of the tools and we explain how these issues were addressed. Whilst avoiding low-level detail, we aim to give enough information to enable our designs (or parts of them) to be understood and even replicated if

necessary, so that the context of the experimental results which appear later in the thesis will be firmly established.

Chapter 6: Analysing the quality of models

This chapter describes how models can be analysed automatically or semi-automatically. Analysis of a model is necessary to assess its quality; it is the means by which a model constructed with one kind of modelling tool might be compared to a model constructed with another. We define three specific measures which we associate with quality: correctness, efficiency, and conciseness. For research of the kind presented in this thesis, in which it is necessary to analyse hundreds of models, manual analysis is hardly practical. We show how a computer-based model analyser can be designed which is an effective research tool with potential for classroom use. Also discussed in this chapter is a second application for model analysis, namely model comparison. An analyser that can identify differences between different children's models, and explain these difference in a way that might stimulate discussion, could enhance the value of modelling tools. We discuss the design and evaluation of a prototype model comparator.

Chapter 7: The school trials

This chapter covers the main empirical results obtained from the large-scale school trials. We describe the design of a modelling course which involved children in building models for a variety of domains. The modelling course produced 632 models representing the work of 82 pupils from five secondary schools. These models were analysed using the techniques described in Chapter 6. Since each child typically built two or more models for the same domain, using different tools, we were able to use the results of the analysis to compare the effectiveness of the established rule-based shell with that of the new knowledge acquisition tools. Other data came from children's recorded times for model building, teacher questionnaires, pupil questionnaires, pre-tests and post-tests. We use this data to test four hypotheses relating to model quality, build times, motivation and representational skill.

Chapter 8: Children building models

Compared to the previous chapter, this chapter offers an evaluation that is more qualitative and less formal. Using evidence from videotapes, dribble files, and informal observation, we provide detailed reports of episodes in which a small number of children build models with our tools. Each episode is interpreted and discussed. Individual children's comments are presented as are the views of the class teacher. These small-scale focussed studies reveal some of the difficulties that children experience, including difficulties in understanding domains, selecting representations, and constructing models. In the course of the chapter it becomes clear that the new modelling tools, although quite usable, could be substantially improved. We distinguish improvements which could be made almost immediately from those which will require further research. We also make suggestions about how teachers could design and supervise modelling activities in the classroom.

Chapter 9: Conclusion

This chapter summarises the results of the research and identifies some of its limitations. We then revisit the main research hypothesis and evaluate it in the light of our main findings. We compare the present research with two other related projects and identify the main original contributions of our research. Areas for future research are then described. Finally, we discuss the general prospects for knowledge-based modelling in education. Change in education is complex and influenced by many factors. Barriers to the uptake of new tools include the requirement for staff development and the need for additional research in teaching and learning. However, computer modelling in education seems to be at least partly in tune with some influential trends and developments. The strong focus within our research on teachers and children should ensure that our tools and findings are realistically grounded in classroom practice. The present thesis is seen as making a modest contribution to realising the enormous educational potential of knowledge-based modelling in education.

Chapter 2 Context and appraisal

This chapter reviews the background to the thesis, drawing on and linking together research in modelling, education, and knowledge-based systems. Although the technology of knowledge-based systems potentially has much to contribute to the development of new forms of educational modelling, our appraisal is that the tools currently in use have been successful only to a very limited extent. This is related to their poor usability. It is argued that tools are needed that give children much greater support in the model building process. Research in knowledge acquisition is identified that seems to represent a promising direction for the design of such tools.

2.1 Computer models

A model is commonly defined as an abstract or simplified representation of a system or process; a computer model is such a representation that can be interpreted by (or as) a computer program. Implicit in the requirement for interpretation by computer is that computer models are essentially *formal*. That is, they consist of statements that obey some well-defined textual or graphical syntax. The manner of their interpretation follows a predetermined procedure.

Computer models are at the heart of *simulations*, a class of applications that was one of the first to be investigated following the emergence of digital computing in the post-war period. Indeed the terms 'model' and 'simulation' are sometimes used interchangeably. A useful distinction however can be made between 'black-box' and 'glass-box' models (e.g. Anderson 1988). A simulation provides a black-box model; its users may be at liberty to adjust its inputs but the fundamental assumptions and purposes of a simulation model cannot be changed. By contrast, the models that we describe in this thesis are constructed by users, not merely run by them, and so are definitely of the glass-box type.

2.2 Qualities of computer models

The overall literature on computer modelling is extensive. The textbook by Davies & O'Keefe (1989) for example provides over eight pages of references covering topics in methodology, statistics, languages, programming, applications, visual input and output, and artificial intelligence. This reflects the wide range of domains that have been the subject of computer modelling, for example weather forecasting, financial models, engineering models, and population models, and the great variety of methods and languages that have been developed. As with much of computing, the field's practitioners find much upon which they can disagree but there is consensus too on many important points, including the following.

(i) Computer models may be helpful for the purposes of comparison, prediction, or investigation within a domain, particularly when experiment with the 'real-world' counterpart is impractical for some reason (e.g. because it does not exist, or would be too dangerous for experimentation).

(ii) Computer models may be classified in various ways. Common categories include: deterministic or stochastic (the latter contain some randomness); static or dynamic (the latter contain a notion of advancing time); if dynamic, discrete or continuous (in the former, events can occur only at fixed time intervals).

(iii) Computer modelling like most forms of technology has potential problems. Effective models may be expensive to construct and maintain. A model may be ill-founded in the sense that its representation of the domain is over-simplified or otherwise inadequate to support the uses that are made of it and furthermore this ill-foundedness may not be easy to detect (Boos-Bavnbeck 1991). Models may be constructed for dubious purposes, for example to support control by organisational or political elites (Dutton 1987).

2.3 Tools for model building

Since computer models are formal representations, the tools that are used for their construction are essentially programming tools. Early work in computer modelling typically used a standard programming language such as FORTRAN. Subsequently simulation-specific languages such as GPSS (Schriber 1974) and SIMULA (Birtwistle et al 1979) were developed which provided facilities for sampling, entity definition, and so on; even more importantly these languages, because they were based on a particular (i.e. process-oriented) view of simulation, incorporated a general-purpose run-time interpreter. The burden on the programmer was thus reduced, at a cost insofar as the facilities for describing the model were somewhat limited and inflexible and the run-time interpreter was not always acceptably efficient. Still, the tradeoff was often acceptable.

The desirability of separating the model description (the programmer's responsibility) from the model interpretation (the system's) is reflected in more recent simulation model building tools. For example STELLA (Richmond et al 1987), a dynamic systems modelling toolkit, provides a model description language based on state change diagrams plus algebraic equations and a runtime interpreter based on the iteration of the equations. Ideally, the modeller's attention should focus on the validity of the domain description without concern for how efficiently the description is interpreted by the machine. This is nicely put by Miller et al (1991) who characterise the distinction between modelling and programming thus:

> *Modelling is to do with describing the world, whereas programming is to do with instructing a computer.* (p177)

In practice, the ideal is not always reached. Even spreadsheets, which today are the most widely used tools for numerical modelling, require the modeller to take some responsibility for efficiency; for example by avoiding 'loops' in cell definitions and by accepting manual control for cell recalculation with a large or complex set of formulae.

2.4 Knowledge-based systems

The idea that modelling is 'to do with describing the world' provides a connection between computer modelling and the field of knowledge-based systems (KBS). The history of the latter has been fairly well documented, by Szolovits (1991) for example. Early attempts at building programs to automate problem solving in specialised domains, such as MACSYMA for symbolic manipulation and DENDRAL for the interpretation of mass spectrometer data, had reasonable success and suggested that similar programs (later called *expert systems)* could be constructed for other domains. A standard architecture emerged, based on the organisational separation of domain knowledge — the knowledge base — from the procedures (inference engine) that would interpret it. There was heightened interest in declarative formalisms, often logic-based, for representing and processing knowledge. Many difficulties were uncovered by attempts to build expert systems in complex domains and these opened up investigations in new areas, notably in knowledge representation, inference methods, explanation, and knowledge acquisition. All of these remain active research areas today.

It will be clear that knowledge-based systems and modelling have in common a concern for domain description and a recognition that such description should be organisationally separated from interpretation. Not all computer models are knowledge-based (for example, statistical models may not be) but most if not all knowledge-based systems can be regarded as models of a particular kind, i.e. as models of expertise, knowledge or belief:

> ... *today knowledge engineering is approached as a modelling activity: the heart of the work of the knowledge engineer lies in the actual construction of models.* (Wielinga at al 1992 p1).

Many of the techniques uncovered by KBS research have led to new tools for simulation and modelling. Examples are T-Prolog (Futo and Szeredi 1982), ROSS (McArthur et al 1986), KBS (Reddy

et al 1986), ECO (Robertson et al 1989), SimKit (Cypher & Stelzner 1991) and SAM (Hensgens et al 1993).

2.5 Modelling in education

Computer modelling in education now has a substantial literature. A recent review written from the perspective of a major UK ESRC-funded project is provided by Mellar et al (1994). A distinction which the authors of that work recommend is that between 'exploratory' and 'expressive' modes of learning with computer models. In the exploratory mode, the model is provided to the learner and the learning activity involves investigating its properties. In the expressive mode, learners build their own models. The two modes may reflect different uses of a tool, rather than differences between tools, in that some tools equally well support both modes. For example, a spreadsheet might be given to learners as a blank worksheet, or with formulae and data in place. The exploratory/expressive distinction is a helpful one in reminding us that the term 'modelling', which in this thesis can be normally understood to mean 'model building', might be interpreted differently in other contexts.

What is the rationale for children building their own computer models? The literature identifies four main learning outcomes that modelling can address, as follows.

(i) **Domain learning:** Building a model of a domain is likely to involve investigating and reviewing domain subject matter, abstracting key ideas and relationships, and formulating concepts with precision. Such activities are believed to help the model builder to learn about the domain. Underwood & Underwood (1990) in arguing for computer model building put the idea concisely:

> *One of the most effective ways of understanding any body of knowledge is to reconstruct it.* (p64).

The notion of learning as the construction of knowledge is the essence of *constructivism* which is a central plank in current

cognitive psychology (Resnick 1989). In contrast to earlier behaviourist psychology, which supported a view of instruction as the transmission of knowledge, constructivism insists that learning occurs not by recording or absorbing information but by a process of active mental interpretation in which new cognitive structures are built by reflecting on experience and by making use of existing cognitive structures. Under constructivist theory, the aim of instruction is to support learners' own knowledge construction: it is not possible to 'deliver' preconstructed knowledge. It should be stressed however that constructivism is not one theory but a family of theories. Its roots are usually traced back to the work of Piaget (e.g. Piaget 1941, Piaget 1974) but modern presentations of constructivism are influenced to varying degrees by post-Piagetian theories, including the 'social constructivism' of Vygotsky (1978), theories of mental models (Johnson-Laird 1983), and 'situated' theories which emphasise the importance of the context in which learning takes place (Lave & Wenger 1991).

Model building as a classroom activity seems well matched to constructivist learning theories. The case was vigorously made by Papert (1980) who advocated the use of the Logo language particularly for learning in mathematics. Papert now calls his version 'constructionism' which he says 'attaches special importance to the role of constructions in the world as support for those in the head' (Papert 1993, p143). Logo's popularity seems to have declined but model building in other forms continues to attract support as an activity for domain learning. The contributors to Mellar et al (1994) describe modelling with spreadsheets, expert system shells, and dynamic systems modelling tools, in a range of domains including mathematics, animal populations, traffic flow, diet, and the classification of bones.

(ii) **Skills in modelling:** Modelling is now an important part of the practice of many disciplines, for example engineering, geography, chemistry, and economics, and so it can be argued that modelling skills *per se* should be taught and learned. An example of this view from a science educator is Wedekind (1993) who states that:

> ... *development and analysis of models ... is a basic scientific method, widely used in research and planning .... It would be logical then to regard modelling as an integral part of science teaching. Students ... should have not only theoretical knowledge about modelling but also practical skills in the development, testing, validation and analysis of models.* (p287).

(iii) **Transferable skills:** In the 1980's strong claims were made for the benefits of learning computer programming, with predictions that Logo programming in particular could develop complex cognitive skills in planning, problem-solving, logical reasoning, and self-regulation that would transfer to other domains (Lochead & Clement 1979; Papert 1980; Nickerson 1983). Subsequent research to test these predictions produced conflicting results (Liao & Bright 1991). Papert objects that his views were misrepresented and that he saw and continues to see Logo:

> ... *as a* means *that can,* in principle, *be used by educators to* support the development *of new ways of thinking and learning. However, Logo does not itself produce good learning any more than paint produces good art.* (Papert 1993, p.xiv; his italics).

Recent evidence to support this comes from Swan & Black (1993) whose studies indicate that although Logo programming alone does not develop problem-solving strategies, a combination of explicit strategy teaching plus Logo programming is effective. Apart from Logo, similar sorts of claims concerning transferable skills have been made for other forms of model building. Wideman and Owston (1993) claim that knowledge-base construction may develop transferable skills in inferential reasoning, self-monitoring and strategy selection. Their study of high school students provides some limited support for these claims.

(iv) **IT skills:** Chipman (1993) notes that during the 1980's the usage of computers in US schools tended to shift from the initially intended uses in subject-matter instruction towards computer literacy instruction. A similar trend was discerned in Scotland (Conlon & Cope 1989) where more recently a debate

about the nature and importance of IT skills has taken place between the advocates of discrete courses in IT and others who would prefer a more integrated and cross-curriculum approach (Bird, Conlon & Swanson 1996). The proponents of modelling generally tend to regard the development of IT skills as of lesser importance than other outcomes but they have been willing to argue that if IT skills are the objective, then model building activities are as good a way to get there as any other (Mellar et al 1994).

With a few exceptions, the empirical evidence for the effectiveness of computer modelling in achieving these four outcomes is thin. There may be several reasons for this. Education is a long term activity: the impact of any one educational treatment may not manifest itself for several years (Snow & Yallow 1982). Computer modelling is relatively new and with the exception of the Logo work, very little research has been done to evaluate outcomes experimentally. Evaluations of the general impact of IT in education have faced disputes about the aims and priorities of education (Hammond 1994). Some commentators, for example Papert (1993), argue in any case that the 'standard experimental method' in which one variable is manipulated whilst all others are kept constant is inappropriate to the context of computers in education. Similar difficulties and disagreements have affected the evaluation of Intelligent Tutoring Systems (Twidale 1993; Murray 1993).

In the UK, the potential of modelling as an educational activity has been at least partly recognised by curriculum reforms. The 1990 National Curriculum of England and Wales included modelling in the attainment target for information technology capability (DES and Welsh Office, 1990) and in Scotland the National Guidelines for the curriculum in the stages P7 to S2[1] recommend 'the preparation, use and evaluation of simple computer-based models using spreadsheets, databases and other such tools' (Scottish Office Education Department, 1993). The proponents of modelling have not always liked the specific

[1]In Scoland, P7 is the final year of the Primary school and S2 is the second year of Secondary school. These years cover the age range 11-13.

emphases within such recommendations (e.g. Mellar, 1990) but in the context of a centralised curriculum it seems better that modelling be represented imperfectly than not at all. Whether government support for computer modelling can survive its recent advocacy of more didactic teaching methods is unclear. For example Gammage (1996) argues that recent DES curriculum policy has been regressive and that prospects for 'child-centredness' in (English) Primary education are bleak.

2.6 Knowledge-based modelling

The application of KBS techniques to education contexts has a fairly long history. It was a natural step from the first expert systems to add some form of tutoring component so that the knowledge embedded in the systems might become available to students. Examples are MACSYMA Advisor (Genesereth 1982) as an extension of MACSYMA and GUIDON (Clancey 1982) as an extension of MYCIN. Subsequent research and development in Intelligent Tutoring Systems (ITS) has continued to draw heavily on techniques from KBS.

Workers in ITS, however, have followed a historically different research path from that which led to the emergence of KBS-inspired model building environments for children. The latter emerged from work by the Prolog Education Group (PEG) and others who in the early 1980's investigated the potential of the logic programming language Prolog as a programming language for children (Ennals 1983; Conlon 1985; Nichol et al 1988). Prolog was seen as a rival to Logo, which (perhaps unfairly) was associated with procedurally-oriented programming. Prolog's declarative semantics was claimed to give the language an important advantage (Kowalski 1984) and seemed to justify use of the term 'knowledge bases' when referring to programs in Prolog. In practice, however, Prolog proved difficult for students and children to learn and children struggled with its programming environment (Taylor & Du Boulay 1987; Cumming & Abbot 1988). Reflecting on the experience with Prolog, leading PEG activists wrote:

The problem can be stated simply — to write a Prolog program of any sophistication requires a long-term logic programming course. Programs consisting of simple inputs of data rules were of limited application in children's learning of school subjects. (Nichol et al 1988 p149).

PEG group members produced modified versions of Prolog such as SIMPLE (Clark & McCabe 1984), MITSI (Briggs 1984) and EMITSI (Cumming & Abbot 1986). Increasingly these tools resembled expert system shells. By the end of the decade many PEG group members had largely transferred their attention to expert system shells. Shells produced by the group included Adex Advisor (Briggs 1987), Primex (Conlon 1991) and Expert Builder (Webb 1993). Prolog's role became that of a development language for researchers rather than a medium for direct use by learners. This process has been discussed by Conlon (1991).

Expert system shells descend from EMYCIN (van Melle 1979), a domain-independent offshoot of the MYCIN expert system. An EMYCIN-type shell provides an interface or interfaces for building and consulting knowledge bases, a rule-based knowledge representation language, and a backward-chaining inference engine. The PEG shells named above are all EMYCIN-type shells. By the middle of the 1980's a growing range of such shells was being heavily promoted commercially for the booming personal computer market (Simons 1985). Shell vendors tended to suggest that, compared with building an expert system from scratch with (say) Lisp or Prolog, use of these products would generate substantial savings in development time and would bypass the requirement for professional programming support. The first of these claims may have been justified but the second was dubious, as we argue later.

2.7 Schools' experiences with knowledge-based modelling

In reviewing the development of computing in education, Chipman (1993) offers some recommendations for the future. Although conceding that 'the view through the silicon chip is clouded', she is clear about the kind of agenda which should be pursued:

The challenge is not breaking the 'icons' of colleagues in educational R&D. The challenge is to develop innovative instruction that is actually used in the nation's schools and has a significant positive effect on the learning of a significant number of students. (Chipman 1993, p363).

If this challenge is to be accepted in the name of knowledge-based modelling then an essential place to start would seem to be research into schools' experiences with *existing* versions of the technology. We undertook such research in 1993/94 among secondary schools in south-east Scotland. The findings have been fully reported in Conlon & Bowman (1995) and we only summarise them here.

(i) **Ownership of shells:** Questionnaires were dispatched to thirty secondary schools randomly selected from the south-east area of Scotland. Of the seventeen questionnaire responses, only three came from schools which said that no expert system shells were owned by the school. The other fourteen said that they owned between one and six different brands of shell (see Table 2.1). Twelve schools had purchased multiple-machine licence agreements for at least one shell.

Number of brands of shell owned	*0*	*1*	*2*	*3*	*4*	*5*	*6*
Number of schools	*3*	*7*	*4*	*1*	*1*	*0*	*1*

Table 2.1 School ownership of expert system shells

(ii) **Brands and platforms:** From the same sample, eight different brands of shell were present (see Table 2.2).

Brand of shell	*No. of schools owing brand*
Primex	9
Adex Advisor	6
Expert Builder	4
Flex	4
Mike	3
Apes	1
Refine	1
Esie	1

Table 2.2 Brands of shells owned

All shells ran under the MS-DOS operating system apart from Primex, which was running on both Macintosh and MS-DOS machines, and Expert Builder which was running under Microsoft Windows only.

(iii) **Usage:** Of the fourteen schools in the above sample which owned at least one shell, one reported that no active use was being made of the software; thirteen reported active use within the subject of Computing Studies; none reported active use within a subject other than Computing Studies.

(iv) **Usability:** Teachers' questionnaire and interview responses showed that they did not expect significantly extended use of the shells by their colleagues in the foreseeable future. Using the shells required training and support. Children found the shells difficult to use. Apart from lack of usability, a shortage of suitable hardware and pressure on teachers to cover syllabuses in the available time were factors in limiting uptake.

(v) **Children's knowledge bases:** We inspected informally 102 knowledge bases that had been built by children. The majority were single-rule or few-rule extensions of knowledge bases supplied as examples by the shell's manufacturer. A minority were novel creations; mostly these were on informal subjects and contained between three and ten rules. The largest knowledge base contained sixteen rules. Almost all of the knowledge bases inspected had a flat structure and made no use of variables. Several contained obvious syntax errors. Many ran only very inefficiently (e.g. because of inconsistent typing of rule conditions that were evidently intended to be equivalent).

(vi) **Teacher's experience/competence:** We identified a sample of schools that were Primex shell owners. Schools were sent a questionnaire to be completed (anonymously) by the teacher who had the most experience of using the shell within the school. Questionnaires elicited information about extent of use and also invited teachers to correct a printed knowledge base which (they were told) contained errors. Of the twelve

teachers who responded, all used Primex either a few times per term or per year: there were no instances of daily, weekly or monthly use. Correction of the knowledge base was on the whole very poorly done and revealed that teachers were not familiar with the shell's knowledge representation language beyond its most basic features.

Our survey probably gives a somewhat exaggerated picture of activity with expert system shells in Scottish schools. It ignored Primary schools (which we knew to lack sufficiently powerful hardware to run the shells), was centred on the mostly affluent and urban south-east area of Scotland, and the questionnaire return was voluntary and incomplete. Nevertheless, the picture that emerges is of a status quo that reflects at most a very modest success for knowledge-based modelling.

Optimistically, it could be said that schools in Scotland have purchased shells in some quantity. In doing so they have selected the education-originated shells such as Primex or Adex-Advisor more often than the commercially-developed products such as Flex. The purchased shells have not become 'shelfware'. They are in active if infrequent use. There are teachers who care about them and there are children who spend time with them. Considering the rather low general levels of use of IT in secondary schools,[2] this is an achievement and a basis on which to build.

Realistically however we must recognise a curriculum uptake that is extremely narrow. There are very clear signs that children and teachers find the shells difficult. Only specialist IT teachers are using the shells and of the possible learning outcomes for modelling in education described earlier, only one — viz., the pursuit of IT skills — is being addressed. The models that children build describe not curriculum domain knowledge but informal topics. Table 2.3 summarises the wide gap that exists between the current state of practice and an envisaged 'ideal' state from the viewpoint of a proponent of knowledge-based modelling.

---

[2]Surveys by the Department for Education and Employment in 1989 and 1994 respectively recorded 32% and 34% of secondary teachers making regular use of IT. However Margaret Bell, the chief executive of the National Council for Educational Technology, describes these figures as 'too optimistic' (Times Educational Supplement, Computers Update 18/10/96 p6).

	Current state of practice	*Envisaged 'ideal' state*
Users	IT specialist teachers	Subject teachers from many curriculum areas and stages
Intended learning outcomes	IT skills	Domain knowledge Skills in modelling Transferable skills
Models	Informal topics	Curriculum topics

Table 2.3 The gap between existing practice and an 'ideal' state

2.8 Advances in knowledge-based systems

As noted earlier, in the mid-1980's a large expansion occurred in the market for microcomputer-based expert system shells. The number of industrial and commercial projects also rose. Durkin (1993) provides a survey of 2500 developed expert systems, a figure which he suggests represents only about 20% of the real total. Almost half were produced on desktop computers using expert system shells. Durkin (1994) writes:

> *During the 70s, most expert systems were developed on powerful workstations in LISP, Prolog and OPS5. This left system development to a select few .... The 80s brought the proliferation of personal computers and easy-to-use expert system shells. People from a variety of disciplines could now develop expert systems.* (Durkin 1994, p38).

However, Durkin does not say how many of these 'people from a variety of disciplines' had the support of computer specialists or were computer specialists themselves.

Perhaps in contrast to Durkin, many writers on KBS have emphasised the *difficulties* involved in building systems. The term 'knowledge engineering', which is now commonly used to describe the process of constructing an expert system, is suitably suggestive of a rather complex activity. Evolutionary prototyping, one of the least prescriptive approaches to knowledge engineering, is depicted in Appendix 1. None of the stages in this process is trivial and it will be noted that the approach involves a *collaboration* between computer specialists (knowledge engineers) and domain experts. The availability of EMYCIN-type expert

system shells has not made the knowledge engineers redundant, for several reasons including the following:

(i) EMYCIN-type shells offer no help with the stages of knowledge elicitation and knowledge structuring. These stages, which together are usually (but not always consistently) referred to jointly by the name of *knowledge acquisition*, were famously described by Feigenbaum (1977) as the 'bottleneck' of expert systems development. Knowledge engineers are needed to help experts to articulate their knowledge and to structure it into a form suitable for implementation.

(ii) Beyond providing an editor and a rule-based language syntax, neither do EMYCIN-type shells support knowledge representation. Unfortunately the task of converting even a well-structured body of knowledge into rule form may be far from straightforward. Learning the shell S1 (an extended EMYCIN-type shell) has been described as comparably difficult to learning a programming language such as Pascal (Ward & Sleeman 1987). Domain experts typically lack programming expertise and look to knowledge engineers to provide the necessary skills.

(iii) Rule-based representations often do not make important distinctions explicit. For example, it is common that some strategic knowledge of problem-solving is implicit in rule order. This makes more difficult the tasks of understanding, debugging and maintaining rule-based systems (Clancey 1983, Aikins 1983, Alvey 1983) and justifies the continuing involvement of a knowledge engineer even after the expert system has been initially produced.

None of this is to deny that expert system shells can be helpful tools when they are appropriately used by suitably trained individuals. But children probably do not fall into this category. Arguably, it never was reasonable to expect that a child (or anyone who is a domain and computer novice) could successfully conflate the roles of domain expert and knowledge engineer, perform knowledge self-elicitation, apply appropriate

structuring, represent the resulting information in formal rule syntax, and test and debug a finished knowledge-based system. If knowledge-based modelling is to progress then tools must be found which give children much greater support in the model building process.

In fact, KBS researchers left the EMYCIN trail long ago. The literature of more recent advances is vast but it will be helpful briefly to indicate three (not exclusive) main lines of work and to evaluate these from an education perspective.

(i) **Methodology:** As noted above, expert systems may be developed by the method of evolutionary (or rapid) prototyping. Kingston (1992) notes that this was *the* way to develop a KBS in the mid-1980's. But whilst it has been often successful in research settings, evolutionary prototyping did not scale up well to larger projects in commercial environments and neither did it always make effective use of the knowledge that accrued over time about successful methods and designs. In Europe a methodology known as KADS (more recently revised as CommonKADS) has emerged which provides a structured, systematic approach to the development of knowledge-based systems somewhat analogous to established methodologies such as SSADM[3] for the development of conventional software (Tansley & Hayball 1993, Wielinga et al 1992).

The central theme of KADS is modelling. Development is recommended to proceed through four analysis models (of what the system will do) to three design models (of how it will be done). One of the analysis models is the expertise model which models the required problem-solving or expert behaviour. KADS assists in the production of this model by making available a hierarchical library of 'Generic task models' each of which is essentially an abstract model for a specific type of knowledge-based task, e.g. diagnosis or planning. In principle, creating an expertise model for a new KBS may amount to selecting a model from this library and adding to it an appropriate layer of domain

[3]Structured Systems Analysis and Design Method. A methodology for (mainly database-oriented) software development supported by the UK government.

knowledge (generally, static knowledge of basic facts, concepts, and structures). The realisation of a hierarchy of knowledge-based tasks is arguably one of the great achievements of KBS research. It is not unique to KADS: in the USA the Generic Tasks school has achieved a similar result (Chandrasekaran 1986).

Although KADS incorporates many ideas that might find some application to education, the methodology per se is primarily (if not solely) intended for knowledge engineers. Even computer specialists require training and 'a long learning curve' to become proficient in the KADS approach (Kingston 1995). Teaching KADS to children does not represent a way forward.

(ii) **Tools and environments:** No consensus has emerged on a single, universal language that is the best choice for representing knowledge in its many forms (if there is a consensus, perhaps it is that no such language exists). Neither has agreement been reached on an ideal architecture for the design of all types of KBS. This suggests two contrasting approaches to the design of tools and environments that are to support KBS construction. The first is to focus quite narrowly on a task or domain, identify typically useful representation languages and techniques for that task or domain, and integrate these more or less tightly to create a highly usable, but restricted, design. Examples of such systems are OPAL (Musen et al 1988) for generating plans for cancer treatment, MORE (Kahn et al 1985) for mechanical diagnosis and SALT (Marcus 1987) for configuration and scheduling. The second is to design for pluralism: provide a combination of representation languages, embed these rather loosely into the design of tools, and let the programmer decide how to select and combine components. Examples are KEE (Intellicorp 1984) and Knowledge Craft (Carnegie Group 1985) both of which combine production rules, frames, and logic representations, and KADS tool, CommonKADS Workbench, and VITAL, all of which are large-scale knowledge engineering workbenches intended to support KADS or a KADS-like methodology (a review of these systems is provided by Kingston et al (1995)). A common feature of the more recent tools has been their sophisticated interfaces, with extensive use of high-

resolution graphics to enable knowledge representations to be edited and visualised.

From an education perspective, the expressive power that is the object of the pluralist tools would probably not be usable. Such tools are primarily aimed at programmers. Knowledge-based modelling tools that are to be usable in schools seem more likely to emerge from a focussed approach to design, although the selection of task or domain could be critically important.

(iii) **Knowledge acquisition:** A very large number of methods and tools have been developed to support knowledge acquisition. Knowledge elicitation techniques are surveyed by Cooke (1994). She classifies techniques according to their 'mechanics' and distinguishes three families: techniques based on observations and interviews; process tracing techniques, such as talk-aloud and protocol analysis, that are generally performed concurrently with task performance; and conceptual or contrived techniques such as laddering, repertory grid, card sort and chapter listing, all of which use indirect methods to elicit domain relationships. Cooke notes that the contrived techniques tend to be restricted in the range of knowledge that they elicit. However, they have the advantage of being (to some extent) automatable.

A survey of knowledge acquisition tools is provided by Boose (1989). Table 2.4 is based on his dimensions for characterising tools. There are some patterns in the available tools that are surveyed. Far more of them support analysis (e.g. classification and diagnosis) tasks than support tasks in synthesis (e.g. planning and design). Tools that fully automate the knowledge engineering 'life cycle' are more likely to be domain-dependent. Many tools have been explicitly aimed at users who are domain experts: these tools are either domain specific, such as the examples of OPAL, MORE and SALT mentioned earlier, or domain independent but task specific, for example AQUINAS (Boose & Bradshaw 1987) and KITTEN (Shaw & Gaines 1987) both of which elicit classification knowledge by the repertory grid technique. The domain-independent tools that are aimed at domain experts generally make use of one of the contrived techniques, as described by Cooke (1994).

Application area	How domain-dependent is the tool? What categories of task can the tool address?
Representations	In what form must expertise be presented to the tool? What types of knowledge can be represented?
Knowledge acquisition techniques	What technique(s) does the tool support?
Model construction	Does the tool capture deep models? Causal models? Models with uncertainty?
System use	How much of the knowledge acquisition technique(s) is/are implemented in the tool? How much of the knowledge engineering life cycle does the tool support? Are the intended users AI programmers or domain experts? How much training is needed to use the tool?
Other features	Is there a learning component in the tool? If the tool supports multiple techniques, how well are they integrated? Does the tool support multiple experts? Does the tool provide multiple views of the knowledge?

Table 2.4 Dimensions for characterising knowledge acquisition tools

Although knowledge acquisition tools aimed at domain experts are clearly limited in scope and technique, they have been used successfully. Boose (1988) identifies around 200 small and medium-size knowledge-based systems developed with AQUINAS and its predecessor ETS. When used manually, contrived techniques such as laddering and card sort can be more effective in eliciting knowledge from experts in classification domains than the more widely used interview techniques (Burton et al 1988a, 1988b). More recently it has been demonstrated that automated versions of contrived knowledge acquisition techniques can generate KADS domain models with only occasional assistance from a knowledge engineer (Kingston 1995).

The rationale for knowledge acquisition tools is to diminish the role of the knowledge engineer, and to expand the role of the domain expert, in the development of a knowledge-based system. In the extreme case, which is still very exceptional, the knowledge engineer is eliminated and the domain expert builds the system without human assistance. This seems to represent a significant possibility for educational knowledge-based modelling. By diminishing the requirement for knowledge engineering skills — or

we could say, by embedding such skills within the computer tool — it seems possible that children will be more successful at model building than they have been with the EMYCIN-type shells.

2.9 Summary

It is of course possible to argue that computer modelling as an educational activity is in some fundamental sense unproductive, or that the techniques of knowledge-based systems have little to offer. Such arguments are pessimistic, however. As this chapter has argued, modelling in general is much in tune with constructivist theories of learning. Computers have an impressive track record as expressive tools in a wide range of contexts. Research in KBS has uncovered many powerful techniques for representing and processing knowledge.

Yet the practice of knowledge-based modelling in schools at present is very limited. Although a start has been made among a small group of teachers the gap between reality and potential seems very wide. Children and teachers generally do not find the tools which they have at present usable. However, these tools are EMYCIN-type shells that are far from the 'state of the art' in KBS. Perhaps it has been a classic case of what Guttag (1991) describes as the main reason for programming being too hard: 'In a nutshell, the wrong people are using the wrong methods and the wrong technology to build the wrong things' (p10). It would be very premature to dismiss knowledge-based modelling on the basis of schools' experiences with the systems that are presently available to them.

The design of better tools might be influenced by many of the post-MYCIN advances in knowledge-based systems. However the work in knowledge acquisition seems to be especially promising. The common design aim of all knowledge acquisition tools is to reduce the labour involved in building a system: they do this by automating some or all of the knowledge elicitation, knowledge structuring, and knowledge representation stages of development. In restricted application areas, knowledge acquisition tools already exist which enable domain personnel to build systems without the aid of knowledge engineers. It seems at

least plausible that knowledge acquisition systems will enable more successful model building by children than has been possible with the EMYCIN-type shells. The remainder of this thesis amounts to an investigation of this hypothesis.

The optimism which initially accompanied classroom work with first Prolog and later expert system shells proved to be unfounded. This underlines the need for caution in new investigations. As explained earlier, in the case of knowledge-based modelling the gap between existing practice in schools and an envisaged 'ideal' state is very wide. Furthermore we are not so naïve as to imagine that better tools, even supposing we can design them so as to achieve a radical increase in usability, will *necessarily* be adopted by teachers. Fullan (1991) characterises change in education as complex and generally depending on a combination of each of three factors: development of new materials, development of new teaching approaches and alteration of teachers' beliefs. New materials alone may not be sufficient to stimulate change. Whilst we do not dissent from Chipman's view (Chipman 1993, quoted earlier) that 'the challenge is to develop innovative instruction that is actually used in the nation's schools', neither do we underestimate the size of that challenge.

Chapter 3 Questions and methods

This chapter describes the research strategy. The research hypothesis is that knowledge acquisition technology will enable better modelling. To test this hypothesis it was decided to design, implement, and evaluate new classroom modelling tools. An important issue concerns the methodology for developing tools: it is necessary to ensure a proper balance between the 'technology push' and the 'learning pull'. It is claimed that the Persistent Collaboration Methodology, a hybrid of user-centred design with action research, provides such a balance. The chapter explains why classification tasks are an appropriate choice of application area and describes a strategy for implementation that will provide the new tools as software extensions to an existing expert system shell.

3.1 The research hypothesis

The previous chapter argued that the EMYCIN-type shells which schools are using at present give children too little support in the model building process. These shells require that children conflate the roles of domain expert and knowledge engineer, perform knowledge self-elicitation, apply appropriate structuring, represent the resulting information in formal rule syntax, and test and debug a finished knowledge-based system. The chapter also described how knowledge acquisition systems automate some or all of the knowledge elicitation, knowledge structuring, and knowledge representation stages of development. Therefore it was suggested that:

> *... knowledge acquisition systems will enable more successful model building by children than has been possible with the EMYCIN-type shells.*

This is the hypothesis that will be explored by the remainder of the thesis. The hypothesis is a prediction about the behaviour of human beings interacting with computer systems that before the research began, did not exist. Lacking relevant and

reliable theories, it requires to be tested empirically by designing and implementing knowledge acquisition systems and evaluating them when used by children. The evaluation will be essentially comparative: the existing EMYCIN-type shells provide the baseline for performance. To validate the hypothesis it will be necessary to find at least one knowledge acquisition system which compares favourably to the EMYCIN-type shells. We may anticipate that a favourable comparision will not be completely one-sided; a qualified success will nonetheless be welcome and indeed the nature of qualifying conditions might illuminate future theories of modelling with knowledge acquisition systems.

In the following sections, we develop a set of questions that the research attempts to answer. We also outline our research methods and indicate their limits.

3.2 Main questions: design and implementation

Chapter 2 observed that successful knowledge acquisition systems are typically limited in their application area to one domain or task. Not all domains and tasks may be suitable for the proposed new systems. For example a very esoteric domain may not be of interest to teachers and a very broad task may not be realisable as a knowledge acquisition system. So an obvious first question is:

- *Which application area (task or domain) is suitable?*

The selection initially requires the choice of either task-orientation or domain-orientation. If task-orientation is chosen, we must then select one kind of task; if domain-orientation is chosen, we must select one domain. As explained below, it will suffice for this research to select just one application area but the identification of alternatives will be useful for later work.

The second question relates to the design of tools. Referring back to Boose's characterisation of knowledge acquisition tools (Chapter 2, Table 2.4) we see that the application area is just one of several dimensions. It is necessary also to make choices for the other dimensions, two of which are clearly crucial, viz. those

concerning representation formalisms and knowledge acquisition techniques. Hence the question:

- *What representation formalisms and knowledge acquisition techniques are available for the selected application area?*

Eventually the research must generate (by some development process) designs to be realised as tools. The development process requires technical analysis but, no less importantly, it needs a methodology for generating, testing and refining designs. So, a third question is:

- *What should be the approach to the development of tools?*

A productive development process will be influenced by many sources, including the outcomes of previous research, analysis of problems and requirements, reflection on observations in formative trials, and discussion with teachers, students, and KBS experts. Hopefully, out of this process will emerge not only a set of particular designs, but also some general design principles and design issues that may be of wider interest to future researchers and developers. Hence, another important question will be:

- *What main principles and issues should be addressed in the design of tools?*

An issue arises as to how many tools should be constructed. A reasonable answer seems to be at least two and at most four, for the following reasons. At least two tools should be built because the space of suitable tasks, domains, and design possibilities may be very large and although validation of the hypothesis requires only a single successful tool, we may need to build more than one tool to find it. Furthermore, each implementation of a new design (even if it fails) will potentially contribute to a theory of modelling. But to attempt more than four implementations is unrealistic. System development is very time-consuming: even with an implementation approach which supports rapid development, the time to bring one tool to a level of refinement

suitable for classroom use is in order of several hundreds of hours. Evaluation adds further to the time requirement.

The tools that are built should be contrasting designs which all relate to the same application area. Varying the application area will make the results more difficult to interpret, since we may not know whether to relate differences in children's performance to the tool design or to the application area.

3.3 Main questions: evaluation

Evaluating a tool means assessing the extent to which it enables successful model building, compared to the EMYCIN-type shells. As a representative of the latter class we selected Primex, not only because our survey (Chapter 2, section 2.7) indicated that it is more widely used in local schools than any other but also because of our approach to implementation, as described below. But how should 'successful model building' be judged? There are at least four types of measure that might be used.

(a) **Process measures** represent observable features of the modelling process, for example the time taken to complete a model and the number of interactions that occur between model-builders.

(b) **Product measures** represent features of the models as end products, for example their correctness and efficiency.

(c) **Attitude measures** represent assessments of how the participants perceive the activity, for example whether children find using the tool enjoyable.

(d) **Learning outcome measures** represent cognitive change, such as change in a model builder's domain knowledge or transferable skills.

Ideally, data should be gathered on a very wide range of measures of all four types. In practice however it is necessary to be selective. Since our research aims to develop usable tools, we

emphasise measures of usability. Below we list the main evaluation questions that we address. Each question requires an answer for each of the implemented knowledge acquisition tools and also for the Primex rule-based shell.

Process measures

- *How much time is required by children to build a model?*
- *What problems do children have in using the tools?*

These measures will indicate the usability of the tool. To implement them requires that a number of specifications for model domains be prepared. These need careful selection, for example to ensure an appropriate level of difficulty and to reflect different ways of structuring information.

Product measures

- *How correct, efficient, and concise are the models?*

These are also measures which relate to usability. Again it is necessary to provide specifications for domains. The concepts of correctness, efficiency, and conciseness require tight definition.

Attitude measures

- *Do children enjoy and feel confident in using the modelling tools?*
- *Do teachers regard the modelling tools as successful?*

Enjoyment and self-confidence are indicators of motivational state (Malone & Lepper 1987). Expert teachers give high priority to developing and maintaining the motivation of learners. Teachers' views are important not only because of their expertise but also because teachers have considerable influence over the selection of classroom resources.

Learning outcome measures

- *Do children learn the representation formalisms that are embedded in the tools?*

Knowledge-based modelling involves constructing a representation of knowledge in some formal notation. There is evidence that the ability to construct representations is important in problem-solving (Greene 1989; Cox & Brna 1995) and some researchers have argued that the link is strong enough to justify teaching representation skills directly (Cox & Brna 1993). If children do learn or internalise (Salomon 1990) the representation formalism by activity with the modelling tool then it will be plausible to hypothesise that modelling enhances problem-solving ability. Furthermore, it will be possible to argue that schools' use of knowledge-based modelling is justified partly by its contribution to children's learning of representations. A proposal for future research along these lines has already been developed by Stenning et al (1996).

3.4 Outline of research methods

Table 3.1 summarises the main questions and broadly indicates the research methods that are used to answer each question. For convenience, the table's rightmost column identifies the chapters in this thesis where the relevant research is discussed. For a summary of all the main findings, see Chapter 9.

Of the methods selected for answering design questions, those for 1), 2) and 4) are standard research methods but those for 3) are not. The Persistent Collaboration Methodology and Extensible Primex were developed specifically for this project. These terms are explained later in this chapter.

Of the evaluation questions, the references to 'large-scale trials' and 'focussed studies' require explanation. The main aim of the former is to obtain a large quantity of models that have been built by many children under authentic classroom conditions using both the new modelling tools and the established shell. Analysis of these models should enable comparisons between tools to be drawn using the product measures. Such trials however may not reveal much about the difficulties which children experience in using the tools. Therefore focussed studies are used to identify these difficulties by detailed observation of a small number of individual children. The focussed studies will provide information

using the process measures. The large-scale trials and the focussed studies are described in much more detail in Chapters 7 and 8 respectively.

	Research Question	*Research Method*	*See*
1	Which application area (task or domain) is suitable?	Literature review; Questionnaires to teachers; Questionnaires to KBS experts.	Ch3
2	What representation formalisms and knowledge acquisition techniques are available for the selected application area?	Literature review; Consultation with teachers; Consultation with KBS experts.	Ch4
3	What should be the approach to the development of tools?	Persistent Collaboration Methodology; Extensible Primex.	Ch3
4	What main principles and issues should be addressed in the design of tools?	Literature review; Analysis of problems and requirements; Formative tests; Consultation with teachers, students, KBS experts.	Ch4 & Ch5
5	How much time is required by children to build a model?	Children's self-timings in large-scale trials.	Ch7
6	What problems do children have in using the tools?	Video recordings of children in focussed studies; Direct observation; Dribble file analysis; Reports from teachers.	Ch8
7	How correct, efficient, and concise are the models?	Analysis (semi-automated) of models returned from large-scale trials.	Ch6 & Ch7
8	Do children enjoy and feel confident in using the modelling tools?	Questionnaires to children in large-scale trials; Direct observation of effort; Children's comments in focussed studies; Reports from teachers.	Ch7 & Ch8
9	Do teachers regard the modelling tools as successful?	Reports from teachers in large-scale trials; In-depth comment from teachers in focussed studies.	Ch7 & Ch8
10	Do children learn the representation formalism that are embedded in the tools?	Pre-tests and post-tests of children's representational skills in large-scale trials; Observation of individual children in focussed studies.	Ch7 & Ch8

Table 3.1 Main questions and research methods

It will be noted that the evaluation does not include tightly controlled experiments. Although experimental, the large-scale

trials were not tightly controlled: they were conducted in normal class time by teachers in secondary schools without the presence of the researcher. As mentioned in Chapter 2 (section 2.5) there has been debate in the research community concerning methods of evaluation. We agree with Murray (1993) that tightly controlled experiments are more suited to evaluation at a later stage of research, when theories and designs are relatively stable and key issues are well understood. The questions in Table 3.1 suggest that for knowledge-based modelling, the present stage of research is as yet exploratory.

As said previously, we share Chipman's view (Chipman 1993) that a central challenge is to develop innovative instruction that is actually used in the nation's schools. Large-scale trials and focussed studies seem more likely than controlled experiments to give feedback on how our systems interact with, and should be designed for, real school contexts. The dichotomy is nicely expressed by Salomon (1993) who, reflecting on his experience of controlled experiments with the Writing Partner software, wrote:

> *To be sure, the study was well suited to test the theoretical proposition concerning the internalization of computer-afforded expert-like guidance and some of the conditions under which this process yields desirable results. But that study could tell us nothing about the way such a tool would affect and be affected by ongoing classroom practices. It was a study of computer tools* and *education, not a study of computer tools* in *education.* (Salomon 1993 p189, his emphasis).

The present study is intended as a study of the second kind.

3.5 Limits to this research

The selected research questions were designed to provide information about the practice and potential of knowledge acquisition systems in education. The use of large-scale trials and focussed studies seemed a good way to ensure that the feedback obtained was grounded in authentic contexts. However, there are limits to what the present research can deliver and some of these are discussed here.

First, this research does not aim to derive a theory of how children build models. Although answering the research questions may provide information that will be useful in developing such a theory, it falls far short of the programme of research in cognitive science that developing such a theory would require.

Second, this research investigates children's learning only to a very limited extent. We assess children's ability to learn how to use the tools (via questions 5, 6 and 7 in Table 3.1) and we assess their learning of representations (via question 10). Other forms of learning, such as domain learning, are not investigated. Again this is related to our view of the (exploratory) stage of the present research in which the proper emphasis is on developing usable tools. To provide any additional assessment of learning within the large-scale trials would not have been straightforward: the organisation of the trials, which we discuss in Chapter 7, was already difficult enough.

Third, although the research aimed to enable more successful model building by children than has been possible with the EMYCIN-type shells, there can be no absolute, incontrovertible validation that the aim has been achieved. This is not simply because our research questions are incomplete. As Salomon (1993) argues, the 'goodness' or 'badness' of a pedagogic tool does not simply lie in the tool:

> *... no tool is good or bad in and of itself. A tool's quality results from and contributes to the whole Gestalt of classroom events, functions, and factors in the context of which it is being used. It is only as part of the whole, well orchestrated learning environment, that a tool is of any value.* (Salomon 1993, p194).

One might add another item to Salomon's *Gestalt:* the philosophical stance of the classroom teacher who uses the tool and who in judging it good or bad does so in part by drawing on his or her attitudes towards and beliefs about the purposes of education. Thus David Carr, a philosopher of education, writes:

> *The greater part of practical and so-called theoretical enquiry with regard to education ... is concerned not with the direct technical solution of unproblematic educational difficulties but with ... complex issues of an evaluative and*

widely contested kind about the true nature of human flourishing. (Carr 1994, p91)

We do not expect in this research to resolve many differences about 'the true nature of human flourishing'. However, at least we can be explicit about the measures which are used for evaluation, so that the basis of any claims for success will be clearly exposed. Our data may be interpreted differently, but legitimately, by people with different contexts and philosophies.

3.6 Persistent Collaboration Methodology (PCM)

Our third research question, viz.

- *What should be the approach to the development of tools?*

— was one that had to be considered at a very early stage. Clearly, one aspect of the question concerns software architecture: this is discussed later. However, no less importantly the question also relates to the need for a methodology for generating, testing and refining designs. Perhaps surprisingly, no suitable ready-made methodology could be identified in the literature on Artificial Intelligence and Education (AIED). As we explain below, however, some general directions were clear, and it seemed feasible and useful in the course of this project to articulate them in a coherent form. The methodology that emerged, which we named Persistent Collaboration Methodology (PCM), is argued to be applicable to a rather wide range of projects. Specifically, PCM should be of benefit to projects in *applied* AIED, i.e. those that aim to construct useful educational products as well as developing theories and techniques. PCM has already been reported elsewhere (Conlon & Pain 1995, 1996). Here we only outline its main elements, aiming to provide sufficient detail to communicate the main contribution of PCM to the present project.

3.6.1 Rationale for PCM

Early literature on AIED (e.g. Sleeman & Brown 1982) gives

the impression of a straightforward answer to the above research question: that tools should be developed by researchers. However past practice, characterised by Clancey as 'developing tools in the isolation of our laboratories and delivering them to students and teachers' (1992, p158), is now widely regarded as inadequate to the task of developing applied AIED systems (Clancey 1993, Murray 1993, Salomon 1993). Such systems if they are to be successful must integrate into and interact with complex environments. Understanding of these contexts requires the participation of educators in the development process and is seen as a precondition of good design. Because designs and contexts interact in unforeseeable ways, the development process should be incremental and feedback-driven.

Outside of AIED, a recognition that development requires feedback from contexts of practice and participation by practitioners finds influential voice from many sources. Notable examples from education are theories of change (Fullan 1991) and the methods and principles of action research (Stenhouse 1975), whilst from the computer systems development literature strong advocacy comes from the proponents of the methods which are variously termed usability engineering, participatory design, socio-technical design, and so on (Winograd & Flores 1986, Ehn 1988, Greenbaum & Kyng 1991, Nielsen 1993).

Although the motivation for and general direction of a new methodology for applied AIED seemed clear, it had not to our knowledge previously been articulated as a systematic process. PCM is essentially an attempt to do that. It identifies the activities, aims, and products of collaborating partners (teachers, researchers, technologists, students) during the development process of a new AIED computer system.

3.6.2 The PCM development cycle

The development process with PCM is depicted in Figure 3.1. Development takes place in cycles represented by the large wheel. Each cycle has four phases: *reflecting* upon contexts, *designing* systems and practices (including incremental adaptation of designs), *acting* through implementation, and *observing* effects. An

individual project could sustain many iterations of this cycle and might start and stop at almost any point on its perimeter. The large wheel is shown to drive, and be driven by, a smaller wheel representing theory. This is intended to capture the idea that designs and theories are generated in parallel. System development both draws upon existing knowledge and contributes to knowledge beyond the immediate concerns of the project.

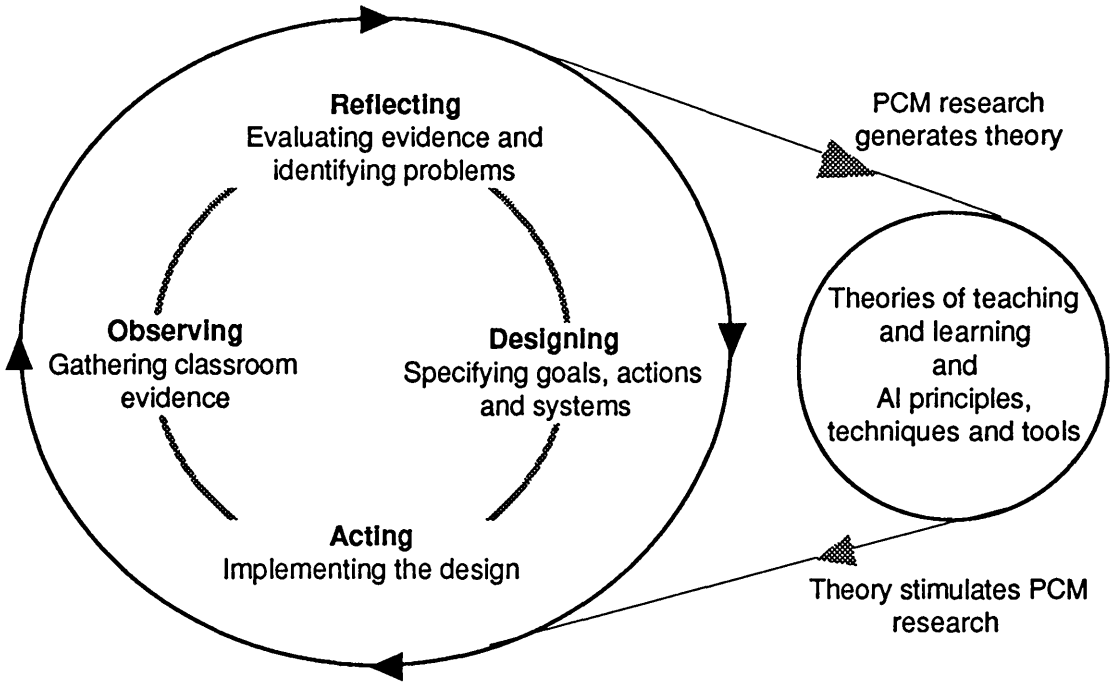


Figure 3.1 The PCM development cycle.

Collaboration occurs at each phase of the development cycle. Each party contributes distinctive knowledge and skills, as shown in Table 3.2, and can influence and be influenced by the contributions of others. A special responsibility of the researchers is to facilitate communication between teachers and technologists: in the ideal case technologists are deeply involved in the work of schools, but in practice their commitment may be less than this. Therefore researchers are needed who are 'bilingual' in education and AI and who can assist the growth of shared understanding.

The case for PCM can be related to the plea by Cumming (1990) that AIED projects should strive for a more equal marriage between the 'technology push' and the 'learning pull'. Laboratory-centred research methods are unlikely to arrange such an equal marriage because they put educational practice at the periphery

of development. In PCM on the other hand reflections on and observations of practice provide the main stimulus. Technologists and researchers are constrained to relate problems and solutions to the context of the classroom. Teachers have crucial roles as collaborating partners. At the same time, the methodology supports a mutually developing relationship between theory and practice. Thus PCM should enhance the prospects of individual projects and also contribute to progress in the AIED field in general.

Phase	*Teachers' aims*	*Researchers' aims*	*Technologists' aims*
Reflecting	Evaluate current classroom experiences; Identify problems of teaching and learning; Relate problems to theories of teaching and learning; Propose new theories of teaching and learning.	Assist teachers to evaluate classroom experiences; Relate problems to theories of teaching and learning; Propose new theories of teaching and learning; Help technologists understand classroom experiences, problems, and theories of teaching and learning.	Understand classroom experiences, problems, and theories of teaching and learning; Relate problems to AI principles, techniques and tools; Propose new AI principles, techniques and tools.
Designing	Set or revise educational goals; Set or revise teaching approaches; Contribute to the design or revision of systems.	Set or revise educational goals and teaching approaches; Contribute to the design or revision of systems; Help teachers to understand the possibilities of the technology; Help technologists to realise educational criteria for systems.	Acquire knowledge of educational goals; Acquire knowledge of teaching approaches; Explain the possibilities of the technology to teachers and researchers; Design new systems or revise existing systems.
Acting	Implement changed teaching approaches; Deploy the new or adapted systems.	Help teachers to introduce change; Help technologists to implement systems.	Implement new or adapt existing systems.
Observing	Keep notes of experiences; Look for patterns, anomalies, interesting cases; Discuss problems with students.	Assist teachers to gather evidence; Assist technologists to gather evidence; Look for patterns, anomalies, interesting cases.	Observe systems in use; Look for patterns, anomalies, interesting cases; Look for system defects.

Table 3.2 Aims of collaborators in the four phases of development.

3.6.3 The limits of PCM

Since PCM is new, its limits and tradeoffs are generally unclear. More work is needed to identify these and to refine the methodology generally. However, potential users of PCM in its present form should consider the following.

(i) PCM is time-consuming. Good communication between collaborators and a shared commitment to success are essential. It helps to have a project leader who has broad connections.

(ii) PCM is somewhat abstract. The tasks identified in Table 3.2 are in many cases complex (e.g. 'evaluate') and there is no detailed description of how they should be tackled. The methodology will require a lot of interpretation for individual projects.

(iii) The relationship between PCM and established software development methodologies like KADS and SSADM is not clear. It is true that the audiences are different: PCM addresses all groups in a collaboration whereas KADS and SSADM address only one group, viz. the technologists. Even so, the latter may wish to know how PCM affects their established methods.

(iv) In Conlon & Pain (1996) three classes of AIED research are distinguished: applied AIED (the class for which PCM is argued to be appropriate), cognitive science, and basic AI. It is conceded that the boundaries of these classes are not always easy to discern and that some projects straddle two or even three classes. Consequently, it may not always be easy to say whether PCM is appropriate.

(v) Any collaboration of the kind proposed by PCM is an inherently complex, dynamic, human process. Different participants bring different agendas. There is always the potential for conflict, domination by individuals and groups, and the disintegration of collective goals into the pursuit of diverse personal interests. Within PCM specifically there appears to be a

basis for tension in that the expected kinds of outcome of a project — developed theory, evolved classroom practice, and new systems, tools and techniques — might well be differently prioritised by researchers, teachers, and technologists. The methodology as yet has nothing to say about how such tension can be resolved.

3.6.4 This project's use of PCM

As noted previously, we regard PCM as a synthesis of existing ideas. The synthesis took shape in the course of the research as we began to design and develop knowledge acquisition tools. To the present researcher, incremental and collaborative approaches have long been familiar: reflection on the progress of the project and on the literature of development enabled the methods to be coherently articulated and abstracted. This in itself seems a good demonstration of one of the main tenets of PCM: that the generation of designs proceeds in parallel with the generation of theory.

A very detailed illustration of the PCM methodology applied to the present research is given in Conlon & Pain (1996). This documents one and a half cycles of the development process, starting at Observing and ending at Reflecting, at the stages of the research concerned with selecting an application area and designing the first tool. Because the aim of that illustration was to communicate the methodology it is presented in the somewhat idealised form of a rational reconstruction. The separation between phases is very sharp and the roles of different individuals are clearly distinct. In practice, of course, the process is more fuzzy, especially so since the present writer combined the roles of project leader, researcher, and technologist. It is not the intention in this report to provide exhaustively detailed coverage of this project's use of PCM (although its influence will doubtless be visible at many points). The interest here lies more in the research outcomes — one of which indeed is PCM itself.

It is stressed that PCM requires independent evaluation by other researchers in different projects. The present research may provide a helpful illustration of the methodology, but our central role in the project's leadership and the fact that the principles

were not articulated until mid-project makes it unreliable as a validation.

3.7 Selecting an application area

We now revisit the first research question. Successful knowledge acquisition systems are typically limited in their application area to one domain or task. Therefore it seemed important to ask:

- *Which application area (task or domain) is suitable?*

In this section we explain how this question was answered. First we discuss the choice between task-orientation and domain-orientation. Next we present evidence from curriculum documents, teachers, and KBS experts to justify the selection of classification tasks as the preferred application area.

3.7.1 Task-orientation versus domain-orientation

It seems quite likely that educational modelling systems could be constructed that employ domain-specific knowledge acquisition techniques. The examples of OPAL, MORE and SALT mentioned in Chapter 2 demonstrate that knowledge of a domain's general features can be embedded within a tool and utilised effectively to drive knowledge acquisition in specific cases. Similar techniques could perhaps lead to new classroom modelling tools in subdomains of (say) mathematics, geography or biology.

However, the present research aims to improve upon EMYCIN-type systems. These systems are not domain specific. Children can and do use shells such as Primex to build (limited) models in a range of domains. A domain-oriented tool could be expected to be much superior to Primex within its own domain, but outside of it there might be little or no competition.

Teachers with whom this issue was discussed generally expressed the hope that future knowledge-based modelling tools could be developed in a way that preserved the domain independence of the present shells. They pointed out that word

processors and spreadsheets, which are other examples of domain-independent software, had been quite widely adopted in schools: teachers value flexible tools. It can be argued also that an important trend in the development of the school curriculum has been to '... play down content in favour of skills and products in favour of processes' (Drever 1988), although it seems possible that this trend is now in decline.

Of course, task-orientation which was the alternative to domain-orientation also implied a restriction. The restriction might be acceptable however if the task was one that is valued by teachers across a range of subject areas and stages of schooling. A hierarchical classification of knowledge-based tasks based on the KADS methodology is provided by Tansley & Hayball (1993). The top level of the hierarchy contains 15 task types, viz: Diagnosis, Verification, Correlation, Monitoring, Classification, Prediction, Repair, Remedy, Control, Maintenance, Design, Configuration, Planning, Scheduling, and Modelling. It was suspected that several of these tasks (henceforth called 'KADS tasks') would have quite wide relevance to education. In the following sections we describe the research that was undertaken to investigate this.

3.7.2 Evidence from curriculum documents

In an attempt to identify the prevalence of knowledge-based tasks in the curriculum an analysis was undertaken of three curriculum documents, specifically those describing Scotland's 'national curriculum' for the 5 to 14 age stages of schooling in the areas of Environmental Studies, Mathematics and Expressive Arts. These documents were chosen for three main reasons. First, they have authoritative status. Second, they are explicit. Each Report characterises a curriculum area in terms of its strands of content, levels of study, and attainment targets. Third, they represent a wide range. The subjects covered occupy more than 50% of all curriculum time at the stages concerned.

Within each document, attainment targets were scrutinised (around 200 in total) and a frequency count was made of clear references to KADS tasks. Table 3.3 summarises the results.

	Env'tal Studies	Mathematics	Exp. Arts	*Total*
Classification	4	2	1	7
Prediction	3	0	0	3
Planning	3	0	0	3
Monitoring	0	0	1	1
Others	0	0	0	0
Total	10	2	2	14

Table 3.3. Frequency of references to tasks in curriculum documents

This exercise was felt to be of only limited value. Most curriculum targets are not couched in task language but only broadly indicate learning outcomes (such as 'draw conclusions...', 'give information about...', 'form generalisations...', 'decide on....'). In practice, teachers no doubt address some of these targets by designing tasks but the nature of these was not discernible from the curriculum documents.

3.7.3 Teachers' questionnaire

In order to gain a more insightful picture of the prevalence of knowledge-based tasks a questionnaire survey of teachers was conducted. A total of 350 questionnaires was issued to four secondary and four primary schools in the south-east area of Scotland. The main criterion in the selection of schools was the availability of a staff volunteer willing to distribute the questionnaires among colleagues and organise their return.

A specimen questionnaire appears in Appendix 2. The design of this questionnaire was problematic. Teachers who completed a pilot version asked that the meaning of the tasks should be fully explained and illustrated with curriculum-oriented examples, but it was feared that to do so might lead responses unduly. The adopted compromise was to provide a brief general description of each task together with some informal (non-curriculum) examples of the task's application.

There were 91 replies. Respondents were asked to indicate how frequently their pupils encountered each type of task, answering on a five-point scale ranging from 1='never' to 5='very often'. The mean responses for the whole sample are shown in Figure 3.2. As can be seen, planning emerged as the task with the

highest mean level of response. The lowest standard deviations of these responses were for planning followed by verification, correlation and classification. Highest standard deviations were for monitoring, maintenance and configuration.

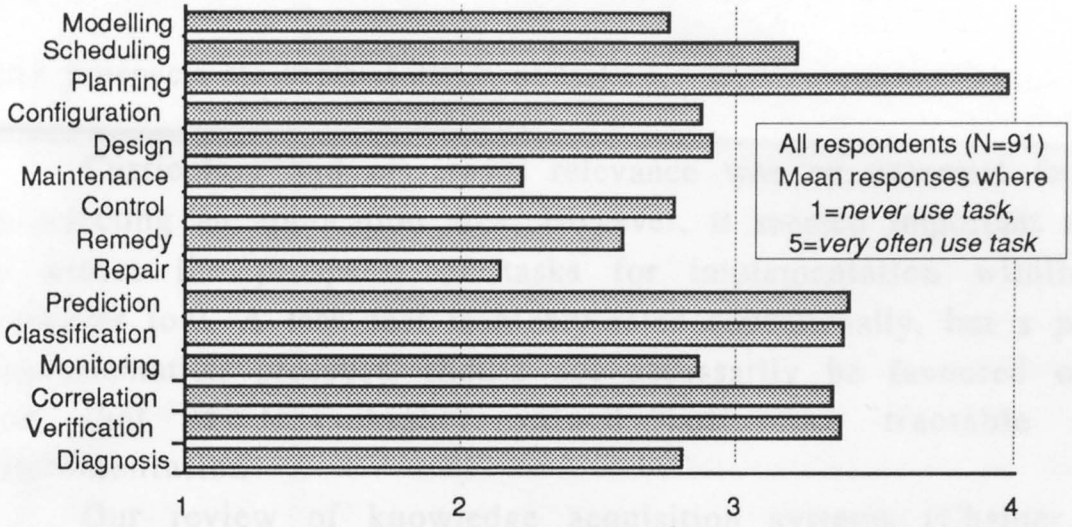


Figure 3.2. Frequency of task usage by teachers

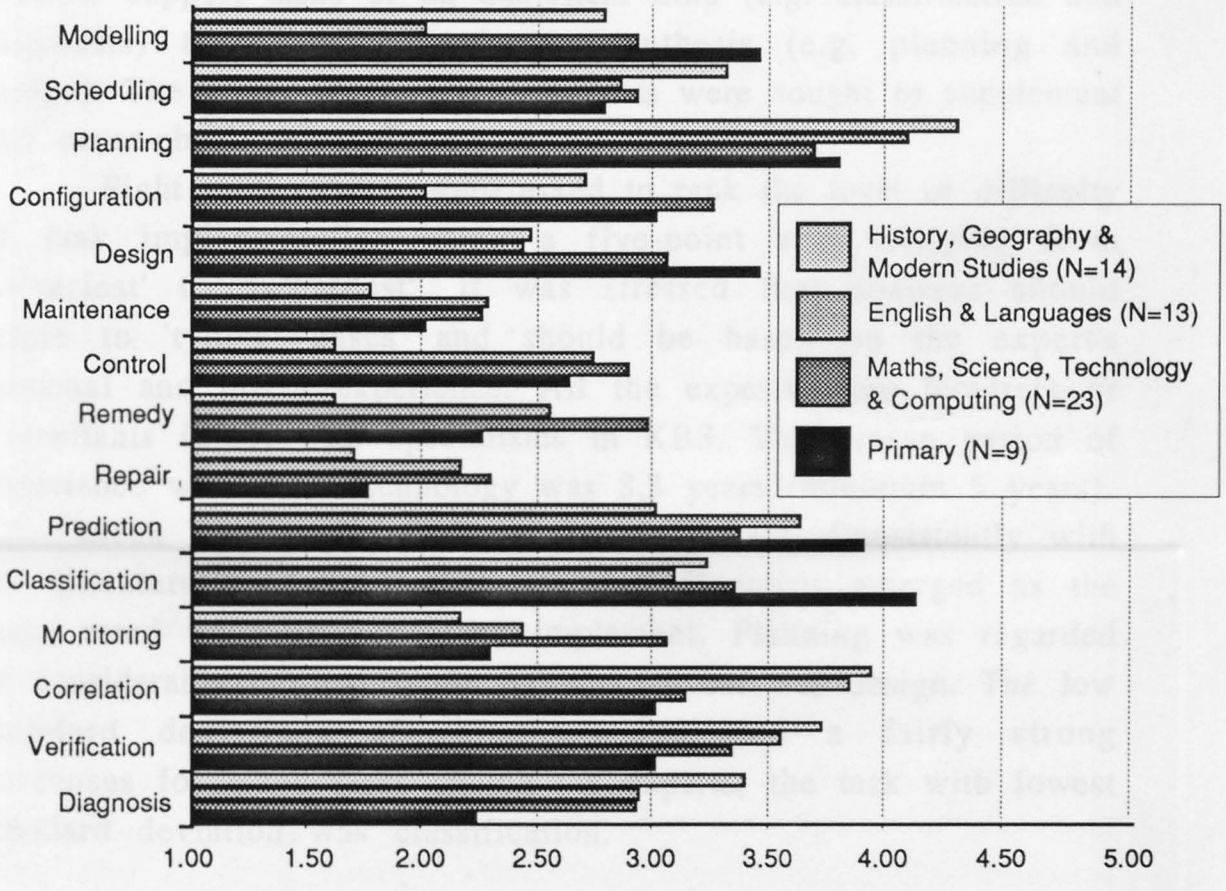


Figure 3.3. Frequency of task usage analysed by subject/stage.

Figure 3.3 summarises an analysis of mean responses grouped by stage and subject. It suggests that some tasks including planning, prediction, and classification are relatively frequently used across a range of school contexts.

3.7.4 Prospects for implementation of tasks

Curriculum and classroom relevance was an essential factor in selecting an application area. However, it seemed important also to assess the prospects of tasks for implementation within a computer tool. A task that is highly rated educationally, but a poor implementation prospect, should not necessarily be favoured over one that is less highly valued but more tractable for implementation.

Our review of knowledge acquisition systems (Chapter 2, section 2.8) had already provided some useful information. For instance, it indicated that more of the currently implemented systems support tasks of an analytical kind (e.g. classification and diagnosis) than support tasks in synthesis (e.g. planning and design). The views of local KBS experts were sought to supplement and cross-check this information.

Eight KBS experts were asked to rank the level of difficulty of task implementation using a five-point scale ranging from 1='easiest' to 5='hardest'. It was stressed that answers should relate to 'typical cases' and should be based on the expert's personal and direct experience. All the experts were lecturers or consultants in AI with specialisms in KBS. Their mean period of experience with KBS technology was 8.3 years (minimum 5 years).

Mean responses are shown in Figure 3.4. Consistently with the literature review, classification and diagnosis emerged as the tasks rated typically easiest to implement. Planning was regarded as considerably more difficult and the hardest was design. The low standard deviations of responses indicated a fairly strong consensus for many tasks among the experts; the task with lowest standard deviation was classification.

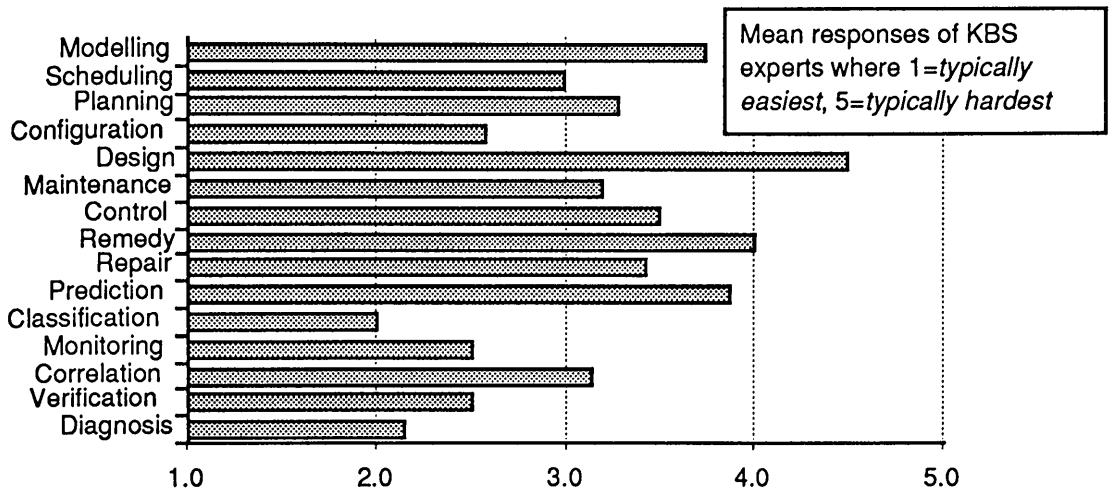


Figure 3.4. KBS experts' assessment of implementation difficulty

Notwithstanding the consensus, it is probably unwise to attach too much weight to the rank order of the results. Several of the experts reported difficulty in relating to 'typical cases' of tasks that in practice have a vast diversity of application. From the perspective of the present research, the main value of the survey is to provide general confirmation of the greater tractability of implementation of tasks in analysis relative to tasks in synthesis.

3.7.5 Final selection

We selected *classification* as a task that would be suitable for a knowledge acquisition tool for educational knowledge-based modelling. This choice was not a difficult one given the evidence described above. Schools report that they make use of classification relatively frequently and in a wide range of contexts. Its potential for implementation seems comparatively good. The fact that our surveyed curriculum documents mention classification more often than any other KADS task is not to be dismissed.

It was clear nevertheless that this selection process was imperfect. It made little specific reference to knowledge acquisition systems nor to the diversity of tasks in practice. The evidence, although favourable, was based largely on opinion. However, technologists and teachers were involved in a way that stimulated much discussion about the project's aims. Relationships were developed that were to prove invaluable at later stages of

the research.

Aside from our selection of classification, the data gives some interesting pointers. For example, planning and prediction look to be favoured educationally but they are rated by KBS experts as difficult to implement. In contrast, diagnosis is a KBS favourite but its educational appeal seems low.

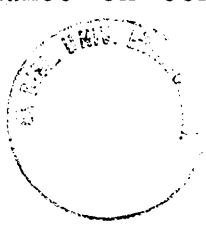
3.8 A strategy for implementation

The research at this point was committed to developing between two and four classification-oriented knowledge acquisition systems, plus support tools. The software engineering effort required for this would be large.

Without significantly pre-empting the discussion on the principles of design which appears in the next chapter, it is convenient at this point to describe our strategy for implementation. The strategy aimed to satisfy the following four conditions:

(i) As far as possible, tools should share interface features. Not only should each individual tool adopt consistent interface protocols, which Shneidermann (1987) describes as a 'golden rule' in the design of any interactive system, but there should be consistency also *between* tools (i.e. a strong 'family resemblance'). Clearly there were limits to this since different tools would utilise different knowledge acquisition and representation techniques. However, the more varied were the interfaces the more difficult it would be to relate children's performance to issues of knowledge acquisition and representation. According to Twidale (1993), the effects of the interface can 'overwhelm' those of other features of a computer-based learning environment. Furthermore, it was doubtful whether schools participating in the large-scale trials would be able to cope with several different tools each with its own highly distinctive interface.

(ii) In order to make the large-scale trials feasible, tools must run with acceptable performance on computer systems of a kind that schools possessed.



(iii) The software engineering process should be efficient and flexible. Tools should be engineered in a way that supports reusability of code and permits experimental design changes to be rapidly implemented.

(iv) Discussions with teachers elicited one further factor. As described in Chapter 2 (section 2.7) schools already owned expert system shells in some quantity. The teachers who would act as coordinators in the planned large-scale trials had invested time and effort in learning about the existing shells. Generally these teachers, whilst welcoming the prospect of improved systems, expressed the hope that they would not have to make a completely fresh start.

3.8.1 Extensible Primex

The final factor was particularly telling: it clearly indicated that an appropriate strategy would be to implement the new tools as modular extensions of one of the existing shells. An obvious choice for the latter was Primex, an Apple Macintosh shell which had been developed several years previously by the researcher.[1] Primex is relatively widely used in schools and of course we had access to the source code. An 'extensible' version of Primex was envisaged which would act as host to a set of codefiles that would implement the new knowledge acquisition tools together, with supplementary tools such as a dribble file utility. Code to be shared between extensions could be embedded permanently into the Primex base or made available as 'generic' extensions. All extension codefiles would be placed in an Extensions Folder to be automatically linked into the Primex base software when the user launched the Primex application. This architecture, which was termed Extensible Primex, is shown in Figure 3.5.

Extensible Primex offered considerable benefits to teachers

[1]Primex 1.0 for the Apple Macintosh was first released in 1990. The program was distributed commercially by the Publications Unit of Moray House Institute of Education. An MS-DOS port of the shell was completed in 1992. Apart from minor bug fixes there had been no further development of the software prior to the beginning of the present project.

and to the researcher alike. Teachers would continue to be able to use the familiar rule-based Primex shell alongside the new tools. From a user's viewpoint, Extensible Primex would appear just like the established Primex shell but for the addition of one new menu. If a knowledge acquisition tool was accessed via this menu, at least part of the tool's operation (e.g. opening and saving files, printing documents) would make use of Primex's regular interface features. The researcher gained a flexible experimental testbed which would be well-suited to a tool-by-tool comparative evaluation. Thus of the conditions discussed above, (iv) is fully satisfied and (i), (ii) and (iii) are satisfied in part.

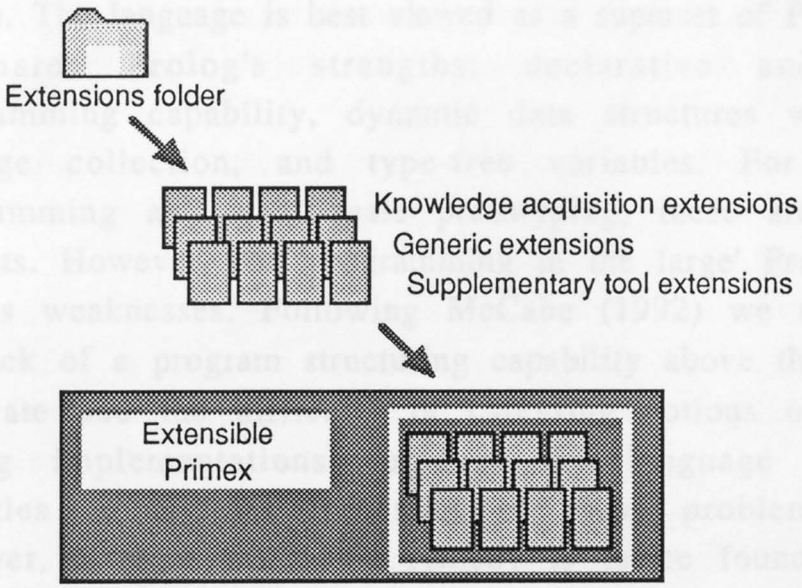


Figure 3.5 Extensible Primex architecture

Of course, the idea of an extensible architecture for a software application is not new. Many commercial software packages make use of extensions or 'plug-ins', in part because there are commercial as well as technical and user benefits to providing additional functionality in this way. However, software developers for education seem to have made little use of the idea. The notion of building on top of software that is already in classroom use has been advocated by Cumming (1993) but seems to have been not much adopted; the explanation may lie with technical difficulties or a lack of continuity in research.

3.8.2 Prolog++

Condition (iii) of the implementation strategy required an efficient and flexible software engineering process which would support reusability of code and rapid experimental adaptation. Extensible Primex provided part of the solution to this requirement. Another part was provided by the adoption of Prolog++ as the project's implementation language.

Prolog++ has been well documented elsewhere (Vasey et al 1990, Moss 1994) but a brief description may be helpful here. An example of a hybrid language design, Prolog++ combines concepts from logic programming (LP) and object-oriented programming (OOP). The language is best viewed as a superset of Prolog. As such it shares Prolog's strengths: declarative and meta-level programming capability, dynamic data structures with automatic garbage collection, and type-free variables. For experimental programming and small-scale prototyping, these are considerable benefits. However, for 'programming in the large' Prolog has some serious weaknesses. Following McCabe (1992) we might mention the lack of a program structuring capability above the level of the predicate and the difficulty of capturing notions of state. Some Prolog implementations augment the language with module facilities to address the first of these problems. Arguably, however, a more principled remedy is to be found in the OOP concepts of encapsulation and inheritance. The introduction of these concepts into the language is what chiefly distinguishes Prolog++ from Prolog.

Encapsulation in Prolog++ is achieved by grouping between a pair of `open_object` and `close_object` declarations a collection of predicate and attribute definitions.[2] By the use of `private` declarations, some of an object's predicates are rendered inaccessible from outwith the object. Predicates that are not `private` provide the object's interface to other objects and define the forms of call that can be made to the object. Inevitably,

---

[2]Prolog++ is a product of Logic Programming Associates (LPA) Ltd. The version described here and used throughout the present project is release 1.0. Recently LPA have introduced version 2.0 which contains some changes, mostly (but not all) minor changes of syntax: for example `open_object` and `close_object` have been replaced by `class` and `end`.

Prolog++ literature mixes OOP terminology with that of LP and refers to the non-private predicates of an object as 'public methods', calls to which are 'messages' that are sent to the object. To illustrate the syntax, to send a message `time(T)` to a `clock` object we write

```
clock <- time(T).
```

Note the partition of the predicate name space: there could be other objects that have public `time/1` methods. Providing object names are distinct, name clashes between methods will not occur.

The values of an object's attributes, which can be understood in Prolog terms as assertions except that they can be accessed only locally (i.e. from within the object), define the object's current state.

Inheritance refers to the hierarchical relationships that can exist between objects. One object can be declared (by means of a special `super` attribute) to have another object as its parent. A message which is sent to an object but which cannot be handled by the object will be passed automatically to the object's parent (and thereafter possibly to the object's more distant ancestors). Inheritance plays an important role in allowing an existing object to be reused, even though some of its methods are not quite appropriate, since a child object can be defined which overrides the inappropriate methods with replacement definitions and adds new methods to implement the required special features. Taken together, encapsulation and inheritance augment Prolog with a powerful program structuring capability.

However, Prolog++ also has disadvantages. Prolog programs when rewritten in Prolog++ typically occupy more memory space and run more slowly.[3] The OOP features really justify substantial additional programming environment support but little is provided beyond the standard programming environment for Prolog. The increased conceptual complexity that comes when two programming paradigms are combined is not to be

---

[3]The standard 'naive reverse' example is claimed by the developers of Prolog++ to run at 97.6% of the speed of the Prolog version, but much worse degredations in performance are common with more realistic examples.

underestimated.[4]

Also, whereas Prolog is now a mature language with an ISO standard and a moderately large user base, Prolog++ is an evolving commercial product with limited support. It competes among a group of languages each representing a distinct attempt to unify LP with OOP; apart from Prolog++ other examples are L&O (McCabe 1992) and Parlog++ (Davison 1991). Nevertheless, for the purposes of the present project these problems were considered to be outweighed by the prospects that Prolog++ seemed to afford for an efficient and flexible software engineering process.

3.8.3 Preparation of Primex 2.0 and 2.5

The preparation of Extensible Primex began late in 1993. The version of Primex that was then current, viz. version 1.06, had been written originally in LPA MacProlog (Johns 1991) and was designed to run under Release 6 of the Macintosh Operating System. The system was now rewritten entirely in Prolog++ (which itself is implemented as an extension to MacProlog) running under Macintosh OS 7.0 and provided with the additional code that would enable the shell to act as the host for future extensions. As a feasibility test of the extensible architecture, a single small extension was developed which provided the facility to import database files into the shell.

At the same time the opportunity was taken to generally upgrade the shell. This seemed worthwhile especially since the knowledge acquisition tools would inherit the benefits. The upgrade was extensive, although not really fundamental, and included changes to the shell's rule-based Primex Knowledge Representation Language (the PKRL), user interface modifications, and improvements in performance. Appendix 3 lists the upgraded features. This version of Extensible Primex was released with the single implemented extension as Primex 2.0 early in 1994.

Unfortunately, Primex 2.0 did not last long. Apple Computer soon afterwards distributed a new version of its operating system. This version, OS 7.5, brought changes in memory management that

---

[4]The Prolog and Prolog++ documentation together comprises seven manuals.

had the effect of preventing Primex 2.0 from running. Since local schools were buying many new Macintosh computers at that time, and since these were all provided with OS 7.5, the problem could not be ignored. The remedy was initially expected to require only recompilation of Primex under an awaited new release of MacProlog. However the new MacProlog, when it arrived, contained extensive changes in language specification that in turn forced many changes to the Primex source code.[5] This version of Extensible Primex, which contained no change in specification, was released as Primex 2.5 late in 1994.

Other developers may wish to design Primex extensions. To do so requires that certain protocols be observed. These are described in Appendix 4.

3.9 Summary

The hypothesis that knowledge acquisition systems will enable better-modelling than has been possible with the EMYCIN-type shells has to be tested empirically. We have described a research strategy based on the design, implementation, and evaluation of knowledge acquisition tools. In developing the tools we have argued that the Persistent Collaboration Methodology, a hybrid of user-centred design with action research, ensures a proper balance between the 'technology push' and the 'learning pull'. The chapter has explained why classification tasks are an appropriate choice of application area. A strategy for implementation has been described that will provide the new tools as software extensions to an existing expert system shell.

---

[5]This is explained in technical note 1 of Appendix 6.

Chapter 4 Foundations for design

This chapter establishes foundations for the design of classification modelling tools. We discuss the place of classification in education and conclude that appropriate representations of classification for children's modelling may include decision trees, classification hierarchies, and factor tables. From the KBS literature we distinguish three inference methods used in classification and we explain why the methods of simple classification and systematic refinement are considered to be more suitable for educational tools than the method of heuristic classification. We survey some of the knowledge acquisition literature and identify a range of potentially useful techniques, including 'contrived' techniques such as laddering, knowledge editors, and induction. These considerations together lead to outline specifications for three tools, each of which is claimed to have the potential to be usable and useful. We explain why, although the three specifications are diverse, they should adhere to a set of overarching design principles relating to the user interface, compositionality, and runtime interpretation.

4.1 Representations for classification in education

The previous chapter concluded that classification is a suitable application area for building classroom modelling tools that use knowledge acquisition technology. Therefore we begin this chapter by examining the nature of classification in education. Our special interest lies in methods of representation, since these will be central to tool design.

Many teachers would have no difficulty in recognising a dictionary definition of classification as something like 'systematic placement in categories' (Collins English Dictionary 1994). As noted previously, teachers report that activities involving classification occur quite frequently in a range of school contexts. In fact education theorists have repeatedly stressed the connections between learning, classifying, and intelligent behaviour (for example Bruner et al (1956), Rosch (1975),

Klausmeier et al (1974), Homa (1984) and many others). Bruner wrote:

> *We map and give meaning to our world by relating classes of events rather than individual events. The moment an object is placed in a category, we have opened up a whole vista for 'going beyond' the category by virtue of the superordinate and causal relationships linking this category to others.* (Bruner et al 1956 p13)

Bruner's use of the words 'superordinate' and 'causal' indicate that his interest extended to class structures that are hierarchical and where the links denote not only class membership relations. This is one hint as to the kind of external representations — including diagrammatic and text notations — that might be useful to the design of knowledge-based tools. To enable children to build a classification knowledge base, we must somehow provide formal methods of external representation[1] that will permit the categories, the features of the categories, and the relationships between categories to be expressed.

Ideally, we should like to identify representations for classification that are already familiar to children, or at least that are regarded by teachers as being accessible to them. This reflects a well established guideline for good design: tools should provide direct access to the concepts that users associate with work in the domain (Norman 1986). Similarly, we note that Stelzner & Williams (1988) argue that the usability of a tool is related to the degree to which its interface matches the user's conceptual framework or 'natural idiom' for thinking about the problem. But do children have a 'natural idiom' for classification? If so, what is it?

It is relevant at this point to ask what representations for classification are recommended to and used by teachers. Table 4.1 shows the classification-oriented attainment targets that were

---

[1] For brevity, we will often refer to a formal method of external representation by one of the terms 'formalism', 'formal representation' or even just 'representation'. This introduces a slight risk that a *method* of representation will be confused with a specific *instance* of using the method, but generally the context will make the clear the intended meaning.

uncovered by our study of curriculum documents (Chapter 3, section 3.7.2). Although the attainment targets themselves mention no representations, the accompanying text does make such references. However the terms used are not always clear: for example there are references to 'classification keys', 'a hierarchy of classification', 'scientific classification systems' and 'simple tables'.

Report	*Strand*	*Level*[2]	*Attainment Target*
Environmental Studies	Science	C	Use simple apparatus, e.g. lenses, to collect information and use **classification** techniques to group unfamiliar things
Environmental Studies	Social Subjects	C	Propose simple **categories** within which information/evidence can be organised, eg types of shop, types of transport
Environmental Studies	Social Subjects	C	Observe artefacts, activities and events and identify and sequence or **classify** their main aspects, eg. the events in a local election; ways people spend money in a local supermarket; features of local buildings
Environmental Studies	Technology	C	**Match the properties** of familiar materials and equipment **to different purposes** and simple design tasks
Expressive Arts	Music	D	**Identify music in a variety of idioms**, eg Scottish traditional, folk, and jazz; distinguish the sound qualities of woodwind, brass, percussion and string sections
Mathematics	Shape, position and movement	A	**Classify** shapes by simple properties
Mathematics	Shape, position and movement	E	Define and **classify** quadrilaterals

Table 4.1 Classification-oriented attainment targets

[2]Levels A-E are intended to denote stages of educational development. The levels are not tightly linked to chronological age in recognition of the fact that different children develop differently. However the curriculum guidelines indicate that levels A, B, C, D, E should be attained during the years P1-P3, P3-P4, P4-P6, P5-P7, and P7-S2, respectively.

In order to gain more information we discussed classification informally with a number of teachers and inspected a range of teaching materials for the 5-14 stages. The results of this enquiry were useful but limited. 'Classification keys' transpire to be what in AI would be called decision trees. We found several explicit recommendations in teaching materials that decision trees should be constructed or used by pupils, including within two widely used mathematics schemes and in a BBC teacher education video on Primary Science.[3] Classification hierarchies are basically the class membership tree structures that are familiar to AI. We found some examples of these, but surprisingly few and mostly in 'home-made' teachers' materials. However, the teachers to whom we showed the example of Figure 4.1 all claimed to recognise this type of diagram and most thought that their pupils would recognise it too.

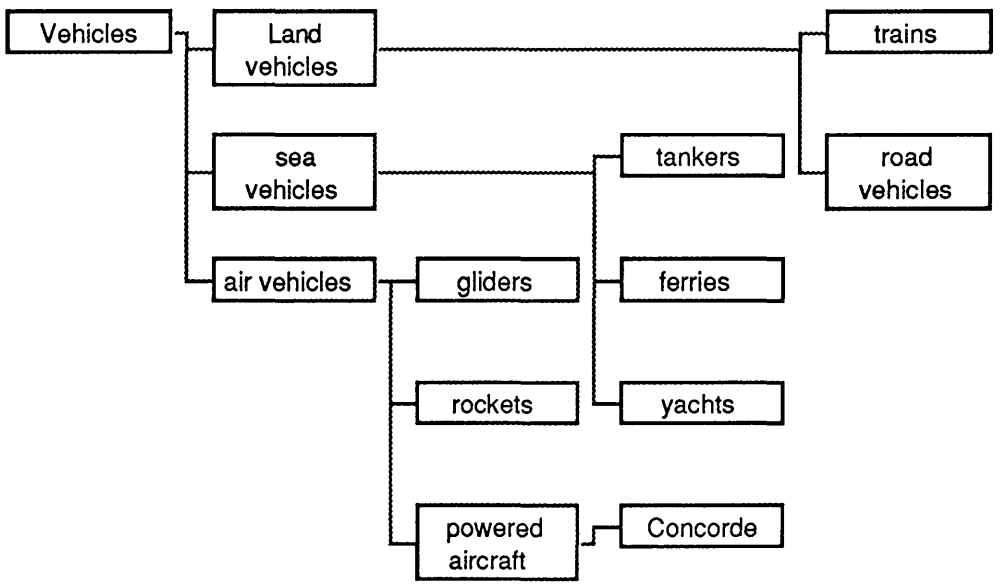


Figure 4.1 A classification hierarchy

Tables are much more common. They seem to be very widely used by teachers to organise and present information of many kinds, not just classification information. Classification-oriented tables vary in form: we noted two main types which we called 'lookup' tables and 'factor' tables. These are discussed later.

---

[3] *Cambridge Primary Mathematics* Module 7 Teacher's Resource Book p27 and pp74-77; *SPMG Heinemann Mathematics* Vol 3 (Teacher's Notes p180) and Vol 4 (Teacher's Notes p271); BBC Video *Teaching Today* Series Primary Science (Classification).

Among other representations we identified examples of Venn diagrams, Carroll diagrams, flowcharts, concept maps, and cartesian graphs. However, these were mostly used for purposes other than classification.

The Cognitive Science and AIED literature on external representations contains some helpful findings. The extent to which children understand class inclusion hierarchies has been investigated by Greene (1989). He reports that multilevel class inclusion is relatively well understood by children at the age of 7 or 8 (slightly younger than predicted by Piagetian theory). Children of this age or older can construct (paper and pencil) tree diagrams 'with minimal instruction' and he concludes that classification trees are 'good descriptive models for how individuals may represent information that is hierarchical in nature' (p87). However Petre (1993) emphasises that to use effectively even well-evolved graphical notations requires skill that, generally, is acquired only by training and experience. Cox & Brna (1993) stress the importance of student's prior knowledge of external representations (ERs) as a predictor of their performance in reasoning and emphasise the value of representations that are actively constructed by, rather than given to, the student. However, the field is young:

> *The question "What constitutes a 'good' ER?" requires further research. There is much folk wisdom and speculation but little empirical work on the issue of representation selection, construction and their use.* (Cox & Brna 1993 p33).

What can be concluded from all this? Some evidence exists that education professionals regard decision trees, classification hierarchies, and tables as formalisms appropriate for children. But we cannot be sure that children at (say) the middle secondary school stage will be familiar with these representations. Nor can we consider them to be 'natural idioms', although Greene's research is encouraging for the case of classification trees. The impression is of a somewhat casual use by teachers of a variety of notations; and of a field which has so far been only lightly investigated by researchers. In view of the weight attached to classification by figures with the authority of Bruner, we found

this a little surprising. However it does add interest to our research question concerning children's learning of formalisms through the use of knowledge-based tools (Chapter 3, question 10 in Table 3.1).

4.2 Classification inference

Within the field of KBS, classification-based methods of problem-solving are among the most widely used and best understood. A particular stimulus to research came from the sometimes bewildering array of techniques and claims that were made for the expert systems of the late 1970s and early 1980s. In an attempt to characterise the 'knowledge-level competence' of some of these programs, Clancey (1985) identified a pattern in their method of reasoning. He abstracted the pattern thus:

> *These programs proceed through easily identifiable phases of data abstraction, heuristic mapping onto a hierarchy of pre-enumerated solutions, and refinement within this category. In short, what these programs do is commonly called* classification, *but with the important twist of* relating concepts in different classification hierarchies by non-hierarchical, uncertain inferences. *We call this combination of reasoning* heuristic classification. (Clancey 1985 p290, his emphases).

Clancey provides helpful diagrammatic representations for heuristic classification. Some of these diagrams are reproduced in Appendix 5.

Heuristic classification as a problem-solving method is today well established. However, simpler methods of classification exist. In the library of KADS Generic Task Models compiled by Tansley and Hayball (1993), entries appear for three methods of classification. In all three methods, the set of possible solutions (category or class names) is completely enumerated in advance — regarded in KBS as a fundamental characteristic of classification. In summary, the three methods are:

(i) Simple classification: the features of some solution class appear directly in, or can be abstracted from, the input data.

Forms of abstraction include definitional abstraction (inferring necessary features of a concept), qualitatitive abstraction (summarising quantities with qualitative measures) and generalisation (selecting an ancestor node in a class membership hierarchy).

(ii) Heuristic classification: as per Clancey's description (above). The solution classes are hierarchically structured. Features can be abstracted from the input data that can be matched heuristically to a class within the solution hierarchy. The hierarchy is then refined using other features of the input data to select the solution.

(iii) Systematic refinement: the solution classes are hieararchically structured. The input data includes features that appear as the features of some class within the solution hierarchy. This hierarchy is then refined using other features of the input data to select the solution.

It can be seen that heuristic classification consists of a heuristic match sandwiched between a simple classification and a systematic refinement. The question that needs to be asked for the present research concerns the scope of the tools to be developed. Specifically, should we aim to support all three methods?

The attraction of the full-blown heuristic classification method lies in its problem-solving power. As Clancey shows, the method is adequate to characterise many celebrated expert systems. Furthermore, the method may not be too difficult to be understood by at least some school pupils: it can be expressed in informal terms as something like 'decide on the important features of the case, use these to select a plausible rough solution, and polish that up into something that fits better'. A tool with which children could learn and apply heuristic classification might be valuable, just as Kolodner's tools and techniques for teaching case-based reasoning have been (Kolodner 1995).

On the other hand, to expect that children might build models that make explicit use of the heuristic classification

method may be asking too much. Certainly such models would be vastly more sophisticated than the rule-based models which children are building now. Considering the survey of schools' modelling discussed in Chapter 2, it seems quite likely that an improvement over the status quo could be achieved even with the simpler classification inference methods. The simpler methods might suffice for many if not all of the models that schools are likely to build in the near future.

Morever, there is reason to doubt the feasibility of building a modelling tool that gives explicit support to heuristic classification in a domain-independent way, whilst being usable to people other than knowledge engineers. For instance, Clancey's own HERACLES (Heuristic Classification Shell) program combines rules, frames and metarules (Clancey 1987). Advanced programming skills are certainly assumed of HERACLES users, who are encouraged to use rules for performing abstraction and heuristic association, frames for representing hierarchies, and metarules for defining the inference strategy.[4] The complexity of HERACLES is a fair indication of how ambitious would be any attempt to automate in a reasonably general way the acquisition of the knowledge that is needed by the heuristic classification method.

We conclude that for the present research, to give explicit support to the full-blown method of heuristic classification is probably neither necessary nor feasible. Simple classification and systematic refinement are simpler and more appropriate methods; ideally we should like to support them both.

4.3 Knowledge acquisition techniques

As noted in Chapter 2 (section 2.8), the evidence from surveys by Cooke (1994) and Boose (1989) indicates that existing domain-independent knowledge acquisition tools often make use of one of the 'contrived' knowledge elicitation techniques:

---

[4]Even with this combination Clancey expresses dissatisfaction, arguing that 'the best system for classification we could imagine' would be a combination of HERACLES with two other toolkits, viz. KL-ONE and SHRINK (Clancey 1985 p337).

laddering, repertory grid, and card sort are among the most common. These techniques are all candidates for our consideration. Another candidate is rule induction, probably the most widely-used machine learning technique in KBS, whereby the system constructs a decision-making procedure by analysing a set of existing examples or case records. We will also consider knowledge editors — usually graphical or form-based editors that are embedded with assumptions about the characteristics of the information that is to be edited. All of these techniques have been used successfully to build knowledge acquisition tools for research, industry, and commerce.

For each technique we now consider the implications of using the technique for the design of a tool, with the general aim of estimating how usable the technique would be by children in building classification models.

4.3.1 Contrived techniques

Contrived techniques are 'active' knowledge elicitation techniques which present the user with a series of questions designed to gain information about the domain. As noted in Chapter 2, the studies by Burton et al (1988a, 1988b) showed that in classification domains, manual versions of these techniques can be very effective in eliciting knowledge from experts. Different techniques are associated with different types of questions but within each technique questions are rather simple and limited in variety, making the techniques quite suitable for automation. The techniques have numerous variations but their general nature is shown in Table 4.2 along with the names of some existing tools that implement the techniques.

The techniques differ also by the representations that they construct: laddering builds a class hierarchy, repertory grid builds a table showing the extent to which each class possesses each attribute (attributes here are dichotomous), and card sort builds a table showing class names, attributes and values. Tools generally make these representations graphically visible, and possibly editable, either during or on completion of the knowledge elicitation process. In many cases the tools contain compilers that

can generate rules from these representations. The rules can be run directly, typically using inference engines based on simple classification or systematic refinement methods.

Technique	*Typical questions/prompts*	*Tools/References*
Laddering	Can you give examples of X? What is Y an example of? How does X differ from Y?	ALTO (Major & Reichgelt 1990) TOPKAT (Kingston 1995)
Repertory grid/triadic elicitation	What do X and Y have in common that differentiates them from Z? What differentiates X from Y?	KSSO (CPCS 1991) TOPKAT (Kingston 1995)
Card sort	Sort these cards into categories of your own choosing	TOPKAT (Kingston 1995)

Table 4.2 Typical questions generated by contrived techniques

It might be expected that use of the contrived techniques will potentially change the character of model building. Normally, model building is associated with a student role which is active and dominant with respect to the role of the system. With the contrived techniques on the other hand, insofar as the system drives the process, the student's role will be more passive.

The full implications are not clear, however, as we illustrate now. Consider two hypothetical scenarios involving contrived techniques. In the first scenario, system-generated questions are comprehensible and appropriately varied. They sustain children's interest, have evident purpose and coherence, and stimulate reflection and discussion. As information is gained the system updates its visible representation of the model in a transparent way. When children wish to change their answers, or express an idea that seems to them important but which is not presently a target of the system's questioning, they can take the initiative either by editing the visible model directly or by restarting the dialogue at a point of their own choosing. In the second scenario, the system's questions are often unintelligible, highly repetitive, lacking in direction, or some combination of all three. There is no visible model to show how the information integrates within an overall structure. When a child enters an answer that he or she

later realises is mistaken there is no way to correct the error, save by starting again from scratch.

These scenarios are extreme, but they have some basis in existing research. Contrived-technique dialogues can be tedious, as Kathleen King (1995, personal communication) confirmed and as Burton et al (1988b) discovered when they encountered reluctance among real-world experts to participate in knowledge elicitation sessions with these techniques. That system-generated questions can be opaque is suggested by the following examples, quoted by Kingston (1995), that have been generated by the card sort tool of his TOPKAT system:[5]

Small is the/an WHAT of Hamster?
Can the/an Zoo collection of Hamster exist even if (the/an) Hamster does not exist?
Is Red an instance of Colour, or a predicate describing the value of Colour?

In fact, many KBS tools for knowledge acquisition are designed on the assumption that the system will drive the dialogue. Users are not expected to take the initiative. However, this assumption may be inappropriate even for non-educational contexts. Major & Reichgelt (1990) report that some users — who were experienced knowledge engineers — of their prototype ALTO laddering tool disliked a purely system-driven style of interaction. ALTO was redesigned to enable the user to take more control.

What can we conclude from this? A main aim of this research is to provide learners with more support in the model building process. The contrived techniques provide one means of giving that support, possibly all the way from elicitation of categories and features through to the generation of code that can be run to perform classification tasks. Thus they merit investigation. However, it is likely that the character of model building will change; possibly but not necessarily for the worse. In the case of the second scenario above, the support seems more like a straitjacket: it is doubtful that it can be called modelling at all, in the sense that modelling means directly manipulating and

---

[5] This is certainly not intended as a criticism of TOPKAT, which is designed not for children but as a support tool for knowledge engineers using the CommonKADS methodology.

refining a glass-box structure. Much may depend on the success of the dialogue design and on the opportunities that are available for the student to take the initiative.[6]

4.3.2 Induction

Induction in KBS usually means the acquisition of rules from data by the use of ID3 (Iterative Dichotomiser 3; Quinlan 1979) or by some related machine learning technique. In a typical induction-based knowledge acquisition tool, such as ExpertEase (Crabb 1985), knowledge is presented to the system in the form of a table like the one shown in Table 4.3. Asterisks represent wildcard or 'don't-care' values. From this table the induction technique would generate a decision tree such as the one shown in Figure 4.2.

Preferred playing style	Fitness	Likes contact sports	Suggestion
*	good	*	Squash, swimming or weights
Team	good	yes	rugby, shinty or hockey
Team	good	no	volleyball
Solo	good	*	track racing
*	poor	*	Darts or chess
*	poor	*	Rambling, golf, or bowls

Table 4.3 Example of input to an induction tool

Importantly, ID3-based techniques are generally able to derive the 'best' (e.g. fewest-nodes) tree for the input data, although since a heuristic is used to select the optimal attribute at each node this is not guaranteed. The decision tree may be displayed graphically or just held as an internal data structure. Generation of rules from the tree, and hence runnable code, is straightforward. A possible problem, commented on by Bramer

[6]By way of an aside, it is interesting to note the similarities between these considerations and those reported from research into 'learning-by-teaching' systems, which cast the student into the role of tutor with a computer tutee. That research likewise identifies dialogue design and locus of control as problematic issues. At present, the state of learning-by-teaching research is described by Nichols (1994) as 'embryonic' but in the future it should be possible for that field and the field of knowledge-based modelling to interact fruitfully.

(1987), is that a tree which is optimal in a classifactory sense may not prioritise decision factors in a way that makes sense to a human who is knowledgeable in the domain.

Unlike the contrived techniques, induction of itself provides no help with knowledge elicitation. The user must provide data in a suitable form. If that can be done then an induction tool should be able to produce efficient runnable code. For building models of simple classification in an education context induction may be a useful technique, but research is needed to decide how much support at the knowledge elicitation stage is needed and whether Bramer's problem is significant in the context of educational modelling.

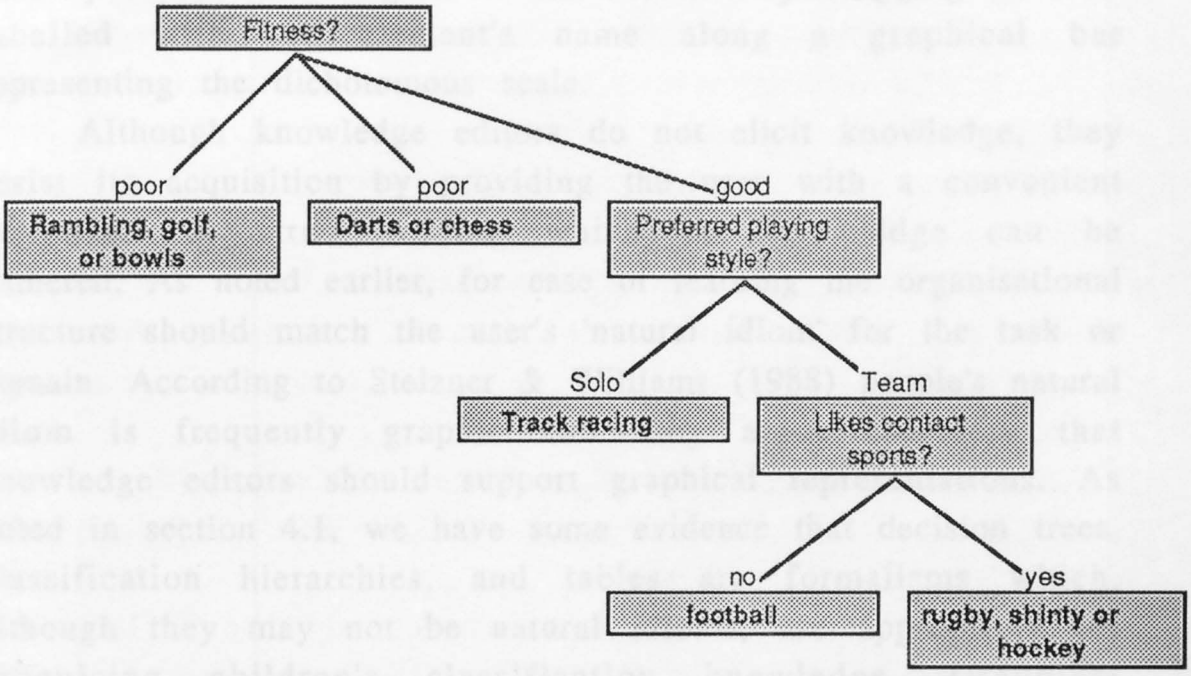


Figure 4.2 Decision tree generated by induction from Table 4.3.

4.3.3 Knowledge editors

Many knowledge acquisition tools provide special-purpose knowledge editors which reflect the structure of a domain or task. For example the Mahogany Help Desk, a KBS tool for constructing help-desk systems, provides a decision tree graphical editor which is said to fit well with the essentially diagnostic task structure of the knowledge it is designed to acquire (Emerald Intelligence 1992). Another diagnostic system providing a decision tree editor is KLUE (Karel & Kenner 1989) which according to Price (1990)

can be used directly by domain experts who can 'build simple systems ... with very little input from trained knowledge engineers'.

A review by Birmingham & Klinker (1993) of task-oriented knowledge acquisition tools indicates that these systems, for example SALT (Marcus 1988), BURN (Marques et al 1988), and OPAL (Musen et al 1988) often use graphical form-filling editors. In general, graphical editors that use direct manipulation techniques are becoming more common. For example in the card sort tool of TOPKAT (Kingston 1995) cards are represented graphically and the user sorts them into categories by direct manipulation. In the repertory grid tool KSS0 (CPCS 1991), the user specifies the rating of a new element by dragging an icon labelled with the element's name along a graphical bar representing the dichotomous scale.

Although knowledge editors do not elicit knowledge, they assist its acquisition by providing the user with a convenient organisational structure into which the knowledge can be gathered. As noted earlier, for ease of learning the organisational structure should match the user's 'natural idiom' for the task or domain. According to Stelzner & Williams (1988) people's natural idiom is frequently graphic and they argue therefore that knowledge editors should support graphical representations. As noted in section 4.1, we have some evidence that decision trees, classification hierarchies, and tables are formalisms which, although they may not be natural idioms, are appropriate for organising children's classification knowledge. Graphical knowledge editors for these formalisms may therefore provide useful support in constructing a classification model.

4.4 Outline specifications for tools

The previous sections have identified considerations which should influence the design of classification-oriented knowledge-based modelling tools. We summarise these considerations below. We then propose outline specifications for three new modelling tools.

(i) Children may be familiar with decision trees, classification hierarchies, and tables as representations for classification and there is evidence that these formalisms are accessible to children (section 4.1).

(ii) Three classification methods of problem-solving can be distinguished: simple classification, systematic refinement and heuristic classification. In this research, the first two methods may be pursued but it is probably not feasible or necessary to aim to give explicit support to heuristic classification (section 4.2).

(iii) Knowledge acquisition techniques that could provide support to children's model building include the contrived knowledge elicitation techniques (laddering, repertory grid, and card sort), induction, and knowledge editors (section 4.3).

(iv) Of the contrived techniques, laddering generates a classification hierarchy and so would be suitable for a runtime interpretation based on systematic refinement; repertory grid and card sort generate flat class structures suitable for simple clasification (section 4.3.1).

(v) Induction can generate efficient runnable code from a table of data (section 4.3.2).

(vi) Knowledge editors can help to make tools usable. Graphical knowledge editors for decision trees, classification hierarchies, and tables may provide useful support in constructing a classification model (section 4.3.3).

We claim that these considerations justify the outline specifications shown in Table 4.4. It is argued that each specification has the potential to realise a usable and useful modelling tool because it utilises an appropriate representation, supports model development with proven knowledge acquisition techniques, and addresses a task (viz., a form of classification) that we have shown to be educationally relevant.

	Specification 1	*Specification 2*	*Specification 3*
Tool name	PDT (Primex Decision Tree)	PFT (Primex Factor Table)	PCT (Primex Classification Tree)
User-level representation	Decision tree	Factor table	Classification tree
Knowledge acquisition techniques	Graphical knowledge editor	Graphical knowledge editor with induction	Graphical knowledge editor with laddering
Inference method	Simple classification	Simple classification	Systematic refinement

Table 4.4 Outline specifications for tools

It is not being claimed that these are the only possible such specifications. On the contrary, this research appears to have uncovered an embarrasment of riches in that we leave unexplored many of the identified possibilities. Potentially, however, the table contains enough variety to generate interesting comparisons. We recognise also that graphical tools are often difficult and laborious to implement and, recalling the discussion on the research questions relating to design and implementation (Chapter 3, section 3.2), it seemed unwise to plan more implementations than the three shown here.

A few words about the individual specifications may be helpful. For all tools, it is intended that the user-level representation will be graphically editable. Examples of each representation were given above (see Figure 4.2, Table 4.3 and Figure 4.1 for PDT, PFT and PCT respectively). As noted earlier, a factor table is one of two types of tabular representation that were identified in school teaching materials. The other type, which we call lookup tables, is illustrated by Table 4.5 in which travel methods are classified according to the values of two attributes, viz. distance and cost. Lookup tables are limited to such classifications involving only two attributes. Factor tables can accomodate any number of attributes and it was considered that this extra expressive power justified their preference. The classification tree tool will provide two alternative methods of entering knowledge, viz. laddering and direct graphical editing. This will permit the issues to be explored which were raised earlier about the locus of control within contrived technique tools.

	Short distance	*Medium distance*	*Long distance*
Low cost	bicycle	bus	bus
High cost	car	train	aeroplane

Table 4.5 A lookup table for classifying methods of travel

4.5 Design principles

As Chapter 3 explained, the new tools were envisaged as related software packages. Their construction shares a common purpose: to test the hypothesis that knowledge acquisition systems enable more successful model building by children than has been possible with the EMYCIN-type shells. The tools are to be evaluated side-by-side in a set of large-scale school trials. These factors influenced the selection of a common implementation strategy, whereby the tools were produced in MacProlog/Prolog++ as software extensions to the existing Primex shell.

As we now explain, the relationships between the new tools also helped to shape a set of overarching design principles relating to the user interface, compositionality, and runtime interpretation.

4.5.1 Family resemblance

The first design principle concerns the user interface. Section 3.8 argued that in order to achieve a feasible and valid evaluation of tools, not only should each tool adopt internally consistent user interface protocols but also there should be a strong family resemblance between different tools.

The new tools PCT, PFT and PDT are all embedded into the Primex environment. The File menu New command acts as before to create a window for a rule-based model (i.e. a PKRL model). Windows for the other kinds of model are created via a new File menu Extensions submenu. Windows for models in PCT, PFT and PDT share the common features shown in Figure 4.3.

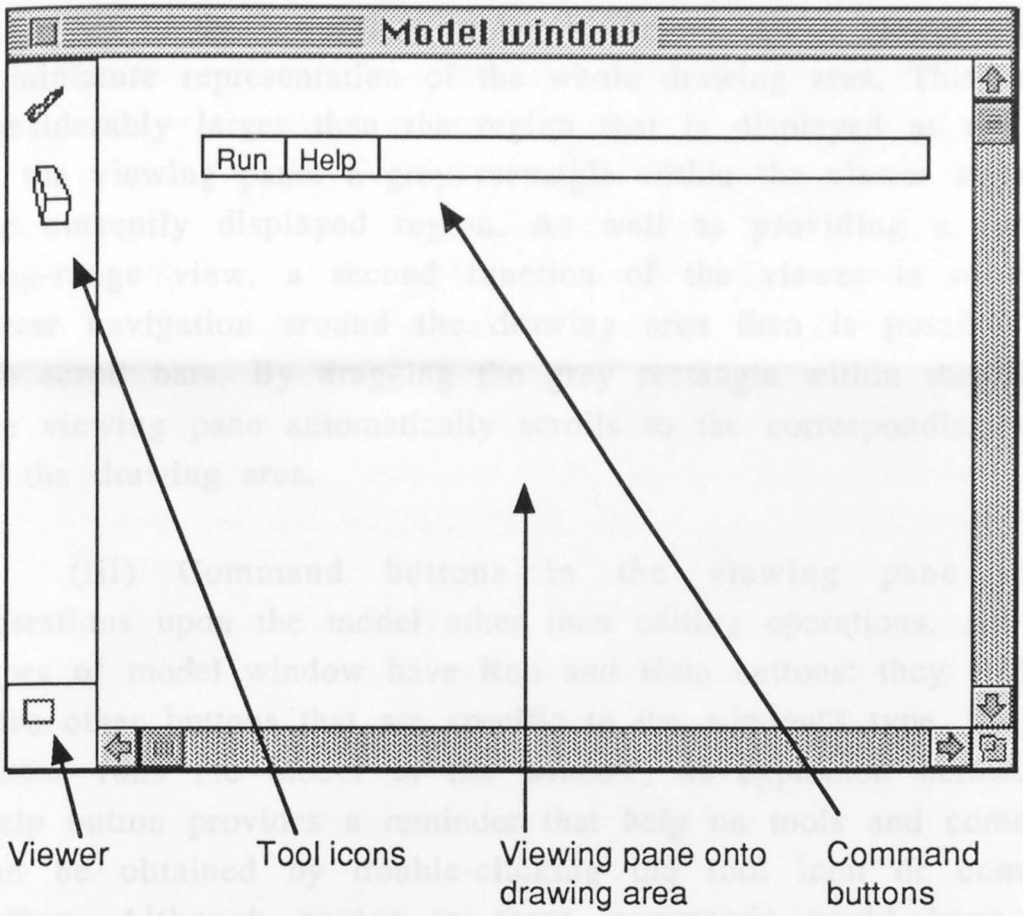


Figure 4.3 Common features of model window layout

Apart from having familiar features of the Macintosh user interface, such as close boxes, resize boxes, scroll buttons, and scroll bars, model windows for these tools contain the following standard elements:

(i) In the left edge of the window, a tool pane contains icons representing direct manipulation tools that are used to construct and edit models in the viewing pane. Icons shown in Figure 4.3 are the pen and eraser, both of which appear in all three modelling tools: other icons are specific to individual model types. A tool is activated by selecting its icon. When the mouse pointer is placed anywhere in the window's viewing pane, its appearance changes to a specialised cursor that gives visual feedback on the currently selected tool. Double clicking on a tool icon or command button (see below) reveals help information on the tool or command.

(ii) In the lower left corner of the window a viewer displays a miniature representation of the whole drawing area. This area is considerably larger than the region that is displayed at any time in the viewing pane: a grey rectangle within the viewer represents the currently displayed region. As well as providing a form of long-range view, a second function of the viewer is to enable faster navigation around the drawing area than is possible with the scroll bars. By dragging the grey rectangle within the viewer, the viewing pane automatically scrolls to the corresponding region of the drawing area.

(iii) Command buttons in the viewing pane invoke operations upon the model other than editing operations. All three types of model window have Run and Help buttons: they will also have other buttons that are specific to the window's type. The Run button runs the model in the window, as explained below. The Help button provides a reminder that help on tools and commands can be obtained by double-clicking the tool icon or command button. Although access to these commands could have been provided via tool icons, we wanted to reflect consistently the notion that a tool icon requires the selection of part of the model, i.e. requires pointing with the mouse.

Most regular Primex commands, such as the File menu's Open, Save as, Close and Print commands, can be applied to all kinds of model window.[7] The common window features and the use of regular Primex commands should ensure that a proportion of the tool-using skills learned in the context of one modelling tool will transfer to any other. To further assist in learning the new tools, a set of user help sheets was produced which emphasise the family resemblance. These help sheets are reproduced in in Appendix 7.

Of course, other interface designs can be envisaged which might achieve the required family resemblance between the new modelling tools. The design shown here is not perfect; for example

[7] A notable exception is the Undo command. A large amount of implementation effort would have been needed to generalise this command across all tools and so this was not done. With hindsight, it is probable that the effort would been justified by the resulting gain in usability.

the command buttons can be scrolled out of view, causing possible disorientation. It was adopted, however, for three main reasons. First, the researcher had previously used a similar design with some success in his PathFinder and PhraseMaker systems, which were similarly aimed at children (Conlon 1993, 1994). In that earlier work the success had been partly attributed to the design's broad similarity to the interface provided by the Claris Works drawing package (Claris Corporation 1994) which is used fairly widely in schools. Second, early feedback from teachers, children and college students indicated that the problems present in the design were not too serious. Third, the design is well matched to the capabilities of the MacProlog graphics package. This means that the implementation effort is less than would be required for some other designs and it enhances the prospects for experimental adaptation.

4.5.2 Coarse-grained compositionality

The second design principle concerns the way in which models may be developed. Miller et al (1991) insist that a modelling system must provide primitives or 'building blocks' that are of the right size or 'granularity' to match the modelling task. They give an analogy from word processing. The familiar building blocks of word processing are individual characters: this is reflected in the design of standard keyboards, which access one character per keystroke. However, differently designed keyboards can be envisaged. One possibility is for each keystroke to create a segment of a character (e.g. an upward arc). Another possibility is for each keystroke to create a sequence of two or three characters (e.g. 'ing' or 'sc'). These non-standard keyboards are unattractive, however, because they would generally increase the effort required to create text. For the task of creating text, the first keyboard would provide building blocks that are too fine-grained and the second keyboard would provide building blocks that are too coarse-grained.

Although Miller's analogy is helpful, we argue that it should be elaborated to recognise that granularity is partly a function of task. Not all tasks in word processing suit character-level building

blocks. For example, the proper building block for document layout may be the page; for text alignment, the paragraph; and for font editing, the pixel. A good word-processing package reflects this by providing commands which operate upon pages, paragraphs, and pixels in addition to commands which operate upon characters.

Arguably, modelling tools should similarly provide variously sized building blocks. For example, a factor table can be viewed as an array of cells, a sequence of rows, or a sequence of columns. A cell-level view is the least abstract but represents the proper focus when one value is to be entered or edited. A row-level view identifies a single classification and is perhaps the most helpful building-block for designing a single whole table. A column-level view isolates a single attribute. When the task is to extend the model's sophistication, this view may be appropriate. The modelling tool should reflect the different kinds of building-block by providing commands that operate at the single-cell, whole-row and whole-column levels.

We claim, however, that in some contexts these building blocks are *all* too fine-grained. A very large factor table model which can only be viewed as a homogeneous collection of cells, columns or rows may be unmanageable. Just as a large word processed document may be more manageable when implemented as a linked set of smaller documents, so a large factor table may be better realised as a set of linked smaller factor tables. In other words, the proper building block for a large factor table model may be the factor table itself. Similarly, it should be possible to compose a large decision tree model from a linked collection of smaller decision trees, or a large classification tree model from a linked collection of smaller classification trees. We show how this more coarse-grained compositionality can be achieved later in this chapter.

One justification for this proposal is by analogy with conventional software engineering. The advantages of developing software systems by some form of modular decomposition have been long recognised (e.g. Sommerville 1982). In general, modular subsystems can be developed and tested independently, if necessary by different programmers. A module can be re-used in

different contexts. A system can be complex yet remain manageable. Debugging and future maintenance is simplified. As our discussion of Prolog++ in Chapter 3 indicated, programming systems have evolved to develop support (e.g. modules, objects, packages, classes) for 'programming in the large' in addition to supporting 'programming in the small'. A modelling tool which provides some coarse-grained building blocks should similarly assist children to build bigger, better models.

Our proposal for coarse-grained compositionality also has a second purpose: it is hoped by this means to provide support for classroom collaborative work. Collaborative work has been shown to be effective in contributing to learning under certain conditions. The underlying mechanisms are not well understood but the opportunities that collaboration provides for peer interaction, especially explanation, are believed to be important (Bielaczyc 1993). Recent research suggests that successful collaborative work with computers can be facilitated by software which provides pupils with the opportunity to express their ideas, and by a design of learning activities which gives a strong structure for collaboration (Repman 1993, Hoyles et al 1994). Of course, an aim of this research is to develop expressive modelling tools: we believe that a strong structure for collaboration could be achieved as follows. We envisage a scenario in which a group of children will use our tools to design collectively an abstract model; say, a 'top-level' factor table which makes reference to lower-level tables which, initially, do not exist. Subsequently the lower-level models can be developed in parallel by individuals or subgroups, possibly working on different computers, then finally combined and tested on one machine. Whether this scenario is realistic should be established by future research.

Although we adopt coarse-grained compositionality as a design principle, our implementation of the principle will be quite limited. For example, the linking of models will not be animated graphically. Although it could be helpful for example to allow two linked decision trees to be automatically redrawn as one larger tree, and vice-versa, to achieve this would require a significant implementation effort. Also, only models of the same kind will be permitted to be linked. It would be attractive to allow composite

models of different kinds, giving extra flexibility in model design; and even more generally, it would be interesting to enable a model of one kind to be transformed automatically into a model of another kind, giving multiple viewpoints (Cheng 1993). However these must remain possibilities for investigation in future research.

4.5.3 Three-stage compilation

The third design principle concerns the way in which models will be interpreted. In our framework, to construct a model is to specify a classification scheme: to run a model is to use the scheme to perform a classification task. In effect, the model becomes an expert system which can classify objects in the model's domain. The system's runtime behaviour will be to query the user on the values of the attributes of some category, and then to identify the category or categories that have those values. From the teacher's and learner's point of view, the point of running a model is to enable its contents to be tested and demonstrated. Running a model may also act as a stimulus to reflection, discussion and further development.

A basic compilation scheme for models is as follows. At any time the information content of a model is specified by its diagram or diagrams.[8] Since the modelling tools are embedded within the Primex shell, it is natural initially to compile this source representation into the standard Primex rule language (i.e. the PKRL). The shell's internal PKRL compiler can then be invoked to generate runnable code that can be interpreted by the regular Primex inference engine. This scheme is feasible because the PKRL is a fairly expressive first-order logic-based language and the Primex inference engine is a reasonably efficient backward-chaining PKRL interpreter supporting a conventional KBS query-the-user runtime mechanism.

---

[8] This is true even when the model has been constructed indirectly, as in the case of a classification tree that has been elicited by the laddering technique, although the diagrams for classification tree models must be understood to include a 'hidden layer' of feature information. This is described in more detail later.

On reflection, however, we elaborated this method somewhat. Figure 4.4 shows the stages that are actually involved in compiling a PDT, PFT or PCT model. Instead of a single-stage compilation from the diagrammatic representation to PKRL, we have introduced an intermediate representation which we call 'normal form'. One advantage of this is that the modelling tools become decoupled from the shell, so that when a new modelling tool is developed only a new Model Diagram Compiler is required: the Normal Form Compiler can be re-used. Similarly, if in future the modelling tools are to be ported to a shell other than Primex then only the Normal Form Compiler needs revision.

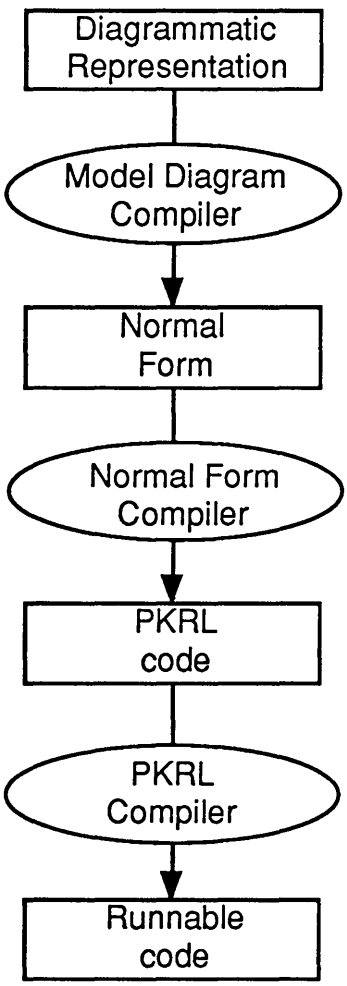


Figure 4.4 Transforming a model into runnable code

A second advantage of the three-stage compilation scheme is that normal form makes a convenient standard representation for PDT, PFT and PCT models. Normal form specifies the information content of a model as a list of Prolog terms each of the form

```
rule(Category, Features)
```

— where `Category` is an atom and `Features` is a list of pairs each of the form `Attribute=Value`. This very simple language is adequately powerful for the purpose of running a model in the manner described above and it also provides a convenient representation for model analysis, as we shall explain in the next chapter. The normal form compiler which maps the language onto PKRL has the important property that one PKRL rule is generated per normal form rule, with sequence preserved, so that (for a given shell) the runtime behaviour of a model is determined by the normal form. Implementation of the normal form compiler is straightforward.

Two of the three generated representations are purely internal. The exception is the PKRL representation, which is written to a window (with the title 'Translated KRL') that can be made visible. It was decided to make this representation available because some teachers felt that children who had made use of the Primex rule language would like to study the correspondence between model diagrams and the contents of the 'Translated KRL' window, which they might even choose to edit.

Of course, the three-stage compilation process is largely invisible to the model builder. When he or she clicks the Run button in a model window (see Figure 4.3 or the help sheets in Appendix 7) a slight pause follows and then the system begins to query the values of attributes.

Some limits should be noted. Each model window is restricted to contain a single model diagram. Since diagrams for the same kind of model can be linked, as explained above, this is not much of a restriction. At any time the user is permitted to have any number of model windows open, subject to memory limits, and these can include windows of different model kinds. When the user clicks the Run button of the front model window, that window's diagram and all of the diagrams that are referred to by the diagram are compiled and run. However, running a model does not cause any visible changes to or 'animate' the model diagram(s). In the researcher's previous PathFinder and

PhraseMaker systems (Conlon 1993, 1994) such animation was provided and it was successful in giving feedback on the interpretation process. In the present research the possibility of diagram animation was investigated but rejected, mainly because the increased complexity of model compilation and interpretation could not be justified by the focus of our research questions.

4.6 Summary

Since we advocate representations for classification that are in use in the curriculum, our research leads us to favour representations with decision trees, classification hierarchies, and factor tables. The inference method of heuristic classification, although powerful, appears to be too complex for our purposes and so we stick with simpler methods. Knowledge acquisition techniques offer an embarrassment of riches and most of what is available appears to have been little exploited by developers in education. From the combined possibilities we could probably have constructed many plausible specifications; this chapter proposed only three, but with at least enough variety to generate potentially interesting comparisons, both between themselves and against the standard rule-based shell. We have further argued that, if such comparisons are to be feasible and valid, tools should adhere to a set of overarching design principles. Armed with the outline specifications and with design principles relating to the user interface, compositionality, and runtime interpretation, we can now proceed to the next chapter which explains in more detail the design of individual tools.

Chapter 5 The modelling tools

This chapter describes three new classification-oriented knowledge acquisition tools, all implemented as extensions to the Primex shell. The tools are: PDT, based on decision tree representations; PFT, based on factor table representations; and PCT, based on classification tree representations. We explain how each tool manifests the overarching design principles that were discussed previously. We describe some of the main design issues that were encountered in the development of the tools and we explain how these issues were addressed. Whilst avoiding low-level detail, we aim to give enough information to enable our designs (or parts of them) to be understood and even replicated if necessary, so that the context of the experimental results which appear later in the thesis will be firmly established.

5.1 PDT: the decision tree tool

Figure 5.1 shows a window containing a small model created by PDT, the decision tree tool. In this section we explain how PDT manifests the design principles that were discussed in the previous chapter. We consider in turn design issues relating to model editing, linking, and interpretation. Although our focus here is on the outcomes of design and their justification, rather than the design process itself, an illustration of the way in which the Persistent Collaboration Methodology was applied in the development of this tool may be of interest. This is provided in Appendix 8. It may be helpful also to refer to the PDT help sheet in Appendix 7.

5.1.1 Constructing and editing decision trees

Key design issues for this tool relate to the assembly of nodes and arcs, maintaining the tree's shape, and distinguishing between question and decision labels. These issues are discussed below.

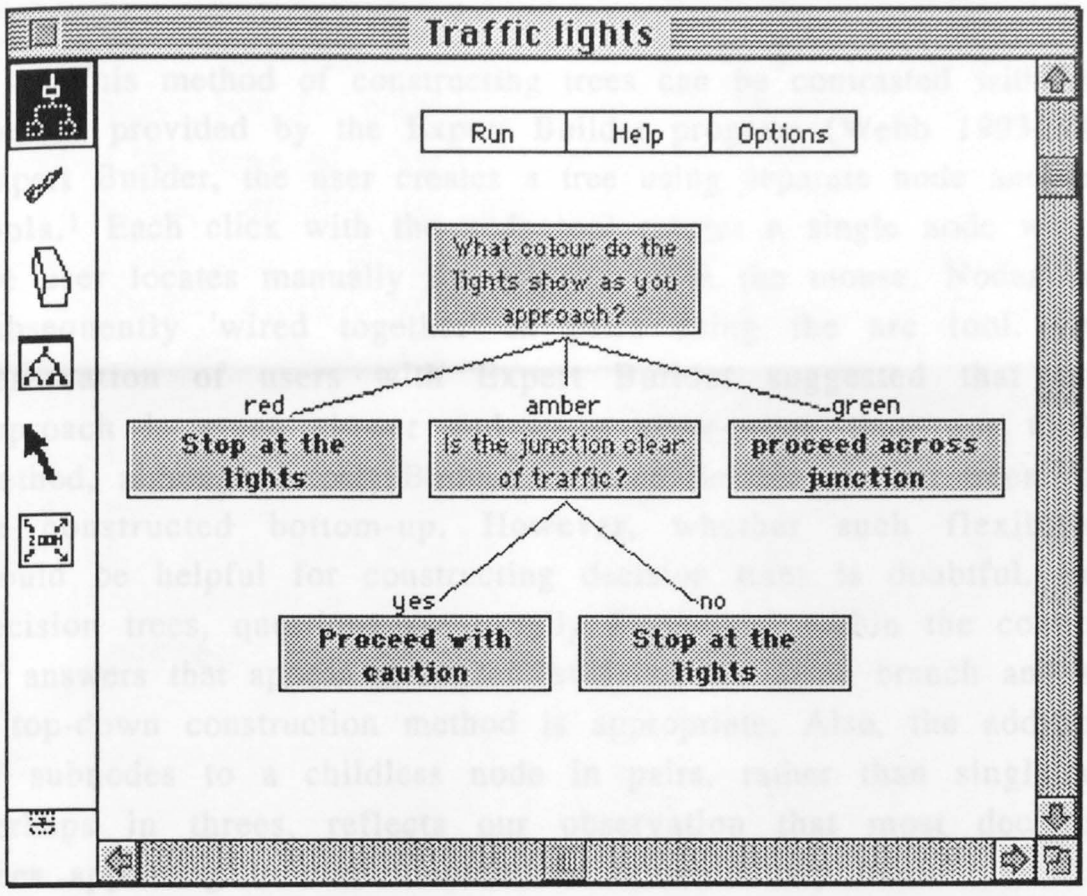


Figure 5.1 A decision tree window with model

5.1.1.1 Assembly of nodes and arcs

In PDT, a new model window comes with a childless root node already in place. When the New Nodes tool is active, as shown by the highlighting in Figure 5.1, clicking on such a childless node causes two arcs and two subnodes to be added to this node. Clicking on a node that is not childless adds to it one additional arc and subnode. Figure 5.2 shows how the tree will grow when the mouse is clicked successively on the asterisked nodes. Nodes and arcs are created with default labels that can be edited with the pen tool.

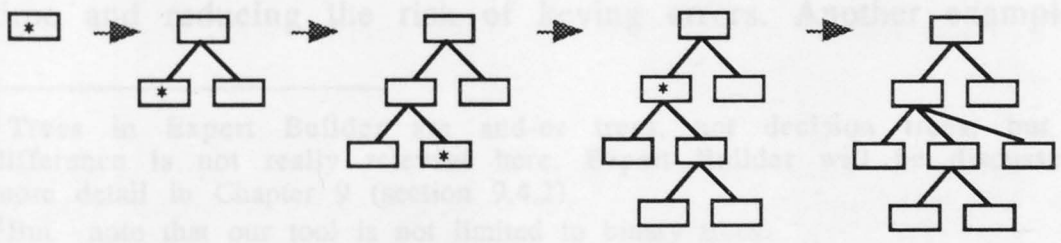


Figure 5.2 Growing a decision tree with four mouse clicks.

This method of constructing trees can be contrasted with the method provided by the Expert Builder program (Webb 1993). In Expert Builder, the user creates a tree using separate node and arc tools.[1] Each click with the node tool creates a single node which the user locates manually by dragging with the mouse. Nodes are subsequently 'wired together' in pairs using the arc tool. Our observation of users with Expert Builder suggested that this approach is much slower and more error-prone than our tool's method, although Expert Builder is more flexible in that trees can be constructed bottom-up. However, whether such flexibility would be helpful for constructing decision trees is doubtful. For decision trees, questions are usually formulated within the context of answers that appear at higher levels on the same branch and so a top-down construction method is appropriate. Also, the addition of subnodes to a childless node in pairs, rather than singly or perhaps in threes, reflects our observation that most decision trees appearing in curriculum materials are binary trees.[2]

Another advantage of our construction method is that the tree is always coherent; its structure cannot become fragmented, possibly leading to errors caused by 'orphaned' nodes or subtrees. This property is preserved by the eraser tool. Clicking on a node with the eraser deletes the node together with (if the node is not a leaf, and following user confirmation) the node's entire subtree.[3] A disadvantage of the method is that a tree cannot be disassembled and reconstructed at will, as is possible with Expert Builder.

Following standard recommendations for interface design (e.g. Schneiderman 1987) we provide some shortcuts for frequent — and not-so-frequent — users. For example, dialogue boxes that are used for editing node labels and arc labels contain popup menus showing labels which already appear on the tree. Selecting one of these automatically enters its text into the edit field, saving time and reducing the risk of keying errors. Another example is

---

[1] Trees in Expert Builder are and-or trees, not decision trees, but that difference is not really relevant here. Expert Builder will be discussed in more detail in Chapter 9 (section 9.4.2)

[2] But note that our tool is not limited to binary trees.

[3] Unfortunately, the user cannot reverse the deletion once it is done. The absence of an 'Undo' command is particularly unfortunate here.

the dialogue box that is used for editing an arc label. A 'Go to node below' button is provided which, when clicked, at once produces another dialogue box for editing the label of the arc's child node, eliminating the need to select the child node with the mouse. Other shortcuts, intended for advanced users, are invoked by modified clicks (e.g. clicking with the shift button depressed). These shortcuts along with other operating procedures are documented in the online help.

Refinement of PDT through many stages of development was observed to increase markedly the usability of the tool. Some of this was due to the automation of low-level operations. For example, users never need to adjust the size of node boxes: a box is automatically set to a size that matches (within limits) the text which it contains.

5.1.1.2 Maintaining the tree's shape

The New Nodes tool attempts to locate newly created nodes without redrawing any part of the existing tree. With a tree that is constructed dynamically, however, it will not always be possible to do this without overlapping existing arcs or nodes. Supposing a tidy tree is desirable, the question arises as to whether the user or the system should take responsibility for maintaining a tidy shape. In a direct manipulation environment it seems reasonable to make the tree's shape the user's responsibility and so we implemented a Drag tool (the arrow icon in Figure 5.1). Dragging a node with the mouse relocates the node and its subtree.

However, observations of users with this tool led to concern about the amount of time that was sometimes spent in making very minor adjustments to the tree. We tried replacing the Drag tool with an internal routine which detected when a clash would occur and automatically redrew all or part of the tree to prevent the clash. However, this approach was strongly disliked by some users.[4] In the final version of the tool the issue is left open. The

[4]In formative tests, one student while using PDT was asked to account for her visible frustration. She explained that she had carefully laid out her tree so that one node, which featured a question to which she attached particular importance, was located at a slightly higher level than its

Drag tool is available and the automatic routine has been brought forward to the user interface as a 'Tidy' tool (the fourth icon from top in Figure 5.1). Clicking on a node with the Tidy tool redraws the node's subtree in a way that eliminates clashes.

5.1.1.3 Distinguishing label types

The node labels of a decision tree have different interpretations depending on where they occur. On a properly completed tree, leaf nodes have labels that represent decisions and non-leaf nodes have labels that represent questions. Whilst it would be possible to leave this distinction as an implicit one, formative tests indicated that users were helped by graphical annotations that clearly distinguished different label types.[5] In PDT, decision labels are shown in bold type. An automatic routine could annotate a properly completed tree, but if the annotation is to be performed incrementally (and reliably) during the tree's development then the user must indicate how labels are to be interpreted. The solution adopted is to include type selection buttons in the node label dialogue box (see Figure 5.3). Immediate visual feedback on the selected type is given on the tree diagram by the font style in which the labels are displayed. If the user does not already supply one, a question mark is automatically postfixed to the label of a question-type node.

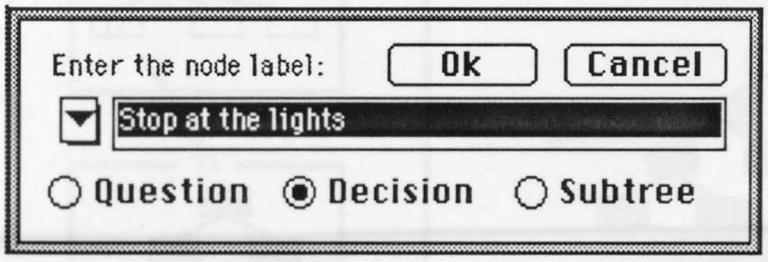


Figure 5.3 A node label dialogue box.

siblings. The automatic tidying routine had demolished this relationship. She was not impressed by the neatly symmetrical tree that it produced.

[5] Especially since labels have a third possible interpretation, i.e. as subtree references. This is discussed below.

5.1.2 Linking decision trees

The principle that models should be composable (discussed earlier) is especially important in the case of decision trees. Not only do they they quickly become large in a spatial sense, they also become in the words of Michie (1990) 'opaque, unmemorisable and not mentally checkable'. Michie ascribes this property to the overload of short-term memory which comes from trying to follow a sequence of questions and answers each of which assumes the context of its predecessors.

We agree that large decision trees are hard to understand. Our approach is to try to manage the complexity by allowing a large tree to be replaced by two or more smaller ones that may be developed and tested independently.

Decision tree linking can happen in at least two different ways. The more simple way, which we call 'continuation' linking, is illustrated by Figure 5.4. Here, a leaf of the tree in window T0 is labelled with the name of another decision tree window, T1. In logical terms, trees T0 and T1 together are equivalent to the tree shown on the right of Figure 5.4, in which the root of T1 has been grafted onto the leaf of T0, replacing that leaf's label with the question from the root of T1.

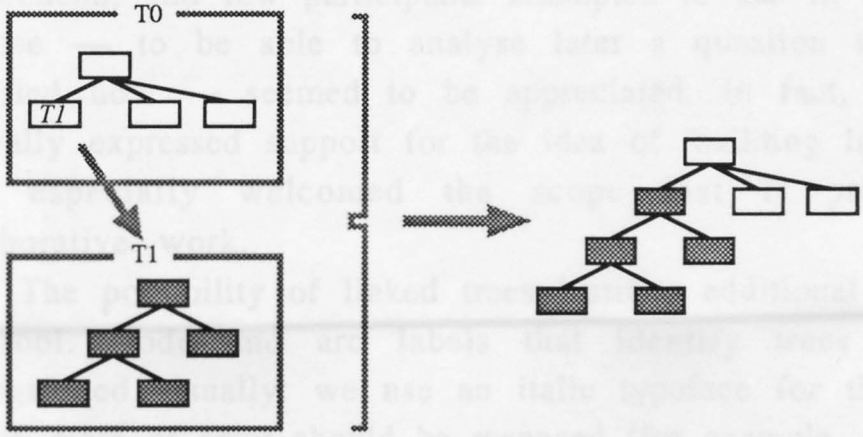


Figure 5.4 Linking decision trees by the continuation method

The other method of decision tree linking, which we call 'interior' linking, is illustrated by Figure 5.5. Here, the node in T0 which makes the reference to T1 is a non-leaf node. The arcs of this node have labels which appear as decisions of the other tree,

and this is essential for the link to be valid. In logical terms, T0 is equivalent to a tree which has the reference node substituted by T1. The matching pairs of arc labels in T0 with decisions in T1 enable branches to be properly connected.

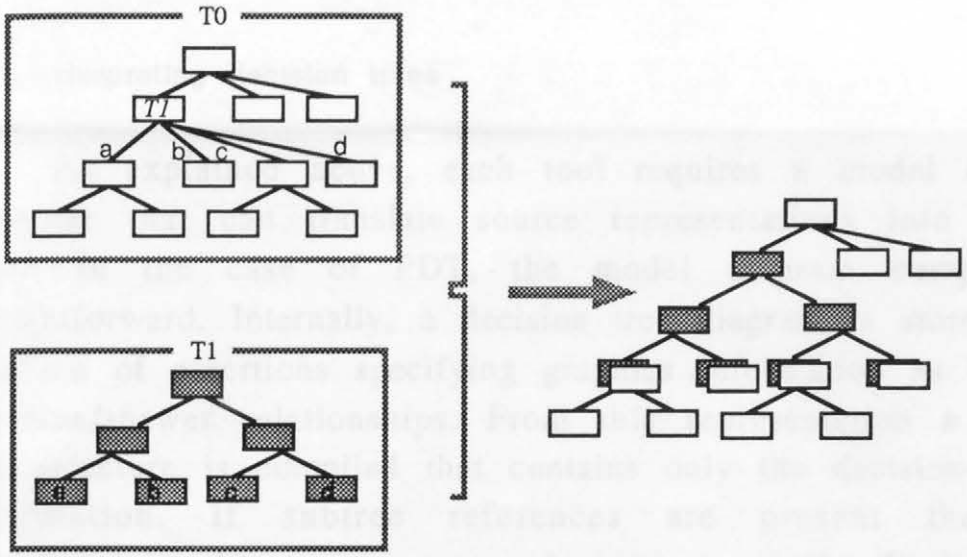


Figure 5.5 Linking decision trees by the interior method

We implemented both continuation linking and interior linking in PDT. In formative trials with teachers and students, continuation linking was easily understood and all participants used it successfully. They found interior linking harder to comprehend, and few participants attempted to use it, although its purpose — to be able to analyse later a question that can be specified now — seemed to be appreciated. In fact, participants generally expressed support for the idea of 'building in parts' and they especially welcomed the scope that it provided for collaborative work.

The possibility of linked trees justifies additional features of the tool. Node and arc labels that identify trees should be distinguished visually: we use an italic typeface for this purpose. A new class of error should be managed (for example, a node may refer to a non-existent subtree, and references may be circular). Some means of creating subtree windows is necessary. In our tool, when a user initially selects the 'Subtree' button in the node label dialogue box (Figure 5.3) the system immediately offers to create the new window. We should also mention the support which a tool can offer to users in browsing a model that is distributed over

several windows. In PDT we provide a hypertext-like browsing facility: if a node refers to a subtree then clicking on the node with the arrow tool will bring the subtree window to the front, and clicking on the root of the subtree window will return to the front the parent window (or one of them, if there are several).

5.1.3 Interpreting decision trees

As explained above, each tool requires a model diagram compiler that can translate source representations into normal form. In the case of PDT, the model diagram compiler is straightforward. Internally, a decision tree diagram is stored as a database of assertions specifying graphics information as well as question/answer relationships. From this representation a Prolog tree structure is compiled that contains only the decision-making information. If subtree references are present then the appropriate subtrees are merged into a single Prolog tree structure. From each branch of this tree structure, one rule in normal form is generated.

Importantly, the sequence of conditions of a normal form rule preserves the question/answer sequence of the decision tree branches. The PKRL compiler also preserves this sequence. The result is that, when the resulting code is run by the Primex depth-first backward chaining interpreter, the user is queried in a way that corresponds exactly to a standard top-down traversal of the decision tree diagram. On any given run, only one branch is traversed and there is no backtracking. Interestingly, this interpretation largely immunises the user to any typographical errors that he or she might make on creating the tree. It makes no difference if the same question appears in different text on different branches of the tree, because no more than one of these textual representations can be encountered at run time.

5.2 PFT: the factor table tool

Figure 5.6 shows a model window created by PFT, the factor table tool. This section explains how the tool manifests the common design principles and also discusses issues that are

specific to the tool. As in the preceding section, we consider in turn model editing, linking, and interpretation. It may be helpful to refer to the factor table help sheet in Appendix 7.

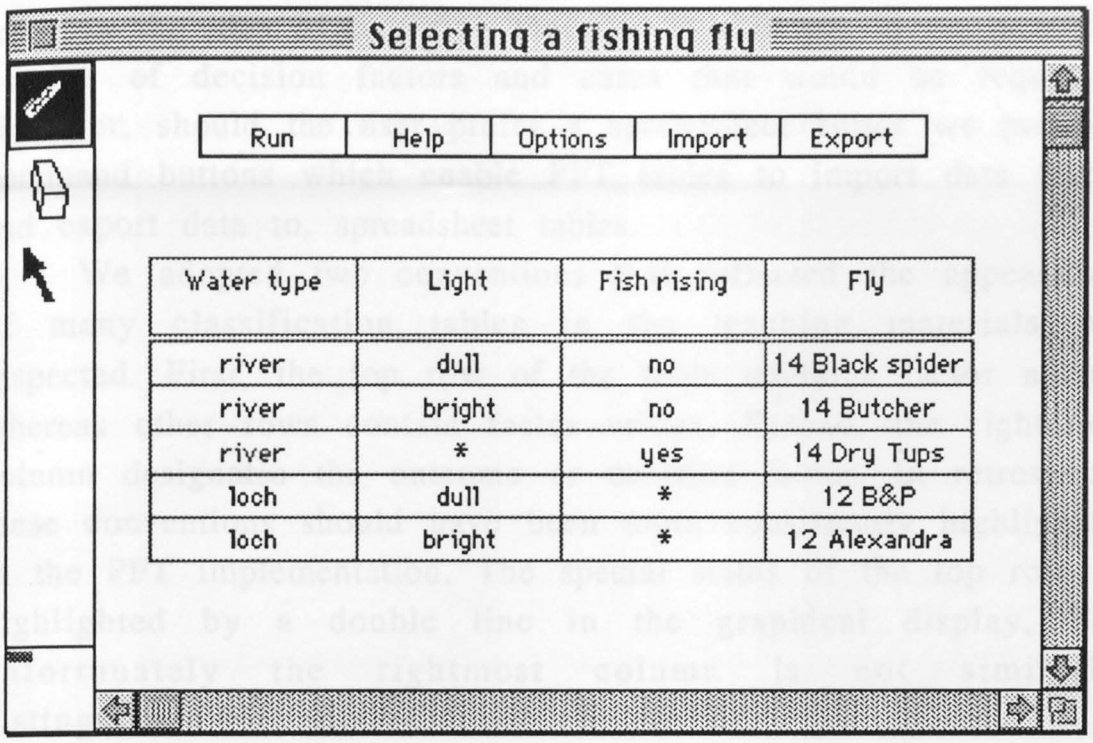


Figure 5.6 A factor table window with model

5.2.1 Constructing and editing factor tables

The factor table editor, although quite simple, nonetheless requires good design. Here we discuss two of the main issues: table layout and data entry.

5.2.1.1 Table layout

An early prototype of the tool did not provide any form of graphical table. Instead the table was created with a text editor and the user was expected to enter special characters (tab and return) to separate columns and rows. Not surprisingly, this was highly error prone.

A pre-drawn table is certainly better. However, the question arises as to the size of table that should be pre-drawn. One approach is to supply a large or virtually infinite table, like a typical spreadsheet; another is to ask the user to specify the size,

and to provide facilities whereby rows and columns can be later added or deleted as necessary. We adopted the second approach, mainly because it could be implemented more easily within our overall design framework, although there also seemed to be some merit in the idea of requiring the user to estimate in advance the number of decision factors and cases that would be required. However, should the user prefer a spreadsheet editor we provide command buttons which enable PFT tables to import data from, and export data to, spreadsheet tables.

We adopted two conventions that reflected the appearance of many classification tables in the teaching materials we inspected. First, the top row of the table contains factor names whereas other rows contain factor values. Second, the rightmost column designates the outcome or decision factor. In retrospect, these conventions should have been more consistently highlighted in the PFT implementation. The special status of the top row is highlighted by a double line in the graphical display, but unfortunately the rightmost column is not similarly distinguished.[6]

In fact, the system will drastically misinterpret the table of a user who, misunderstanding the convention, enters his or her outcome factor into the leftmost column. The PFT editing tools provide no easy way to retrieve a bug of this kind. This is a weakness which could have been eliminated by a method similar to that used to distinguish question-type nodes from decision-type nodes in the decision tree tool. We did identify the problem during the formative trials, but underestimated its importance until the school trials revealed that the bug affected a sizeable number of factor tables.

5.2.1.2 Data entry

Figure 5.7 shows the dialogue that is generated by the pen tool when the user clicks on a cell. As with the dialogues of the decision tree tool, this one contains features designed to expedite data entry and reduce errors. If the selected cell (shown

[6]Of course, the identity of the 'rightmost' column can be changed by editing since columns can be freely added to a table.

highlighted in the diagram) is presently empty but the cell above is not empty then the edit field is automatically prefilled with the value contained in that cell, reflecting the observation that values in adjacent cells within the same column often recur. The same observation lies behind the presence of the popup menu: this menu contains all the values that appear anywhere in the column, so that any of them can be rapidly reselected. The Next button — or the return key, its keyboard equivalent — completes the editing of the current cell and immediately brings into the editor the value of the cell's right-hand neighbour (or the leftmost cell of the row below, if the current cell is in the rightmost column). A table can be quite rapidly filled with data from the top left cell to the bottom-right cell by repeatedly typing a cell value followed by return. Formative tests showed this to be a suitable method for many users, but it can be argued that more flexibility (e.g. the ability to traverse the table in the reverse sequence) is desirable.

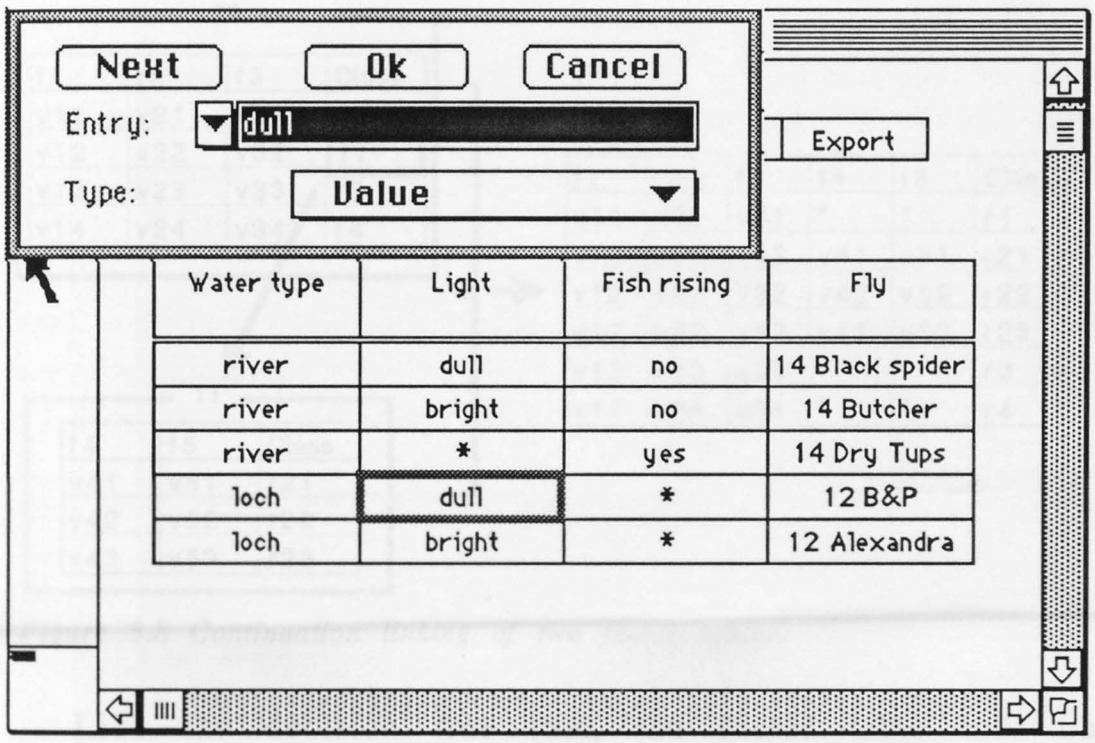


Figure 5.7 Editing data in a factor table

5.2.2 Linking factor tables

Analogously to the case of decision trees, there are at least two ways of linking factor tables. In continuation linking, a cell in

a table's rightmost column specifies a subtable instead of giving an outcome factor value directly. In interior linking, a cell in a table's header row specifies a subtable instead of naming a factor. Following our experience with the decision tree tool we decided to implement only the first method, since we considered this method to be more likely to be used within the timescale of this research.

Figure 5.8 illustrates how two continuation linked tables T0 and T1 may be regarded as logically equivalent to one larger table. The highlighted cell in the second row of T0 refers to table T1. For linking to be possible these tables must have disjoint sets of factor names, the union of which gives the factor names of the joined table. Rows of T0 other than the second row appear in the joined table with wildcard (don't-care) values for factors that do not occur in T0. The second row of T0 appears in the joined table replicated and extended once for each row of T1. An algorithm for joining tables is easily generalised from this illustration.[7]

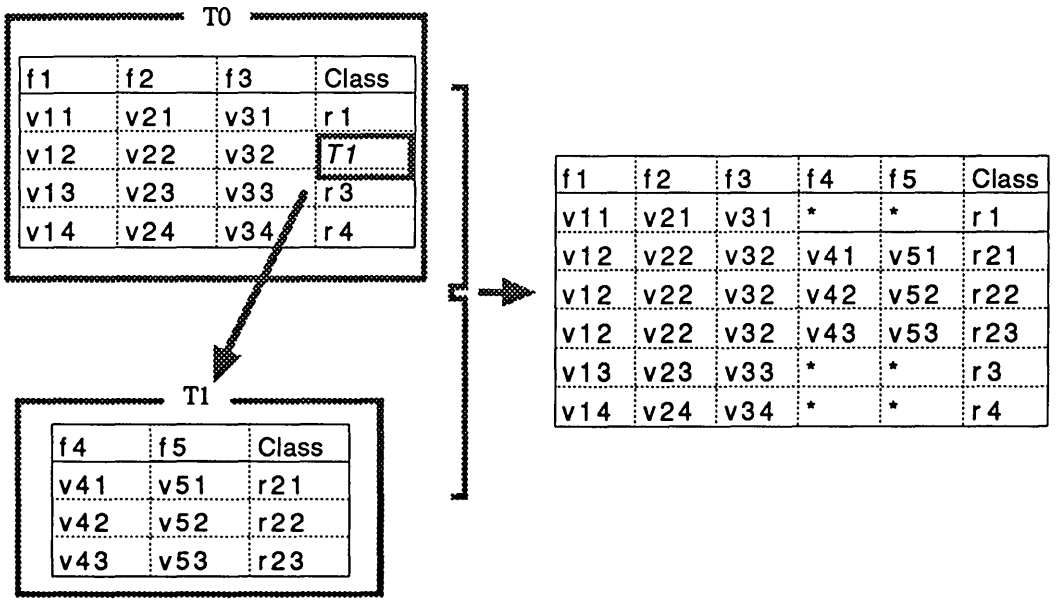


Figure 5.8 Continuation linking of two factor tables.

Like the decision tree tool, the factor table tool has environment features which assist users to build linked models. The edit dialogue (see Figure 5.7) enables a cell to be designated

---

[7] This rather abstract description should not be taken to imply that table linking is incomprehensible to users. An explanation of it that we found helpful was along the lines of: 'If we get to the highlighted cell in T0, then look up the answer in T1'.

as type 'Table name' and such cells are written with an italic typeface. The system will automatically offer to create a window for a new subtable. Errors such as invalid table references are trapped and hypertext-style navigation of links is provided, both along the lines of the corresponding decision tree facilities.

5.2.3 Interpreting factor tables

In the outline specification we proposed that information from a factor table model would be transformed into efficient runnable code by means of ID3 or some similar induction technique (see sections 4.3.2 and 4.4). The induction algorithm would be implemented as part of the model diagram compiler.

The present version of PFT does indeed incorporate the induction algorithm, but we do not make its use mandatory. The user may choose between induction and a more naïve method. The latter simply compiles one normal form rule per table row, with rule conditions composed from cell entries in left-to-right column order, omitting wildcard entries. Figure 5.9 shows the dialogue (accessed by the Options button shown in Figure 5.6) which offers the two methods. Note that the choice is presented as one between different strategies for runtime questioning. The 'Left to right' option corresponds to the naïve compilation method and the 'Smart' option corresponds to induction. For the purpose of the large-scale trials we selected 'Left to right' as the default selection, for reasons which we now explain.

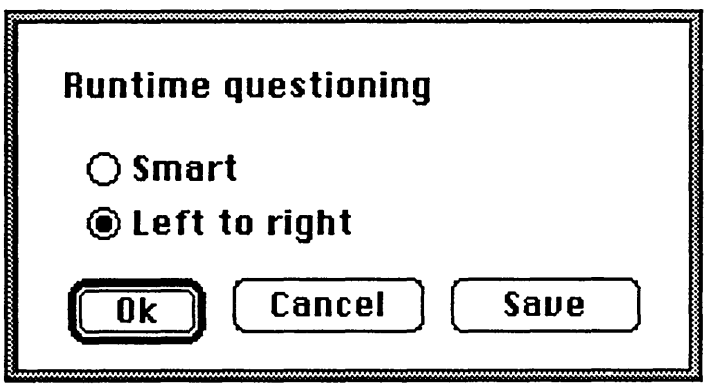


Figure 5.9 Compilation options in the factor table tool

There are no technical obstacles to the implementation of ID3-based induction. As mentioned previously, this technique

attributed to Quinlan (1979) has been embedded in many commercial KBS products. We selected a variant of ID3 called ACLS (Analogue Concept Learning System; Paterson & Niblett 1982) since it seemed well matched to our application. In ACLS, the 'windowing' mechanism that enables ID3 to cope with large data sets is removed, simplifying the implementation, and the method is adapted to cope with numeric as well as symbolic data.

Appendix 9 compares how PFT compiles one particular factor table, in the domain of ecological modelling, by the induction and naive methods. As can be seen, induction generates a much smaller PKRL representation in which one attribute disappears entirely and the others are re-sequenced with respect to the original table. Both methods generate code that correctly classifies each example in the table, but the runtime behaviour differs markedly between them. When running the naïve code the system always asks four questions — one per attribute — to reach a solution and the question sequence exactly corresponds to left-to-right column order. When running the induction-generated code the system never asks more than three questions and the sequence of questions bears no obvious relationship to column order.

Less obviously, there is also a difference between the two compilation methods in the effect of making a change to the source model. After editing a single cell, the induction compiler may generate code that is structurally different from the code that pertained prior to the edit. In consequence, runtime questioning may change radically between runs. The naive compiler on the other hand generates code that at runtime behaves minimally differently.

This example illustrates the efficiency of induction: a classifactory scheme is derived that distinguishes between the input cases with an optimal (or near optimal) minimum number of tests. However, the example also reveals a potential problem: the runtime behaviour generally bears no clear relationship to the user's model and it disregards some of the information that the user has provided.

We alluded to this problem previously and noted that its significance was initially unclear.[8] In the formative trials, discussions with teachers — who were participants in a knowledge-based modelling course that formed part of their Master of Education programme — suggested that the problem was serious. The teachers argued that children would be disoriented by an opaque runtime interpretation and believed that children might unduly ascribe the unexpected system behaviour to an error in their model. In the case where a child's model actually did contain an error, an opaque and unstable runtime interpretation would not be helpful in debugging. The teachers doubted whether these drawbacks were outweighed by the induction compiler's ability to generate efficient code. In fact, it might be preferable to encourage children to take responsibility for efficiency, as could be done with the naïve approach by arranging the table with the most discriminating factors to the left, since this might stimulate reflection on the properties of the domain.

These reservations can be illustrated by reference to the children's model[9] shown in Figure 5.10. When this model is run with the induction option selected, the system's behaviour is to obtain a value for the `Food` attribute. A butterfly is then immediately categorised. This is of course justifiable in information-theoretic terms: under the closed-world assumption (Reiter, 1978) the only butterflies that can be classified are the four in the table and each butterfly is uniquely identified by its `Food` value.[10] However, the children who build the model may not adopt an information-theoretic perspective. Indeed, it seems quite possible that prior to runtime they will have considered `Colours`, `Food` and `Habitat` as more or less equally important attributes in

---

[8] The only other known use of induction in an application aimed at children is Michie at al (1989). However, theirs is a learning-by-teaching system, not a modelling system, and its design is such that the properties of induction discussed here are probably not discernible at the user interface.

[9] The authors of this model are a pair of ten-year old children who constructed the table with the aid of a school library book. They ran the model under the default naïve compilation setting and thus were spared the confusion that is conjectured here.

[10] `Habitat` is an equally good discriminating attribute and the system's preference for `Food` is essentially arbitrary.

butterfly classification. If so, they may now be perplexed that the system's decision-making takes account only of the `Food` attribute. Worse, if the children now add to the table information about a certain butterfly that has (say) `Food=Oak leaves` and `Habitat=marsh`, the induction compiler will recognise that the optimal discrimination factor now becomes `Habitat`. The next run will therefore ignore `Food` and prompt only for a value for `Habitat`, causing perhaps an increase in the level of children's confusion.

Colours	Food	Habitat	Names
Blue	Wild thyme	Woodland	Large Blue
Blue	Wild flowers	Chalk and limestone downs	Adonis Blue
Purple	Oak leaves	Top of oak trees	Purple Emperor
Mostly brown	Sheeps fescue	Chalk and limestone	Silver spotted skipper

Figure 5.10 Children's butterfly model.

The formative trials with teachers and students using the induction-based compiler indicated that these objections are empirically justified. The researcher was frequently called upon to defend the behaviour of the system. In contrast, when the naïve version of the compiler was used, teachers and students were almost never surprised by the runtime behaviour. This explains why we selected naïve compilation as the default selection for the purpose of the large-scale trials.[11]

Induction remains as an option, however, because it seems at least possible that its capabilities will be useful in some contexts. For instance, in the Appendix 9 example, the fact that one particular attribute can be ignored in classifying the table's cases may well reveal something interesting to an ecological modeller. However, to identify the useful contexts will require further research.

What can be concluded from this? We suggest that in designing knowledge-based modelling tools, we should favour not

[11] It remained possible that children in the trials could change the setting. However, we would be able to detect if this happened since the PKRL generated by a model is stored as part of the model file.

only representations which are transparent to users, but interpretation methods too.

We can further illustrate the point with one more observation from the formative trials. It is common for KBS developers to implement induction with a 'consistency check' which flags as invalid a table that contains two clashing rows, i.e. rows which are identical in all respects save the outcome factor value. Such a check is justified on the assumption that classes are functionally determined by (combinations of) factor values. We originally implemented consistency checking in PFT. However, in formative tests two students protested that the system had rejected a table similar to the one shown in Figure 5.11. They were not satisfied with the researcher's suggestion that a new factor might be added to the table which distinguished the first two cases. On reflection, their point was accepted. A strictly functional view of classification prevented the expression of alternatives (i.e. disjunction), with no clear compensating benefits. The consistency check was removed.

Swimming	Disco	Mountain climbing	Holiday centre
yes	yes	no	Blackpool
yes	yes	no	Skegness
yes	no	no	Mull
no	no	yes	Torridon

Figure 5.11 A factor table with 'inconsistent' cases.

5.3 PCT: the classification tree tool

In this section we discuss the design of PCT, the classification tree tool, following the same headings as for the decision tree and factor table tools. Figure 5.12 depicts a PCT window with a small model. As explained below, the window does not show the 'hidden layer' of feature information consisting of a set of terms of the form `Attribute=Value` that may be attached to each node. It may be helpful to refer also to the classification tree help sheet in Appendix 7.

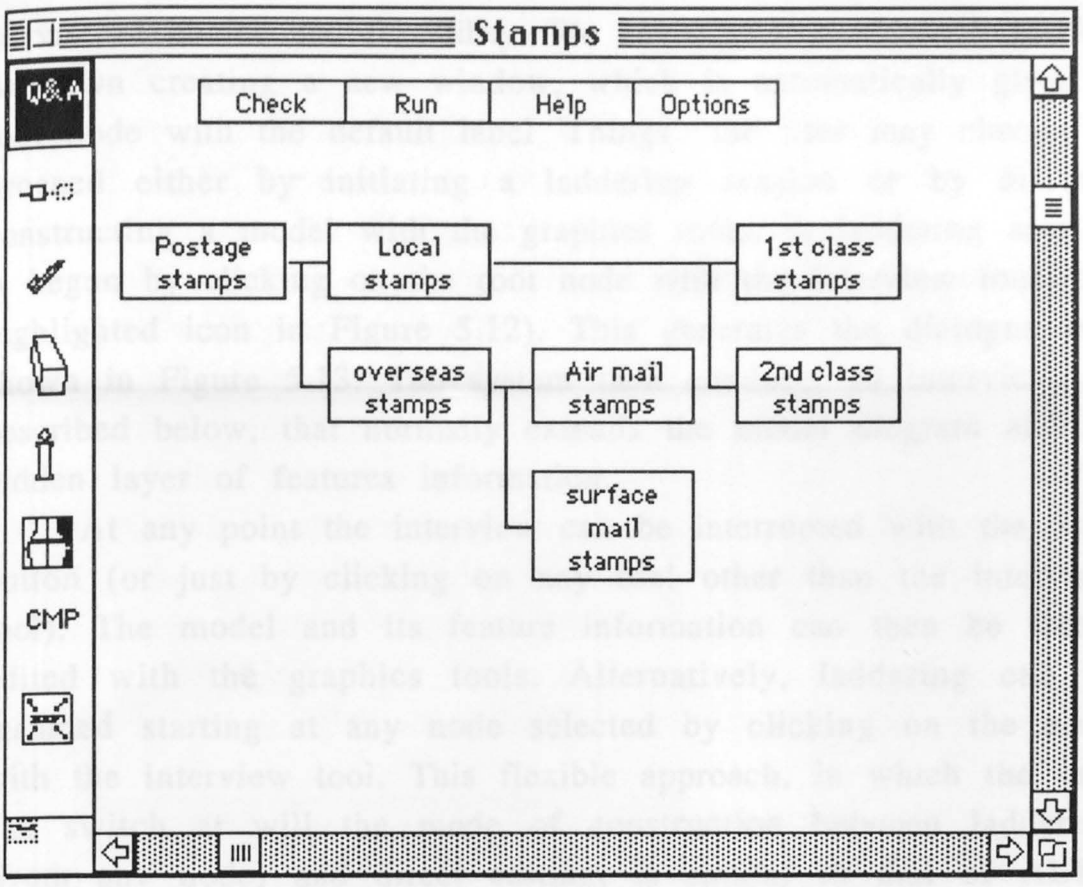


Figure 5.12. A classification tree window with a small model.

5.3.1 Constructing and editing classification trees

In our previous discussion leading to the tool's outline specification, we proposed that the classification tree tool should provide two alternative methods of entering knowledge, viz. laddering and direct editing of a graphical tree structure. Laddering seemed to be a promising way of giving strong support to the modeller, but if it proved to be too restricting then direct graphical editing might ensure that the tool would still be usable in some form. An important design issue was how to combine these two approaches to building a model. Other key design issues concerned the diagram layout, the inheritance mechanism and the laddering dialogue strategy. We discuss these in turn below.

5.3.1.1 Combining laddering with direct editing

Since we were uncertain about the method by which users would prefer to operate the tool, it seemed wise to allow a large

amount of choice and flexibility. We achieved this in the following way. On creating a new window, which is automatically given a root node with the default label 'Things', the user may choose to proceed either by initiating a laddering session or by directly constructing a model with the graphics tools. A laddering session is begun by clicking on the root node with the interview tool (the highlighted icon in Figure 5.12). This generates the dialogue box shown in Figure 5.13. The system then conducts an interview, as described below, that normally extends the model diagram and its hidden layer of features information.

At any point the interview can be interrupted with the Stop button (or just by clicking on any tool other than the interview tool). The model and its feature information can then be freely edited with the graphics tools. Alternatively, laddering can be resumed starting at any node selected by clicking on the node with the interview tool. This flexible approach, in which the user can switch at will the mode of construction between laddering (from any node) and direct editing, is similar to that of ALTO (Major & Reichgelt 1990). In formative tests, users were observed often to use the facility to interrupt laddering and resume laddering from another node. However, they were only occasionally observed to switch from laddering to direct editing.

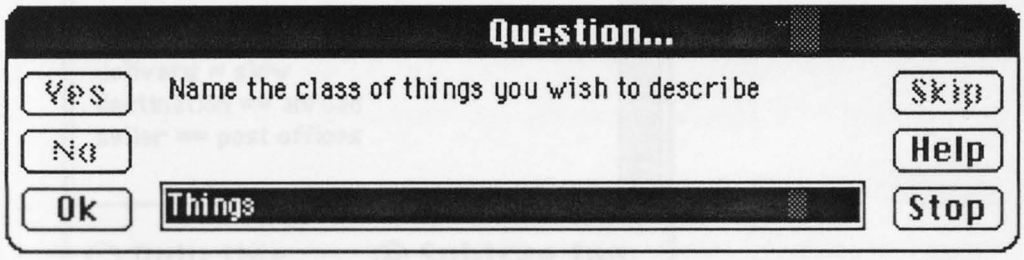


Figure 5.13 The initial dialogue in a laddering session

5.3.1.2 Diagram layout

The combination of laddering with direct editing partly explains the tree's grid-like style of layout (see Figure 5.12). Importantly, we wanted a classification tree to look quite different from a decision tree. A different appearance would help to emphasise the fact that the two kinds of diagram have very

different semantics.[12] Additionally, however, the grid-like layout of classification trees facilitates the automatic tidy placement of new nodes created by laddering. Unlike the case for the decision tree tool, in PCT the system must be capable of assuming complete responsibility for the tree's tidy shape. A model diagram that is constructed solely by laddering will involve no direct manipulation of graphics by the user.

As noted above, a model normally comprises a hidden layer of feature information in addition to a visible diagram. It was considered that to make the information visible on the diagram could produce a cluttered and confusing display. Therefore PCT provides an 'information' tool (the one with the 'i' icon in the model window) which shows a node's feature set, and that of its subtree if required, in a separate dialogue. An example of this dialogue is shown in Figure 5.14.[13]

Figure 5.14 Viewing feature information from a classification tree

The dialogue can be retained on screen, so it is not necessary to commit its contents to memory. Even so, that this information is not visible as an integral part of the model diagram is in one sense

---

12 We note how this consideration contrasts with our design principle that different tools should have a strong family resemblance (discussed earlier).

13 The double-equals shown in the dialogue's text box denotes an inherited as opposed to a locally defined feature. We discuss this dialogue in more detail later.

unfortunate: compared to other tools, the user has to work harder to obtain a full description of the model. A future development of the tool should consider alternative methods of displaying feature information.

5.3.1.3 The inheritance mechanism

The basic notion of inheritance may seem to be straightforward: features aggregate from classes to subclasses. Inheritance may indeed be straightforward in its simplest implementations. However, when AI researchers sought to extend these simplest systems with facilities for handling exceptions or to provide multiple inheritance (or both) they encountered significant problems. Some of these problems are technical, others are more philosophical; they have been much discussed in the literature (e.g. Brachman 1985, Touretzky 1986, Cornell-Way 1994). Our intention here is not to reiterate that discussion but to explain in what ways issues of inheritance were relevant to this research. We explain why it was decided to implement PCT with an inheritance mechanism equipped with both exception handling and multiple inheritance, the latter taking a somewhat novel form.

The standard justification for exception-handling is that the real world contains exceptions to many generalisations. For example, in Figure 5.15 we represent the familiar notion that penguins are exceptional birds by including the feature `can fly=false` in the feature list for the `penguins` class.

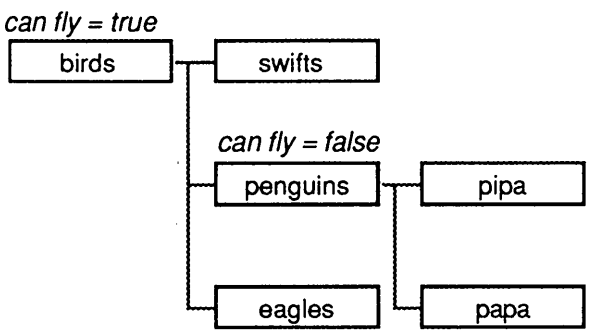


Figure 5.15 Representing exceptions by overriding inheritance

An inference mechanism for inheritance that is equipped with exception handling recognises that the feature `can fly=true` that would normally be inherited by the `penguins` class from the

birds class is in this case overriden (i.e. cancelled). Such a mechanism gives to the penguins class, and hence also to the pipa and papa subclasses, only the value false for the can fly attribute.

We implemented exception handling in PCT and discussed examples like the one above with teachers and students. They saw in them nothing problematic. The general principle, that features of subclasses override those of superclasses, was considered to be quite accessible to children. It was recognised that in the penguin example, without a mechanism for exception handling we would be forced to adopt a less direct method of representation, such as the one shown in Figure 5.16. It seemed unlikely that children would find this easier. Considering Brachman's (1985) objections to cancellation within inheritance systems, we note that his criticisms are partly directed at an undisciplined use of the mechanism (e.g. defining a giraffe as an elephant which is not big, not grey, does not have a trunk, etc.) and partly at attempts to claim more than is justified for what such systems can achieve. We do not regard these objections as particularly serious in the context of our application.

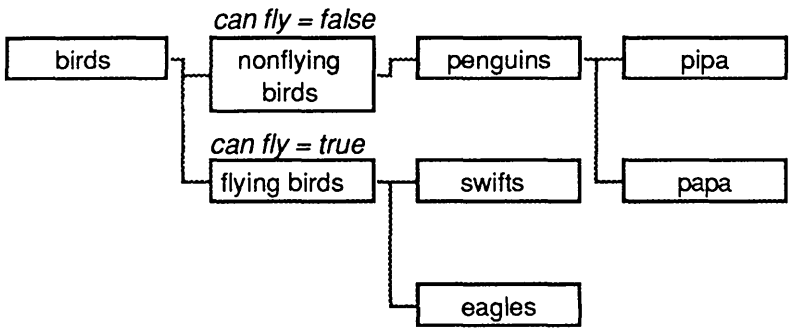


Figure 5.16 Representing exceptions without an overriding mechanism

Provision for multiple inheritance has a similar justification to provision for exceptions: objects in the real world are often naturally viewed as belonging to more than one class. In a system which does not explicitly support dual membership, a representation for such objects is often less natural and less concise. In PCT, therefore, multiple inheritance is directly supported. Figure 5.17 shows how a model can be constructed that captures the salmon's status as both a sea fish and a river fish. Notice that the tool highlights the dual membership by showing

two copies of the salmon node, each with an italic label. This was preferred to the alternative of showing just one salmon node with two arcs, one to each parent, mainly because we wanted to preserve the neat grid-like layout of the tree.[14]

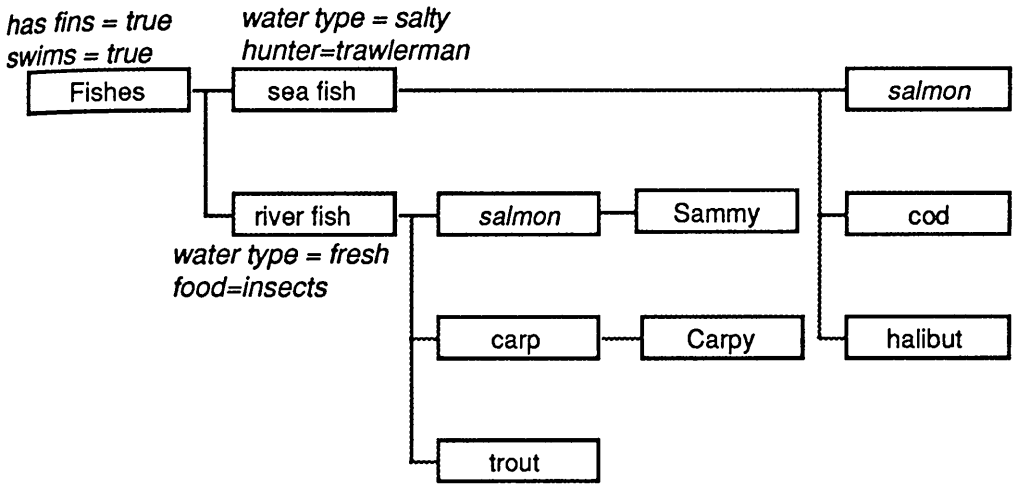


Figure 5.17 Multiple inheritance

The problems associated with multiple inheritance are mostly related to efficiency and semantics. An efficiency problem arises because to identify the features of a class, the inference mechanism must potentially search a large number of inference paths from the class to the root. However, tests with PCT indicate that this is unlikely to be a serious issue with the modestly-sized models that children can be expected to construct.

A semantics problem arises because attribute values that are inherited from different paths may clash in a way that is not easily resolved. In Figure 5.17 for example, although there may be no difficulty in agreeing that Sammy should inherit both features of Fishes together with hunter=trawlerman from sea fish and food=insects from river fish, the value that should be inherited for the water type attribute is less clear. A simple approach to this difficulty, used by ALTO and many other inheritance reasoning programs, is to select whichever value is first found by a depth-first search of the inheritance network. With such a

[14]The use of two copies does not imply that the user needs to duplicate any information. If the user wants to edit or view the features of salmon he or she can do so equivalently at either node. Likewise, Sammy could appear as a child of either salmon node; the subtrees of a class are the union of the subtrees appearing at each node labelled with the class name.

reasoner the value of `water type` is essentially an arbitrary selection between `salty` and `fresh`.

Another example, provided by Touretzky (1986) and shown in Figure 5.18, demonstrates that the combination of exception-handling with multiple inheritance may be troublesome. In this example, Clyde the elephant is a member of the class of Royal Elephants which are exceptional among elephants in that they are not grey. Without the arrowed link, an inheritance reasoner equipped with exception handling duly concludes that Clyde is not grey. However, when the arrowed link is added to the network, the reasoner (if it uses a first-found strategy) may change its decision.

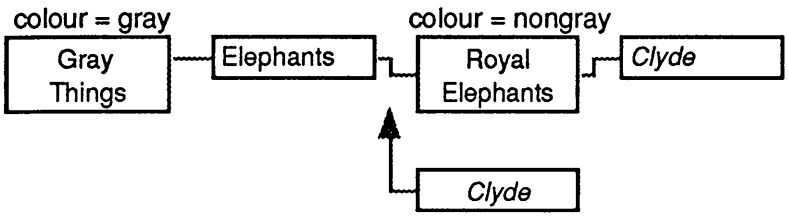


Figure 5.18 Touretzky's example

Touretzky argues that adding the new link should not affect any conclusions about Clyde since the link only redundantly asserts what could already be inferred, i.e. that Clyde is an elephant. He proposes a new rule for inheritance inference, which he calls the 'inferential distance' rule, that eliminates the ambiguity in the example. Informally, Touretzky's rule regards class A as 'nearer' to class B than to class C if and only if there is an inference path from A through B to C, and not vice-versa. Thus in Figure 5.18, and regardless of whether or not the arrowed link is present, because there is an inference path from Clyde to Elephants *via* Royal Elephants, and there is not a path from Clyde to Royal Elephants via Elephants, the inferential distance rule concludes that Clyde's inheritance from Royal Elephants takes precedence over that from Elephants. Under Touretzky's rule, therefore, Clyde is not grey. Redundant links never make a difference because they cannot affect what he calls the 'between-ness' relationship of classes.

However, Touretzky's rule does not always resolve a clash. In Figure 5.17 for example there is no path from `salmon` via `river`

`fish` to `sea fish`, nor vice-versa. Touretzky argues that in such cases, if the system is to behave rationally then it must *not* choose. In multiple inheritance systems, not all forms of ambiguity can be rationally resolved.

We accept that Touretzky's rule provides a more satisfactory basis for resolving some inheritance clashes than is offered by 'first-found' methods or by methods which prioritise shortest paths. The PCT inheritance mechanism is based therefore on Touretzky's rule. Where this rule is inconclusive, the user is given responsibility to determine the outcome, as illustrated below. The system makes no arbitrary choices.[15] In discussions with teachers on the design of the classification tree tool there was some limited feedback in support of this approach. However, it should be said that teachers did not regard the issues relating to inheritance mechanism as having a high priority.

As noted previously, the study by Greene (1989) suggests that class hierarchy is relatively well understood by children at the age of 7 or 8. Greene's study, however, seems to have been restricted to the simplest form of class hierarchy, lacking examples which require the consideration either of exceptions or multiple inheritance. There is no known research on how well children may fare with systems that include these concepts.

5.3.1.4 Dialogue strategy

Previously in this thesis we acknowledged the potentially crucial role of the dialogue strategy in any educational tool built around a contrived knowledge acquisition technique such as laddering. Although laddering is associated with a typical set of questions, based originally on an analysis by Hinkle (1965), the literature on automated strategy for the technique seems to be thin. Also, a lack of consensus is evident in what exists. For instance, we note that Rugg & McGeorge (1995) recommend that questioning can equally well start anywhere in the domain and should aim to develop the hierarchy depth-first, whereas Major &

---

[15] Even if there was a rational basis for resolving clashes, our previous observations would indicate that unless the rationale was transparent to the users it would probably not be well received by them.

Reichgelt (1990) recommend starting from the top-level class and asking the user whether depth-first or breadth-first development is preferred.

However, even if KBS experts had reached agreement on a best strategy it is doubtful whether such a strategy would reflect the distinctive requirements of an educational context. As discussed earlier, classroom users are domain novices not domain experts. They may have little understanding of computer systems. The product of the interaction, i.e. the model, whilst important, is less important than the learning which the interaction may promote in the model builder. We argue that these considerations justify a strategy with the following broad characteristics:

- Questions will be designed not only with the aim of eliciting knowledge but also to stimulate reflection;
- There will be frequent opportunities to revise answers;
- The system will sometimes restate and summarise information;
- The user will be free to interrupt the interview at any point and select another node in the hierarchy for continuation.

We have already explained how PCT achieves the last of these and how it flexibly combines laddering with direct graphical editing of the model diagram. As an illustration of the other characteristics, Appendix 10 lists a dialogue interaction which develops a model similar to the one shown in Figure 5.17. The default dialogue strategy which generates this interaction was evolved over extensive formative trials. It uses the following main techniques:

- The model is built by two top-down passes over the domain. In the first pass (lines 1-26 in Appendix 10) only the hierarchy of classes and not feature information is elicited. In the second pass (lines 27-88) feature information is elicited, with the opportunity also to extend

the hierarchy.[16] Users mainly preferred this strategy over an alternative strategy which elicited class names together with feature information in a single pass, perhaps because with our strategy the first pass rather rapidly constructs the model diagram which then serves as an aid to the user in identifying features in the second pass.

- In both passes the tree is traversed depth-first. This was mainly preferred to a breadth-first strategy because with breadth-first elicitation, consecutive questions may relate to quite different parts of the domain, making the dialogue seem less coherent.
- Before eliciting feature information for a class, the system asks whether information that would normally be inherited is valid (e.g. lines 43, 52). This anticipates the possibility that the class is an exception; if it is, then the system will elicit the details. Another possibility is that the user has not recognised the implications of what he or she has said already. By presenting the information at this point, the user may decide on reflection to edit the features of an ancestor class, either directly with the pen tool or by clicking on the appropriate node with the interview tool to restart laddering from that node.
- After eliciting feature information for a class, the system generates a summary (e.g. lines 39, 47). This provides an opportunity for the user to revise answers and hopefully also stimulates reflection on the validity of what has been entered thus far.
- When the system detects an instance of ambiguity (i.e. after applying Touretzky's rule, an attribute still has more than one value) it draws attention to this and lets the user decide how to proceed (e.g. line 78). Again the aim is to stimulate reflection. As noted above, the system does not insist that the ambiguity be resolved.

---

[16]Two trigger conditions are used to detect when the system should terminate the first pass: the user either declines further to elaborate the class hierarchy, or the user interrupts the interview. Observations suggested that the second condition may be too weak. Experienced users could perhaps be given the option to switch manually between the two passes.

We believe that these features of the dialogue strategy are largely novel to our tool. Other features, such as the way in which the system uses its existing knowledge of attributes to guide further elicitation (e.g. lines 63, 65), are shared with ALTO.

The main problem that emerged during the formative trials concerned the syntax adopted for feature information. The normal syntax of a feature is a formula of the form `Attribute = Value`. Not only did this syntax cause some difficulty but users often seemed to struggle to find attribute and value representations that expressed what they wanted to say. To mitigate the difficulties, we implemented the following measures:

- There are no syntax restrictions on what can serve as attribute and value symbols — (almost) any string of characters will be admitted. For example, `size of the fish = 3lb or more` will be recognised as a valid formula.
- The equals sign can be omitted. For example, `size is 3lb or more` will be parsed equivalently to `size is 3lb or more = true`.
- The dialogue boxes into which feature information is entered contain Help buttons which offer syntax reminders.
- Previously defined attributes and values are selectable from menus rather than requiring to be recalled and retyped.

More radically, features can be composed of symbols selected from menus filled by a prestored lexicon. This possibility, which is described in the following chapter, eliminates the difficulty to a considerable extent. However, prestored lexicons were not used for the main school trials because their implications are quite far reaching. Also, unless all the modelling tools were to have this facility, the validity of comparisons between different tools could have been undermined.

It should be noted that PCT does not use all the forms of question that are associated with laddering. For example, it never

asks questions of the form 'What is X an example of?'. The ability to restart the dialogue from any node largely obviates the need for this question. In general, formative tests suggested that our dialogue strategy in its present form is sufficiently powerful to elicit complete models.

In the previous chapter (section 4.3.1) we characterised modelling with the contrived techniques using two hypothetical and extreme scenarios. In the first scenario the system asks questions that sustain the interest of the modeller, the model's state is continuously and visibly updated, and the modeller can switch from laddering to direct editing whenever he or she chooses. In the second the system's questions are unhelpful, the model is not visible, and the modeller is not permitted to take control. We claim that PCT lies much closer to the first scenario than to the second. However, it is quite possible that further research could improve upon the present dialogue strategy. To assist such research, and also to provide some flexibility for users, the tool has been implemented with switchable settings for some of the main dialogue parameters, including depth-first versus breadth-first traversal and two-pass versus one-pass model construction.

5.3.2 Linking classification trees

Figure 5.19 below illustrates how classification trees can be linked. The trees on the left appear in different windows. The label T1 of a node in the upper tree appears at the root of the lower tree. In logical terms, the two trees together are equivalent to the tree on the right. Unlike the case for decision trees and factor tables, this scheme makes no use of window names: class names suffice. Furthermore, it makes no difference whether T1 occurs on the upper tree as an interior node or a leaf node. For an interior node, the linked tree gives as the subtrees of T1 the set of subtrees which appear under T1 in either of the two original trees.

Ideally, perhaps, the opportunity to create a new subtree window should be provided as part of the laddering dialogue strategy. One way to do this would be to offer a 'new window' option in the dialogue box shown in Figure 5.13. PCT does not yet

implement such a feature. Instead we provide a tool (the sixth icon from top in Figure 5.12) which, when the user clicks on a node, automatically creates a new window that contains a single-node classification tree. The label of the single node is a copy of the label of the node that was clicked. To support multiple-window models, a hypertext-like browsing facility is provided that is similar to the facilities previously described for PDT and PFT.

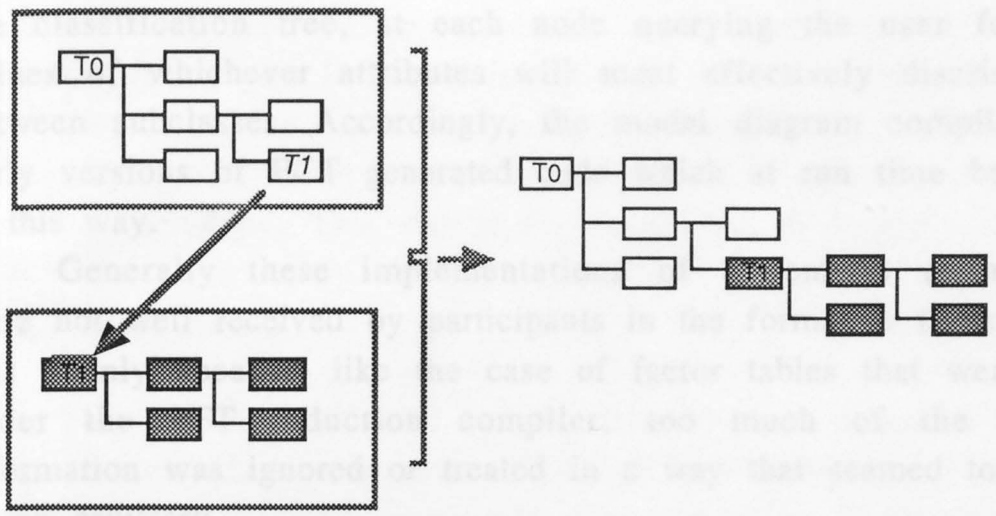


Figure 5.19 Linking classification trees

5.3.3 Interpreting classification trees

As explained in section 4.5.3, the approach consistently adopted in this research has been to compile the model into PKRL code that could be used as the knowledge base for a classification expert system. Accordingly, this approach was implemented in PCT. After clicking on the Run button in (say) Figure 5.12 the system will query the user about the features of a postage stamp until an appropriate categorisation can be reached, if one exists.

In the case of classification tree models, however, it is apparent that several other methods of model interpretation are possible. Three specific methods were identified as being potentially effective in a classroom context because they seemed likely to stimulate reflection, support tests and demonstrations, and enable the properties of the model to be explored. These methods, respectively concerning anomaly checking, feature reporting, and class comparison, were investigated and are

described below. However, we first provide a description of the Run function.

5.3.3.1 The Run function

The original specification had proposed that models would be interpreted by the classification method of systematic refinement (as described for example by Tansley and Hayball (1993)). Systematic refinement involves a top-down traversal of the classification tree, at each node querying the user for the values of whichever attributes will most effectively discriminate between subclasses. Accordingly, the model diagram compiler for early versions of PCT generated code which at run time behaved in this way.

Generally these implementations of systematic refinement were not well received by participants in the formative trials. This was mainly because, like the case of factor tables that were run under the PFT induction compiler, too much of the user's information was ignored or treated in a way that seemed to them to be arbitrary.

To illustrate the point, consider again the `Fishes` model shown in Figure 5.17. Our system's initial query would invite the user to select a value for `water type`. If the user gave the answer `fresh` then the system would ask for the value of an attribute (not shown in the figure) that would help to discriminate between `salmon`, `carp` and `trout`, and so on. Although this line of questioning is justified by the closed-world assumption, users found it disconcerting. They wondered why the information they had supplied for `Fishes` had been ignored, and why `water type` had been singled out among the attributes that had been entered for `sea fish` and `river fish`.

Teachers who were participants in a Master's level knowledge-based modelling course proposed an alternative strategy. In their strategy, the Run command initiated a bottom-up traversal of the tree with every feature of a class being checked until all were validated or one was falsified. Furthermore, to allow for the possibility that the class that the user was attempting to categorise is not present in the model, queries for

values included the options 'Something else' and 'Does not apply'. Following discussion, this approach was implemented and used in the large-scale trials.

The rejection of systematic refinement can be taken as further evidence against the use of opaque interpretation strategies. On reflection, however, it is not clear that the decision to abandon the method entirely was correct. It should be possible to implement a version of systematic refinement which does not ignore information and which offers the same 'open-world' options as those mentioned above. Such an implementation may be no more opaque in its lines of questioning than the implementation that was adopted, and it would have the advantage of exemplifying a widely applicable and efficient classification inference technique. This should be the subject of future research.

5.3.3.2 Anomaly checking

This function, accessed by the Check button shown in Figure 5.12, detects and warns about conditions that may indicate anomalies in the model. A sample report is shown in Figure 5.20. If a condition is triggered then the system makes a suggestion on how the anomaly — if the user accepts that it is one — could be removed.

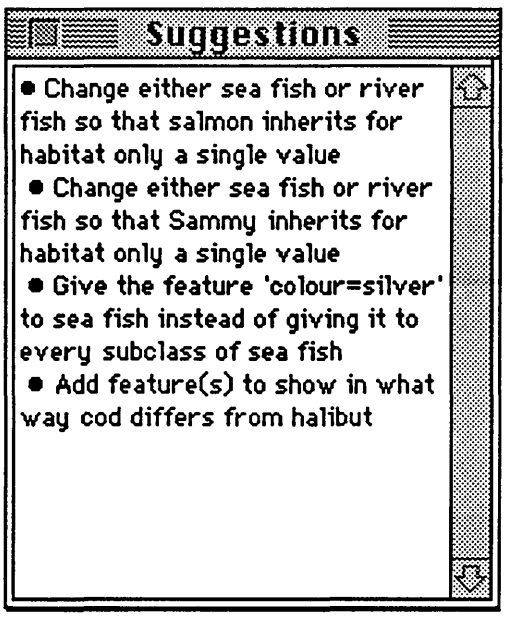


Figure 5.20 Anomaly checking in PCT

Table 5.1 shows the checks that are implemented along with the justification for each check. The Non-distinct Siblings check and the Feature Elevation check are also provided by the 'knowledge checking' function of ALTO: the other two checks are believed to be novel to PCT. We limit the number of suggestions that may be proposed by any use of the Check button to five. This reduces the time required to complete the computation to (typically) a few seconds and also prevents the user from possibly being overwhelmed by excessive output.

Check name	*Trigger condition*	*System suggestion*	*Justification*
Non-distinct Siblings	Sibling classes S1 and S2 have identical features.	Add or amend features to show in what way S1 and S2 are distinct.	Models should make explicit distinctions.
Featureless Subclass	A subclass S of C has no (locally defined) features.	Give S a feature to show in what way it is a special kind of C.	As above.
Feature Elevation	All subclasses of C define the same local feature A=V.	Remove the feature A=V from the subclasses and give it to C.	Models should be concise.
Inheritance Clash	A class C inherits two different values for the same attribute.	Edit C or the ancestor classes of C to remove the clash.	A clash of values may not be intended.

Table 5.1 Analyses that are implemented by the Check button

Our main aims for the Check function were to stimulate reflection on the user's part, help the user to catch errors and incompletenesses in the model, and improve the model's eventual runtime performance. Formative tests indicated that these aims were quite fully achieved. The reports which the button generates were typically studied attentively and users were often observed to make changes to their model immediately after reading one of these reports. Once they knew about it they tended to use the button regularly, even though no other function of the system depended on it.

The facility could be developed further: for instance, in the example shown in Figure 5.20 it may not be helpful to offer two suggestions that stem from the same source, as do the first two suggestions shown; and some automatic assistance might be

offered in implementing those suggestions that the user considers to be worthwhile.

5.3.3.3 Feature reporting

As noted earlier, this function is accessed by a tool which, when a node is clicked, reveals the node's feature information. An example was shown in Figure 5.14 above. This example illustrates the default 'Features only' setting of the dialogue. Two other settings can be selected from the pull-down menu: a 'Features & sources' setting additionally shows the class that is the source of each non-locally defined feature; and a 'Kind-of info' setting describes a class in terms of the local features which distinguish the class from its parent or parents. With the latter setting, clicking with with the tool on the river fish node in Figure 5.17 would obtain the description:

river fish is a special kind of Fishes that has water type = fresh and food = insects.

5.3.3.4 Class comparison

A natural question to ask about a classification tree is: How does class A compare with class B? The CMP tool (seventh from top in Figure 5.12) provides a way to answer questions of this kind. Clicking with this tool on (say) the `cod` node of Figure 5.17 produces a dialogue box like the one shown in Figure 5.21 below. The class to which `cod` is to be compared is selected from a pull-down menu.

We provide two methods of comparing classes. Both identify similarities and dissimilarities between the classes under comparison, but the techniques used are different. The first method, illustrated by the active `By features` button in Figure 5.21, is based on a comparison of features. First are shown features that are identical between the two classes; next, features in which attributes are shared but with different values; and finally, features in which attributes are unique to one or other class. This is the method that is recommended by Valley (1992)

and used in her EXPLORES (Explanation-oriented Expert System Shell) system.[17]

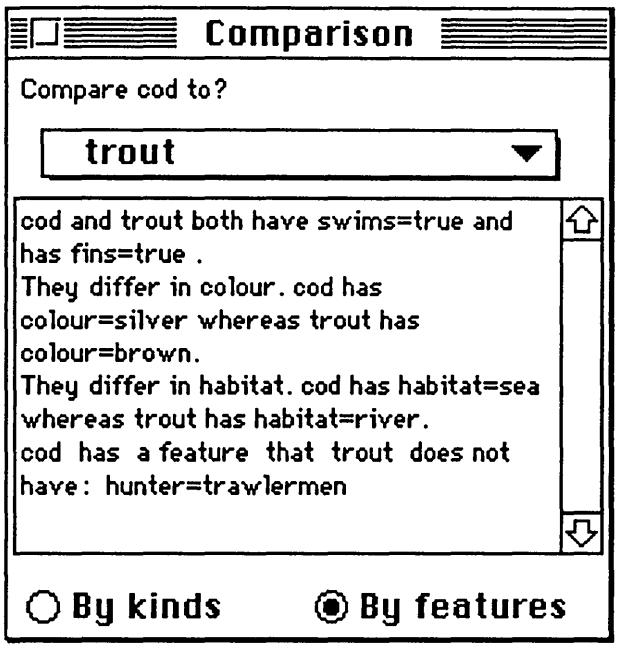


Figure 5.21. Comparing two classes by features.

Our second method, which we call `By kinds` and which is illustrated in Figure 5.22, is based on the relationship between the two classes in the overall class hierarchy. There are two main cases of `By kinds` comparison. In the case where the two classes are on different branches of the tree, the meet (i.e. the class identified by the node at which the branches join) of the two classes is computed. Each class is then described as a distinctive specialisation of this meet, either by referring to the ancestor of the class nearest the meet (as in the dialogue illustrated) or (if the meet is a parent of the class) by referring to the local features of the class. In the case where the two classes are on the same branch of the tree, one class is described as a specialisation of the other in essentially the same way. In our implementation, these cases are elaborated slightly to handle classes affected by multiple inheritance.

Formative trials suggested that both methods of comparing classes have advantages and disadvantages. The method based on exhaustive analysis of features gives fullest detail; sometimes, in

[17]EXPLORES will be discussed in Chapter 9 (section 9.4.1).

fact, more detail than users can readily comprehend. The method based on class hierarchy relationships gives an abstract comparison that seems to be easily understood, but lacks detail. PCT therefore implements both methods, but with `By kinds` as default, reflecting a mild overall preference that emerged from the formative trials.

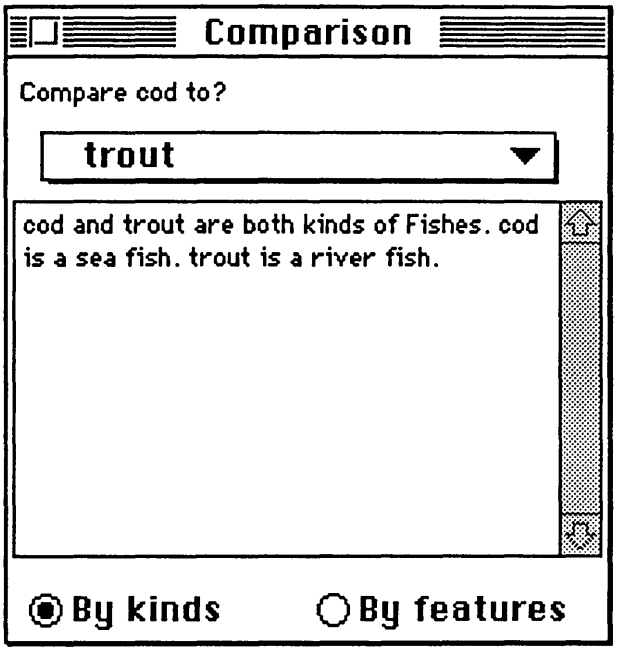


Figure 5.22 Comparing two classes by kinds.

5.4 Summary

The outline specifications and the design principles that were proposed in the previous chapter (sections 4.4 and 4.5 respectively) proved to be reasonably robust in the face of system analysis and implementation, although it is certainly true that the final shape of all three tools has been massively influenced by feedback from formative trials. Users' criticisms of runtime strategies which they perceived as opaque and arbitrary led to each of the two significant revisions that were made to the outline specifications, i.e. those relating to the use of induction for the decision tree tool and the use of systematic refinement for the classification tree tool.

All of the tools presented non-trivial detailed design issues. The classification tree tool presented some particularly interesting issues, such as how to combine laddering with direct model

editing, how to interpret inheritance, and how to design the dialogue strategy. If the example of this tool is any guide, then future research in the educational exploitation of 'contrived' knowledge acquisition techniques and hierarchical knowledge structures will present both challenging problems and rich possibilities.

Chapter 6 Analysing the quality of models

This chapter describes how models can be analysed automatically or semi-automatically. Analysis of a model is necessary to assess its quality: it is the means by which a model constructed with one kind of modelling tool might be compared to a model constructed with another. We define three specific measures which we associate with quality: correctness, efficiency, and conciseness. For research of the kind presented in this thesis, in which it is necessary to analyse hundreds of models, manual analysis is hardly practical. We show how a computer-based model analyser can be designed which is an effective research tool with potential for classroom use. Also discussed in this chapter is a second application for model analysis, namely model comparison. An analyser that can identify differences between different children's models, and explain these difference in a way that might stimulate discussion, could enhance the value of modelling tools. We discuss the design and evaluation of a prototype model comparator.

6.1 Measures of quality

In the next chapter of this thesis we present the results of an analysis of the quality of large numbers of Primex models. To make this analysis possible, we need to define the measures which we identify with quality: specifically, correctness, efficiency, and conciseness. This we do below, seeking definitions that ideally (but not quite in practice, as we will see) will satisfy four criteria. First, definitions should capture reasonably well the intuitive meaning which we hold for these measures. Second, definitions should have sufficient rigour to enable them to be unambiguously applied. Third, the same definitions should be applicable to any model, regardless of whether its representation is as a decision tree, factor table, classification tree, or rule set. Fourth, definitions should not depend upon the details of how models are compiled and interpreted. The first two of these criteria have obvious importance. The justification for the third and fourth is that we aim to compare different kinds of models, and would wish also

that our results and methods will be useful to researchers working with environments other than Primex. If definitions depend on specific features of our implementations, these aims are less likely to be achieved.

6.1.1 Correctness

In any programming or modelling system, it is frequently desirable to have some means of assessing whether a particular model is correct; informally speaking, to be able to check that the model gives the right answers. In most discussions of correctness in computer science, a crucial role is played by the notion of a *specification*. A specification of a software system provides the definitive statement of the inputs that are to be valid and the outputs that should be computed by each of these inputs. A program is correct or otherwise not in some abstract sense but only with respect to its specification, which is presumed to exist in advance of any validation of correctness.

Our programs are models of classification. As explained in Chapter 3, the model diagram compilers for PDT, PFT and PCT all compile models into the language of 'normal form'. Simple PKRL models can be compiled into this form also. In normal form, a model becomes a list of Prolog terms of the form

```
rule(Category, Features)
```

— where `Category` is an atom and `Features` is a list of pairs each of the form `Attribute=Value`. There may be many rules for any one `Category`. Suppose that a student's model, `SM` say, has a normal form representation consisting of j such rules:

```
SM = {rule(CSi,FSi): 1 ≤i ≤ j}
```

Given a case `I` to classify, consisting of say `n` features:

```
I = {A1=v1, A2=v2, ..., An=vn}
```

— where each `Ai=Vi` is a feature observation, and given a category `C` from those of `SM`, we define a relation:

```
classifies(SM,I,C)
```

— which holds if the model SM assigns I to a category C. In Prolog-like form,[1] the definition is:

```
classifies(SM,I,C) if
   member(rule(C,F),SM) and
   subset(F,I).
```

— where member and subset denote respectively the membership and subset relations on list representations of sets.

As noted above, a specification should define two things: valid inputs and input/output relationships. In principle we could provide a specification for the model SM in almost any language (English, Lisp, Prolog++, etc.). However, it will be convenient to require that the specification, like the model, is available in normal form. This is feasible because each valid classification case I and category C can be represented by a term rule(C,I) in the specification.

Suppose then that a model RM exists which serves as a specification or 'reference model' for SM. Let RM have k rules:

```
RM = {rule(CRi,FRi): 1 ≤ i ≤ k}
```

Now we can define a relation correct(SM,C,RM) meaning: the model SM defines a category C correctly according to the reference model RM. For correctness, the valid classification cases that categorise C in the reference model should be the same as the valid cases that categorise C in the student's model. We can express this in Prolog-like form as:

```
correct(SM,C,RM) if
   ∀ CRi,FRi,
        member(rule(CRi,FRi),RM) ⇒
        (classifies(RM,FRi,C) ⇔ classifies(SM,FRi,C)).
```

[1] 'Prolog-like form' is used in this chapter to present definitions in a semi-formal way. It is based on Horn clauses but may opportunistically contain elements of full first-order logic, set algebra, functional notation, etc., as demanded by clarity and convenience of expression. Translation into standard Prolog is straightforward.

— where '⇒' and '⇔' denote logical implication and equivalence respectively.

Usually we will be interested in the percentage of a reference model's categories that is correctly described by the student's model. However, it may be helpful to make a special case of the property of 'full correctness'. A model SM is fully correct according to a reference model RM if it correctly defines all of the categories of RM. We write:

```
fully_correct(SM,RM) if
   ∀ C,I, member(rule(C,I),RM) ⇒ correct(SM,C,RM).
```

Some observations on this approach to correctness follow.

(i) Correctness checking as described here is a purely syntactic operation. In practice, however, most students' models can be expected to use symbols some of which differ from those used in the reference model, even although the intended informal semantics of the symbols may be the same. Some form of preprocessing will be necessary to translate symbols used in the student's model into semantically equivalent symbols in the reference model. This is described later in this chapter.

(ii) Our definitions do not preclude the possibility that SM might contain unrecognised categories. Providing none of RM's categories are missing or buggy (see Figure 6.1) SM will still be regarded as fully correct. SM may be correct even if it contains additional, superfluous cases for some of the categories of RM. We argue that this is not a flaw in our view of correctness. Rather, it is analogous to the conventional situation for program correctness: no assurances are given about what a program will do if it is supplied with data that is outwith its specification. In our characterisation of correctness, the feature lists of the rules of RM define the data upon which validation of a model depends.

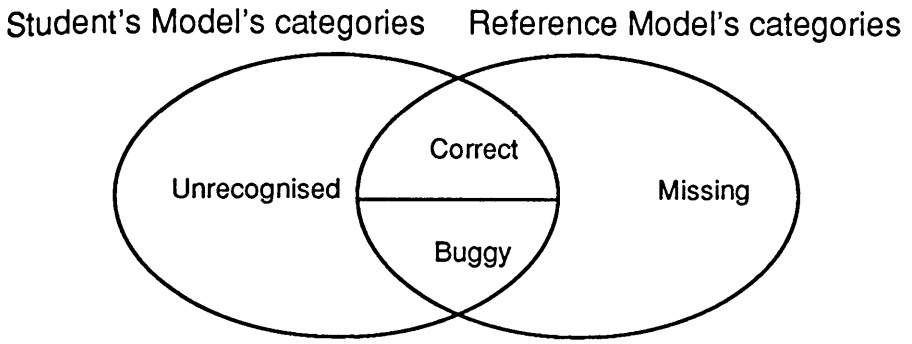


Figure 6.1 SM and RM categories

(iii) In some rule-based systems it is conventional to distinguish logical correctness (concerning what the program's logical reading implies) from procedural correctness (concerning what the program actually computes when it is run). In the case of Prolog, for instance, the runtime strategy is a depth-first search which may not be capable of computing all of the logically implied solutions, and so a program may be incorrect logically but correct procedurally. We ignore this distinction on the understanding that the runtime strategy for models is sound, complete, and guarantees termination, so that logical and procedural correctness are aligned.

6.1.2 Efficiency

Whereas correctness is a static measure of model quality, efficiency is a dynamic measure: it assesses how well a model performs at runtime. Conventionally, the efficiency of a computer program is related to its time and space complexity but these measures are not specific enough for our purposes. We want to assess how effectively a model functions as a classifier, from a discerning user's perspective. Given two fully correct classifiers, a discerning user might well regard as more efficient the classifier which asks fewer and better questions. To define 'better questions' is likely to be difficult so we propose to relate efficiency to the total number of questions which a system has to ask in order to categorise all of its cases over successive runs.

A procedural definition of efficiency can be offered as follows. Initially we restrict our attention to models that are fully

correct. Suppose that SM is a fully correct student's model for which exists a reference model, RM say, containing k rules:

$$RM = \{rule(CRi, FRi): 1 \leq i \leq k\}$$

We now run SM k times in succession. On the i-th run we provide the data FRi from RM (with a minimum quantity of any other data that is requested by SM) until the model identifies category CRi, which it is bound to do eventually because of the correctness assumption. Let SQ represent the total number of questions required to do this over all k runs. Now suppose that a second fully correct model PM exists for the same domain. Running PM k times in a similar way, suppose that this model is able to complete the runs in PQ questions. Then we define the efficiency of SM with respect to PM as PQ/SQ. Ideally we choose for PM a model that performs optimally so that this statistic is the lowest possible which can be obtained for SM.[2] For convenience, we will often multiply the result by 100 and refer to 'percent efficiency'.

In general, of course, a student's model may not be fully correct. For models other than fully correct models the description above requires a small adjustment. Efficiency is normally regarded as effort devoted to *useful* work and so we propose to restrict the efficiency measure to the correct part of SM. If SM defines correctly only a subset of the categories defined by RM, then compute SM', PM' and RM' as the subsets of SM, PM, and RM containing only rules for these correct categories. Now run SM' and PM' only on the data from RM' and compute the ratio of total questions asked as before. Of course, if a student's model defines no class correctly, its efficiency will be undefined.

It will be clear that efficiency as we define it here is not solely a property of one model. The selection of an optimally performing model for PM will be important. Also, the number of questions asked during a run is affected by the manner of the model's compilation and interpretation. Factor table models provide the most obvious illustration of this: when a user runs a

---

[2]We call such an optimal model PM a 'performance model'. We do not assume that PM must be the same as RM because a reference model need not perform optimally: it may be verbose and inefficient.

factor table with the PFT 'Smart' option enabled, the induction compiler may generate a normal form representation of the model that runs more efficiently than would have been possible with the 'Left to right' option (see the previous chapter for a discussion of this).

6.1.3 Conciseness

Informally, a concise model is one that is brief. Given a model in normal form, say

```
SM = {rule(CSi, FSi): 1 ≤ i ≤ j}
```

— possible measures of conciseness might be based on counting the number of rules or the number of characters within rules. However, the former measure seems too coarse-grained (rules may be few, but very lengthy) and the latter too fine-grained (we do not want to encourage single-character symbol names). As a compromise, we propose to count the number of 'phrases' in the model. By a phrase we mean a category name or feature, so that each term `rule(CSk,FSk)` contributes a quantity of phrases equal to the length of the list `FSk` plus one. As before, we propose to restrict the measure to the correct part of the student's model. If the correct part of `SM` contains `SP` phrases, and the correspondingly reduced part of a model `PM` contains `PP` phrases, then the conciseness of `SM` with respect to `PM` is defined to be `PP/SP`.[3]

An alternative notion of conciseness that is occasionally useful is economy of concepts. Generally, given two models that are equal in all respects including phrase count, but with one using a less diverse set of symbols, we might prefer the less diverse model on the grounds that the smaller symbol set reflects a less complex representation — one that is more 'economical' in its use of concepts. If the number of distinct symbols (which include category names, attribute names, and attribute values) appearing

[3]In practice, it is often possible to find a model of the domain which is both (approximately) optimally concise and efficient. Hence we can measure conciseness with respect to the same 'performance model' that is used to measure efficiency.

in the correct part of SM is SS, and the number appearing in the correspondingly reduced part of a performance model PM is PS, we define the economy of SM as PS/SS.

As before, we will often multiply these fractions by 100 and refer to 'percent conciseness' and 'percent economy'. We should note also that as with efficiency, these measures are not solely properties of the model. For instance, a model diagram compiler could eliminate some text in generating normal form.

6.1.4 Summary of quality measures

Table 6.1 provides a summary of the quality measures.

Measure	*Defined by*	*Notes*
Correctness	Percentage of a reference model's categories that is correctly described by the student's model.	The feature lists and categories of the reference model's rules specify the valid inputs and outputs.
Efficiency	Number of questions required to generate correctly defined categories of the student's model, expressed inversely as a percentage of the number required for the same categories with the corresponding part of a performance model.	Requires pre-analysis of correctness. Will be undefined if model is wholly incorrect. Implementation related.
Conciseness	Number of phrases in the correct part of the student's model, expressed inversely as a percentage of the number in the corresponding part of a performance model.	As above.
Economy	As for conciseness, but uses the number of distinct symbols rather than the number of phrases.	As above.

Table 6.1 A summary of the model quality measures.

Do these definitions meet all the criteria we identified at the beginning of section 6.1? Unfortunately they do not, but we consider that the shortfall is not disastrous. The definitions broadly capture the underlying intuitive notions and they are reasonably rigourous, so the first two criteria are met. By basing the measures on normal form, the third criterion is satisfied also:

the same definitions apply to all kinds of Primex model.[4] However, this success is partly at the expense of the fourth criterion which aims for implementation-independence. It is quite conceivable that other implementations of the modelling representations described in this research will not make use of normal form in our sense; those that do may generate and process it in different ways that affect the outcome of applying some of our measures. In the case of efficiency, however, it is difficult to see how implementation influences could be altogether excluded, without radically departing from our intuition about what constitutes efficiency in this context.

6.2 Design of a model analyser

In practice, manually applying the quality measures to a Primex model is not a trivial task. Even for a small model, a full and careful analysis is tedious and error prone and may take an hour or more of effort. For research of the kind presented in this thesis, in which it is necessary to analyse hundreds of models, this is not an attractive prospect. This section presents the design and evaluation of a computer-based model analyser which can greatly reduce the burden of model analysis.

6.2.1 Motivation and related research

Apart from serving the needs of research, there is another reason for our interest in the automation of model analysis. The quality of any model constructed in the classroom is potentially of interest to students and teachers. At present they generally rely on informal interpretations of quality, using manual inspection of the representation together with the feedback that can be gleaned when the model is tested. Although this may work well enough for small models, and in conditions when adequate time is available for the task, with larger models and less favourable conditions

[4]With the possible exception of some PKRL models that may be difficult to express in normal form, e.g. models containing conditions with multiple parameters. Fortunately, these are rare.

some significant properties of the model may not be noticed or understood.

Problems with students testing knowledge-based models have previously been identified by researchers. Webb (1993) reports that a large proportion of discussion is concerned with the modelling interface. Wideman & Owston (1993) report that when a model under test behaves unexpectedly, the cause is often a typographical slip which requires lengthy but conceptually unproductive debugging. If learners become able to construct bigger models — an aim of the present research — then these problems are likely to become more serious.

We propose that a computer-based model analyser could assist students and teachers in obtaining rapid, accurate and detailed feedback on the quality of models. Although such an analyser will not affect the usability of the modelling tools, it should enhance their usefulness.

In fact, automatic program analysis for educational purposes is an established field of research. Most effort has been devoted to analysing programs written in languages that have been prominent in the teaching of computer science, such as Pascal and Lisp. Although the context and the languages are different from ours, the motivation for research is similar:

> *Students spend a lot of time writing computer programs for which they get usually no feedback.* (Vanneste et al 1993 p250)

Progress in building analysers for computer science teaching languages has been difficult. The claims that have been made for current systems are rather modest. For example CAMUS (Vanneste et al 1995) is reported to be capable of analysing certain (e.g. non-recursive) types of program averaging 25 lines in length. The largest programs analysed by IDS (Lutz 1995) are 50 lines in length. Both CAMUS and IDS are analysers for programs written in Pascal.

There are no known previous attempts to apply automated analysis to knowledge-based modelling systems. However, we note that not only are there good reasons for doing so, as described above, but also that the technical difficulties may be less severe than has been the case for languages such as Pascal.

The Primex modelling tools, with their relatively simple representations, provide a useful opportunity to explore some of the possibilities.

6.2.2 Architecture of the PMA

We constructed an analyser (named PMA — Primex Model Analyser) which with some human assistance attempts to apply the quality measures to a Primex model. The PMA can accept as input any kind of Primex model. However, the analysis is not guaranteed to succeed in every case, as explained below.

The PMA is not fully automatic. Although a fully automatic analyser is an attractive prospect, we felt that our main priority should be to develop an analyser that could be used as a research tool. In this context the assumption that human input will be available is a realistic one which considerably simplifies the system's design. Furthermore, the experience of developing a semi-automatic analyser might provide a helpful basis for the design of a more fully automatic tool at some time in the future.

Appendix 11 illustrates the PMA user interface and describes a typical session. The PMA, which is implemented in Prolog++ as a Primex extension, has the architecture shown in Figure 6.2 below. In this diagram, boxes denote data, ovals denote processes, arrows identify inputs and outputs, and underscored words denote computed quality measures. Below we describe the system's data and processes and we outline their implementation in the PMA.

Student's model: This can be any kind of Primex model. The model should describe one of the domains covered by the Library of Reference Models (but it is the PMA's responsibility to discover which domain this is). The PMA actually processes the model in normal form. For a model constructed in PDT, PFT or PCT, normal form representation is automatically included in the model file format at the time the model is saved. For PKRL (i.e. rule) models,

an attempt is made to pre-parse heuristically the PKRL to normal form when the PMA first loads the model.[5]

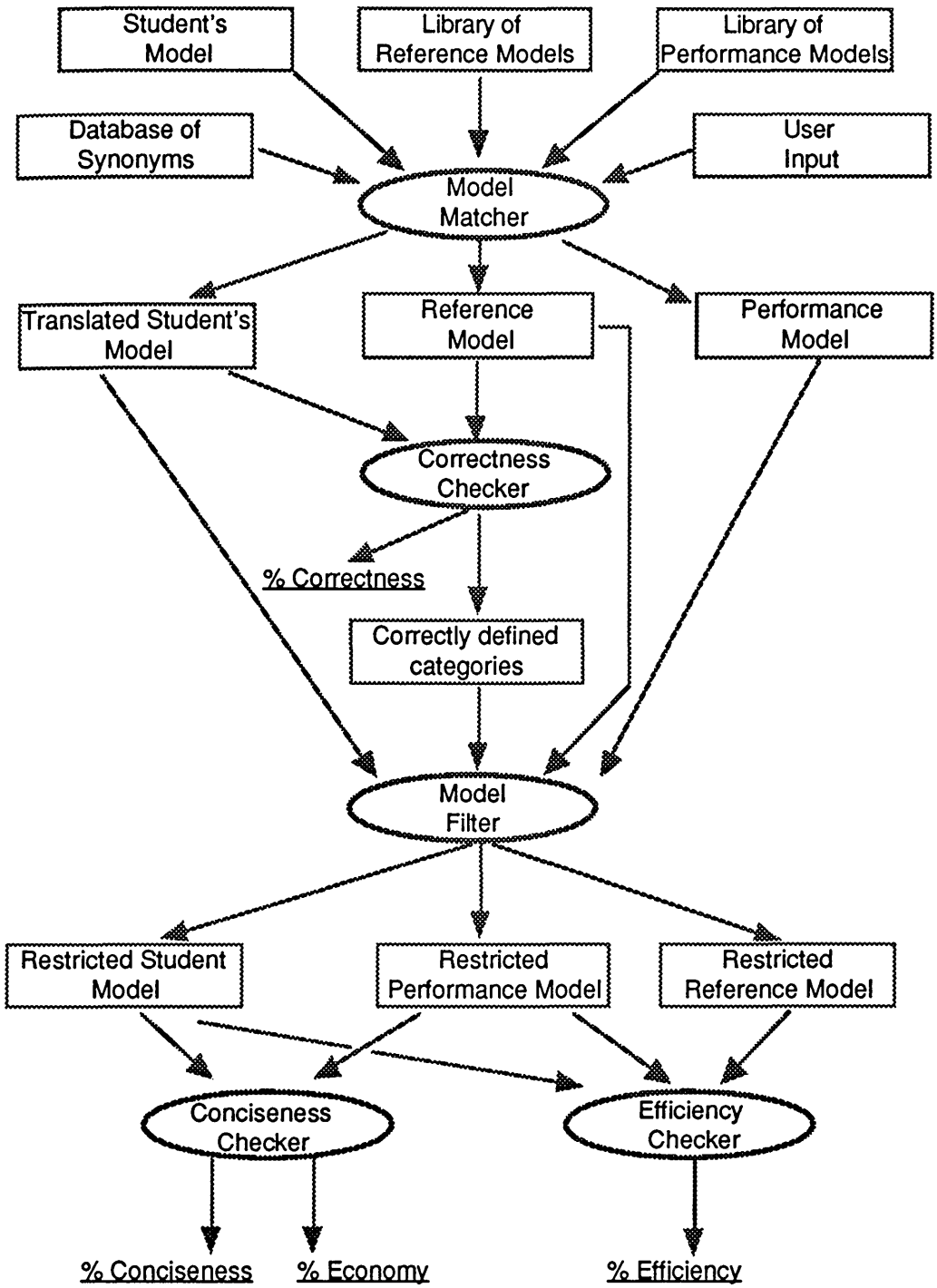


Figure 6.2 Architecture of the Primex Model Analyser

[5]The parse has to be heuristic because the PKRL is a first-order logic language in which conditions can express polyadic relations. The logic of normal form is much simpler, conditions being limited to binary relations of the form `Attribute=Value`. The parser assumes that the student will use only the propositional subset of the PKRL and will fail if this is not so. A PKRL proposition such as `The colour is black` will be parsed to `The colour is black=true`. Subsequently the Model Matcher should be able to translate this into `colour=black`.

Library of Reference Models (LRM): Each reference model should be an exhaustively full and definitively correct description of a domain. A domain can be represented by more than one reference model, each model using the filename convention `Domain - ExpN` where `Domain` is a name shared by all reference models for the same domain and `N` is an integer. A reference model may be constructed in any of the Primex knowledge acquisition tools (PDT, PFT, or PCT).

Library of Performance Models: These are expected to be 'optimal performance' models of their domains, as described earlier. They use the filename convention `Domain - performance`. It is not mandatory to supply a performance model for each domain: if none is available, the reference model is made to double as the performance model. This may be less than ideal because reference models will sometimes be verbose and inefficient. A performance model can be constructed in any of PDT, PFT, or PCT.

Database of Synonyms: This defines symbols that will be treated by the translator as synonyms for symbols appearing in the LRM. A synonym may be entered into the database as specific to named LRM models, or applicable to all LRM models.

Model Matcher: The goal of this process is to produce a Translated Student's Model which preserves the informal semantics of the Student's Model but contains only symbols common to one of the LRM models (i.e. the selected Reference Model). The process first identifies the categories described by the Student's model. These are then mapped where possible to the categories described by each of the models in the LRM, using a text-matching algorithm based on substring search: the algorithm recognises number and case variations and synonym equivalents. LRM models giving a poor correspondence are eliminated at this point. If none remain, the analysis fails immediately at this point. Otherwise, for remaining LRM models a mapping is now constructed between the features of rules in the Student's Model and those in the LRM models. This combines the same text-

matching algorithm as before with heuristics that recognise the equivalence of some common patterns of feature expression (for example, the feature `thick border=yes` is recognised as equivalent to `border=thick`). The LRM model with the most complete category and feature mappings is then designated as Reference Model. If this model's mappings contain gaps then the user is questioned to elicit directly the appropriate correspondences. In any event the user is expected to check that the system's mappings are valid. Sometimes, for example in cases where the student has used an idiosyncratic representation, the user's best course of action will be to edit directly the Student's Model prior to submitting (or resubmitting) the model to the Model Matcher: in the worst case the analysis may have to be abandoned. Assuming, however, that a complete and satisfactory set of mappings from the Student's Model to the Reference Model is obtained, these are then used to generate the Translated Student's Model. If a Performance Model is available for the domain of the Reference Model, this is identified; if none exists then the Reference Model is designated as the Performance Model also.

Correctness Checker: This process computes the correctness measure and the set of correctly defined categories. It uses an algorithm based directly on the definitions given in section 6.1.1.

Model Filter: This process obtains the subset of a model in which rules are eliminated other than those that refer to the correctly defined categories.

Efficiency Checker: This process computes the efficiency measure using the method described in section 6.1.2. To compute the number of questions that would be asked on a given run, a procedure is used which simulates the behaviour of the Primex runtime system. The simulation is non-interactive: the PMA operator is not required to answer any questions.

Conciseness Checker: This process computes the conciseness and economy measures described in section 6.1.3.

6.2.3 System performance

We evaluated the Primex Model Analyser with a view to answering the following questions:

- How effective is the PMA as a research tool?
- Would the PMA in its present form be usable by teachers and/or children?
- What if anything needs to be done to produce a more fully automatic tool?

These questions will be answered in the following section. This section describes the method of evaluation and the data that was obtained on the system's performance.

The method was as follows. Models produced by the main school trials (described in the next Chapter) were analysed with the PMA and, in the case of a sample of these models, we studied how the PMA performed. The sample consisted of 151 models representing the entire set of runnable models[6] produced by pupils within one of the trial schools. The PMA was configured with a Database of Synonyms (see above) containing 15 entries that had been developed for the domains during previous analyses. The sample of models included a variety of Primex model types and problem domains,[7] as shown in Table 6.2, and was considered to be reasonably representative of the models that were processed overall.

	PKRL	*PDT*	*PFT*	*PCT*	Total
Flags	12	13	12	0	37
Horses	12	18	15	0	45
Widgets	12	11	6	0	29
Darts	0	0	0	4	4
Boings	9	12	8	7	36
Total	45	54	41	11	151

Table 6.2 Model types (top row) and domains (left column) in the sample

[6]The school actually produced 162 models of which 11 were non-runnable due to serious syntax errors. The Primex compilers already check and generate reports for these errors. Instead of repeating these checks within the PMA, we adopted the design assumption (common in automated program analysis) that a model to be analysed is runnable.

[7]Domains will be described in the following chapter. Their specifications are given in Appendix 12.

As each model was analysed, data was gathered to show:

• Whether the PMA analysis succeeded or failed. For PDT, PFT, or PCT models, a failed analysis is invariably due to the Model Matcher's inability to generate an adequate set of symbolic mappings from the Student's Model to any Reference Model. For PKRL models, an additional possible source of failure is the pre-parse from PKRL to normal form.

• Where the analysis succeeded, the number of questions that were asked of the user by the PMA to elicit correspondences and whether or not the student's model required to be edited. Of special interest were cases where the analysis succeeded with no questions and without any editing, since these represent analyses that were fully automatically generated by the PMA.[8]

Taking the 151 model analyses together, the results were as follows. In 3% of cases the analysis failed. In 64% of cases it succeeded without any manual editing of models being done and in the remaining 33% of cases it succeeded with the aid of a manual edit. The average number of questions that was asked by the system during a successful (i.e. non-failing) analysis was 0.8. A fully automatic analysis (in the sense described above) was performed in 45% of cases.

Tables 6.3 and 6.4 shows the results grouped by domain and model type respectively. Note that 'Fully automatic' cases are a subset of the cases shown as 'Succeeded without editing', so the entries in rows 2-4 for each column total to 100%.[9] The first table shows that the PMA required more human assistance in some domains than in others, with best overall performance in the Horses domain. The second table shows that the system was less capable in analysing PKRL (rule-based) models than it was in analysing models developed with the new knowledge acquisition tools.

---

[8]But although there was no human intervention in these cases, the decision *not* to intervene (i.e. to accept without change the PMA's category and feature mappings) required human judgement.

[9]Ignoring rounding errors.

	Flags	*Horses*	*Widgets*	*Darts*	*Boings*
Number of models	37	45	29	4	36
Analysis failed	5%	2%	3%	0%	0%
Succeeded with editing	30%	4%	41%	50%	64%
Succeeded without editing	65%	93%	55%	50%	36%
(Fully automatic)	22%	80%	41%	0%	33%
Average no. of questions	2.2	0.1	0.9	1.0	0.2

Table 6.3 Results of PMA analysis on models grouped by domain

	PKRL	*PDT*	*PFT*	*PCT*
Number of models	45	54	41	11
Analysis failed	9%	0%	0%	0%
Succeeded with editing	47%	22%	34%	27%
Succeeded without editing	44%	78%	66%	73%
(Fully automatic)	33%	56%	44%	45%
Average no. of questions	1.2	0.8	0.5	0.5

Table 6.4 Results of PMA analysis on models grouped by type

It is not difficult to offer an explanation as to why the analysis of PKRL models may cause greater difficulty. As described above, the PMA processes models in normal form. For models constructed in PDT, PFT or PCT, normal form is directly available to the analyser whereas for PKRL models it must be specially generated. This extra process is heuristic and a source of possible error. An examination of individual cases confirms that translation from PKRL to normal form caused analysis to fail on some occasions. On other occasions the translation was done in an imperfect way which led to difficulties in the Model Matcher process.

It is less easy to explain why the analyser performed better in some domains than others, although model size and complexity are probably factors. Table 6.5 shows the mean number of total phrases and the mean number of distinct symbols occurring in models in the sample, grouped by domain (omitting the Darts domain since it is represented by very few models). We note that models in the Horses domain, which gave best PMA performance, are the smallest by either measure. Also, the Synonyms Database possibly gave more help to some domains than to others, thus

improving the likelihood of a fully automatic analysis within these domains; however this is not obvious from an inspection of the database contents.

	Flags	*Horses*	*Widgets*	*Boings*
Mean total phrases	38	18	47	21
Mean distinct symbols	21	13	31	17

Table 6.5 Mean size and complexity of models by domain

6.2.4 Assessment and future development

The questions asked at the start of the previous section can now be answered. As a research tool, the PMA in its present form is extremely useful. In a clear majority of cases in the sample discussed above, human intervention was limited to answering one question plus confirming the validity of the category and feature mappings. To analyse a model in such a case becomes the work of at most a few minutes. In the minority of cases which required a manual edit, this might be extended to 5-15 minutes (depending on the complexity of the model domains and the extent of the editing required). Failed analyses are rare. Hence the PMA offers the researcher a major saving of effort relative to manual analysis, which as noted previously can take an hour or more for a single model.

Although the PMA has not yet been used outside of a research context, it seems at least possible that the analyser in something close to its present form would be usable by teachers who were given suitable training. As the interaction described in Appendix 11 indicates, the skills involved in using the system are mostly of a linguistic and domain-related nature, including: ability to recognise semantic correspondence between different symbolic representations; ability to edit a model's normal form so as to enable successful matching with a reference model; and (perhaps) ability to implement new reference models, performance models, and synonyms, so as to extend the system for new domains.

Considering the skills involved, it is less clear that the PMA would be usable by children. The system's requests for user input are currently rather opaque. It is not obvious that children could be relied upon to validate the system's category and feature

mappings. No doubt with further research the system could be improved to mitigate these problems somewhat.

One possibility may be to limit children's use of the PMA to those cases of model comparison in which the system can be expected to operate in a more or less fully automatic way, for example to domains such as Horses and to non-PKRL model types. In our data there were 33 cases of non-PKRL models in the Horses domain. None of these required manual editing; 27 cases were processed fully automatically, and in the remaining 6 cases the system just asked one question per case. A problem could still arise in these cases in that children may not understand the system's question or be able correctly to reply. Furthermore, should an exceptional case be encountered in which the system's category or feature mappings were inappropriate, it seems doubtful that a child would be able to recognise the fact and undertake successfully the required manual edit. Finally, we note that in very simple domains manual checking is sufficiently undemanding to call into question the justification for using a computer analysis tool in the first place.

Ultimately, the limitations of automated model analysis can be related to the problem of understanding natural language. If we allow categories and features to be described in an unconstrained way then we must accept some risk that models will be hard to interpret, certainly when the interpretation is undertaken by a machine. One route towards a more fully automatic analysis may be to constrain users' freedom of expression: this is discussed in the next section. With unconstrained expression, it may never be possible to develop a fully automatic, infallible analyser, but it should at least be possible to improve upon the existing system. It may not be difficult to identify natural language processing (NLP) techniques which do better than the approach of synonyms and heuristic text matching that is presently used by the PMA's Model Matcher. An interesting challenge will be to find methods which can exploit NLP techniques whilst still making it possible for the system to be extended by non-NLP specialists so as to cater for models in new domains. This should be a matter for future research.

6.3 Design of a model comparator

A second application in which model analysis is necessary is in assisting the comparison between two children's models. As explained in Chapter 4 (section 4.5.2), we were influenced in the design of our tools by the idea that model building could be a collaborative activity. One way in which pupils could collaborate would be to work as a group in designing and building a model composed of several parts, taking advantage of the model linking capabilities of the new modelling tools. Another way, discussed in this section, would be for pupils to build two or more different models for the same domain, and then to compare and contrast the features of each model.

6.3.1 Rationale for model comparison

Although no previous investigation of the idea is known, it seems reasonable to suppose that the activity of comparing two different models could be educationally beneficial. Insofar as differences between models reflect differences of belief, to identify such differences could provide a powerful stimulus to discussion. Explanation and defence of a model feature may lead to negotiation about how the feature might be changed and revision of belief. Although this could happen also in the case where just one student has built a model, which is then examined and discussed by others, we would expect the process to have a more committed quality in the case where each participant is a model builder. Building a model implies making an explicit commitment to an interpretation of a domain. The exposure of differences may therefore lead naturally to both justification and critique.

Research in automated program analysis does not seem to have considered the possibility of student-to-student program comparison. The literature from that field almost invariably assumes that the goal is to assess a student program against some form of program specification provided by an expert: this is also the approach of the Primex Model Analyser, as discussed above. It might be argued that computer science programming classes —

which provide the usual context of use for automatic program analysers — deal mainly with deterministic and formal problems for which the existence of definitively correct solutions is normal. However as Haggith (1995) observes, in the real world absolute consensus is very rare and for many domains to which modelling has been applied outside of education, such as environmental planning, the notion of a definitively correct solution is not really applicable. Haggith argues that rather than trying to iron out inconsistencies between different models we would often do better to view them as representing distinct, but perhaps both legitimate, viewpoints.

However, any empirical investigation of these ideas will confront an immediate difficulty. In practice, it can be very difficult to identify the differences between two models. Like the task of manually validating a single model against a specification, manual approaches to comparison are likely to be error-prone and incomplete. We suggest that some form of machine assistance could be beneficial. In the next section we describe the design of a prototype system, the Primex Model Comparator (PMC), which we developed in order to explore the feasibility of giving machine support to children in comparing different knowledge bases.

6.3.2 Architecture of the PMC

In view the above considerations, in designing the Primex Model Comparator we adopted a more pluralist perspective than seemed appropriate for the PMA. The PMC makes no use of reference models. Its task is to identify differences between models, not to judge them to be correct or incorrect. Furthermore, because the intended users are children we aimed to design a system that would be fully automatic. As explained above, this implies a need to constrain the language which can be used to define a model. We will explain later how model builders are provided with a standard lexicon to ensure that the same symbols will represent the same concepts.

Like the PMA, the PMC was implemented in Prolog++ as a Primex extension. Figure 6.3 shows the system's architecture. As

can be seen, the design is considerably simpler than that of the PMA.

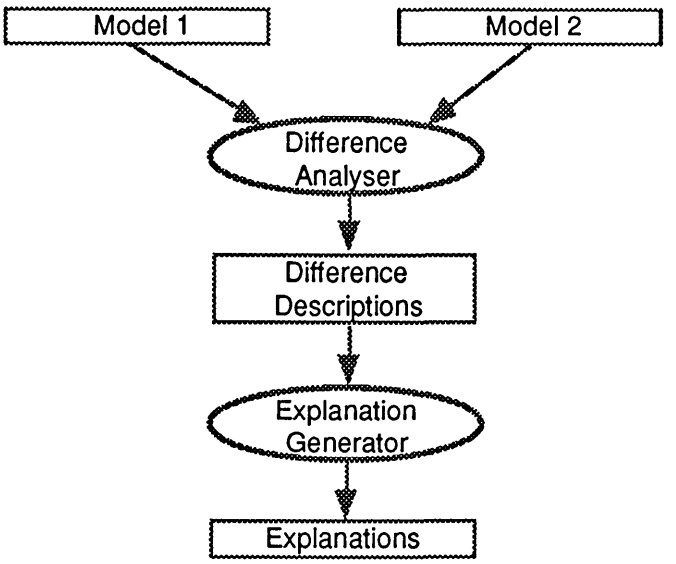


Figure 6.3. Architecture of the Primex Model Comparator

Below we describe the system's data and processes and we outline their implementation. Since a technical description of this system has been published previously (Conlon 1995), which includes a worked example and performance benchmarks, we restrict ourselves here to essential points. For an illustration of the PMC's user interface and an account of a typical session, see Appendix 13.

Model 1 and Model 2: These must be models constructed in PDT, PFT or PCT. Models can be compared that have been constructed with different tools. However, the PMC cannot accept PKRL models because these models do not provide normal form, as explained earlier.

Difference Analyser: This process generates differences between the two input models. The method is a variation upon that used by the PMA's Correctness Checker: in effect, each model is used as a reference model for the other. First, we define in Prolog-like form a `partition` relation to partition the classes defined by the two models, as follows:

```
partition(Model1,Model2,S1,S2,S3) if
   findall(C, member(rule(C,_),Model1), Classes1) and
   findall(C, member(rule(C,_),Model2), Classes2) and
   findall(C, member(C,Classes1-Classes2), S1) and
   findall(C, member(C,Classes2-Classes1), S2) and
   findall(C, member(C,Classes1∩Classes2), S3).
```

— where `findall` has its usual Prolog meaning and '-' and '∩' denote the set complement and set intersection operations on lists. No further work is done on `S1` and `S2` (the non-shared classes). The interesting differences stem from the classes `S3` that are common to both models. For each class `C` in `S3`, we look for a test case `I` that classifies `C` in one model but not in the other. We do this by defining a relation

```
difference(Model1,Model2,S3,Diff)
```

– meaning: `Diff` represents a difference between the classifications made by `Model1` and `Model2` for an element of `S3`. The definition, which uses the `classifies` relation defined in section 6.1.1, is composed of two symmetrical clauses as follows:

```
difference(Model1, Model2, S3, Diff) if
   member(C,S3) and
   test_case(C,Model1,Model2,I) and
   classifies(Model1,I,C) and
   not classifies(Model2,I,C) and
   Diff = diff(I,C,truein(Model1),falsein(Model2)).

difference(Model1, Model2, S3, Diff) if
   member(C,S3) and
   test_case(C,Model1,Model2,I) and
   classifies(Model2,I,C) and
   not classifies(Model1,I,C) and
   Diff = diff(I,C,truein(Model2),falsein(Model1)).
```

The meaning of `test_case(C,Model1,Model2,I)` is: `I` is the body of some rule for the category `C` in `Model1` or `Model2`. In Prolog-like form, the definition is:

```
test_case(C,Model1,Model2,I) if
   member(rule(C,I),Model1∪Model2).
```

— where ∪ is the set union operation. We arrange that the output of the Difference Analyser is a list of `diff/4` terms of the type shown in the `difference` clauses.

Explanation Generator: This process produces the text for the PMC's explanation dialogues (see Appendix 13). Categories from the classes `S1`, `S2`, and `S3` are presented as 'Extra conclusions', 'Missing conclusions' and 'Inconsistent conclusions'. If the user seeks an explanation of a category `C` in the latter class, the difference descriptions for `C` will be retrieved. A term of the form

```
diff(I,C,truein(ModelA),falsein(ModelB))
```

will be explained as:

There is disagreement about when decision C is correct.
In the case when I
ModelA says it is correct
but ModelB does not.

6.3.3 Constraining models to a lexicon

As stated above, the PMC is fully automatic. Model analysis without human intervention is feasible under the assumption that the models to be compared will represent the same concepts by the same symbols. In this section we describe how the new modelling tools PDT, PFT and PCT provide a standard method for constraining to a preconstructed lexicon the language that may be used for model construction.

Figure 6.4 illustrates how the user selects such a preconstructed lexicon. To generate a new model window, each tool provides a dialogue box like the one shown.[10] Normally the user clicks 'Ok'. However, the dialogue also offers a 'Based on...' button which, if clicked, enables an existing model file to be selected via a standard file dialogue. The symbol set from the selected model is then abstracted from the file and this becomes the lexicon (or lexical basis) for the new model. Edit fields which

[10]The one shown is for PFT but the same buttons are available in the corresponding PDT and PCT dialogue boxes.

would normally appear in model construction dialogues are replaced by menus prefilled with symbols from the lexicon.

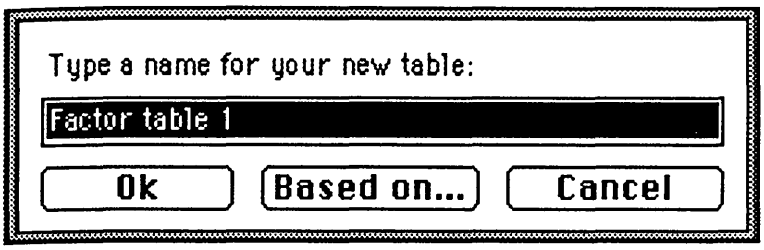


Figure 6.4 Dialogue to create a new PFT window

This method is implemented consistently across all the new tools. Any kind of model can provide the lexical basis for any other kind of model: for example, a classification tree model constructed through a laddering session might be based on the lexicon of a model previously constructed as a decision tree. An exception are models constructed in the PKRL. The abstraction of a model file's symbol set depends on the availability of its normal form representation. As noted earlier, this is straightforward for files constructed with the new modelling tools but not for PKRL models.

In designing this facility we were unsure as to how it might affect the student's experience of model building. Obviously, use of the 'Based on' button implies a loss of freedom: the student becomes unable to describe the domain in his or her own terms. More positively however, the provision of a lexicon might be seen as a form of scaffolding (Collins, Brown & Newman 1989). By providing names that are useful for describing the domain, the teacher carries out a part of the task that the student would find difficult. The student also benefits from the elimination of typing errors. However, the strength of these effects was far from clear.

6.3.4 Classroom experiments

We conducted two small-scale classroom experiments to formatively evaluate the prototype Primex Model Comparator. The experiments were conducted in two two-hour sessions held on consecutive days with a class of 19 S3 pupils in a state

comprehensive school.[11] On the day prior to these sessions, the class teacher had provided an introduction to the PDT tool and pupils each constructed a small decision tree model. This had been their first encounter with Primex.

6.3.4.1 Experiment 1: Using the PMC for debugging

For this experiment, the hypothesis was that the PMC would assist children to debug faulty models. The method was as follows. Children were introduced to the idea that a model might have faults or bugs relative to an independent specification. They were shown three methods of identifying bugs: testing, visual inspection, and the use of the PMC in conjunction with a known correct model. A number of debugging tasks were then set. Children were provided with some specifications and model files and they were told that some of the model files contained bugs, which they were invited to locate and repair using any of the three methods. Following some initial practice tasks, children were divided randomly into two groups and were asked to attempt four further tasks, with one qualification: use of the PMC was permitted only on alternate tasks. The task sequences were as follows:

```
Group 1: 1+PMC, 2-PMC, 3+PMC, 4-PMC
Group 2: 1-PMC, 2+PMC, 3-PMC, 4+PMC
```

That is, Group 1 first attempted Task 1 with the aid of the PMC[12] followed by Task 2 in which only the non-PMC debugging methods were permitted, and so on. For any task, having identified bugs children were asked to repair these and save the resulting model to disk. Repairs were undertaken by free editing. There was no use of the 'Based on' facility.

---

[11]The children had just entered Third year of Secondary school and were mostly fourteen years of age. They were in the first month of a course of Computing Studies at Credit level of Standard Grade. The school classroom in which the experiments took place was equipped with 20 PowerPC-based Macintosh computers onto which Primex (with all extensions) had been installed.

[12]If they wished to use it: it was open to children to tackle these tasks only with testing or visual inspection.

For illustration, two of the tasks are shown in Appendix 14. Children recorded the start and stop time for each task and saved repaired models to disk. In the event, most pupils managed to complete only three tasks. Some data was lost when a small number of pupils' model files from completed tasks could not be located on the school's hard disk after the session. Files from models that were successfully retrieved were analysed with the PMA.

The results of this experiment are summarised in Table 6.6. Children's performance on all tasks was unexpectedly good: the vast majority of models were fully repaired, regardless of whether or not the PMC was used. The overall mean time for repairs was slightly less for tasks that made use of the PMC (5.6 minutes versus 5.1 minutes) but the difference is not statistically significant. Therefore the hypothesis is not validated by this data. Considering the small numbers involved and the ceiling-level task performance, this is not very surprising.

Task	*1+*	*1-*	*2+*	*2-*	*3+*	*3-*	*4+*	*4-*
No. models	9	9	8	9	6	5	2	6
No. fully repaired	9	9	8	8	6	5	2	6
Fully repaired %	100%	100%	100%	89%	100%	100%	100%	100%
Mean time (mins)	2.7	4.2	5.8	4.9	7.8	8.0	5.0	6.5
Standard dev. time	1.8	2.2	2.6	1.6	2.6	2.1	N/A	1.8

Table 6.6. Results of debugging tasks with (+) and without (-) the PMC

Informal observation did however suggest that children found the PMC useful. No child was seen to avoid using the tool in a task where its use was permitted. In many cases they were observed to obtain a PMC analysis of a model *before* consulting the model's intended specification: evidently the PMC output enabled rapid pinpointing of the difference between the two. Children gave much less attention to the PMC's detailed explanations and at least one child volubly expressed dismay at her inability to make sense of these.

6.3.4.2 Experiment 2: Effect of 'Based-on' on model size

In the second experiment we investigated the effect of the 'Based on' facility on children's model building. The hypothesis

was that the facility would enable the construction of larger models. First, children were shown how to use the PCT tool to construct a classification tree model, using both laddering and direct graphical editing. They were then divided randomly into two groups: a 'Free-edit' group and a 'Based-on' group. Children in the Free-edit group were invited each to build a classification tree model for a domain of vehicles, with complete freedom to describe the domain in their own terms. Children in the Based-on group were similarly asked to build vehicles models, but these children were told to select the 'Based-on' button (see Figure 6.4 above) so as to base their models on a model provided by the researcher. The researcher's model contained 94 total phrases and 56 distinct symbols. At the end of the session, children's models were gathered for analysis.

Unfortunately, again not all models could be recovered from the school's hard disks and other models were later found to have been saved incorrectly.[13] The results are summarised in Table 6.7. Although the models of the Based-on group were larger on average than those of the Free-edit group, the difference is not statistically significant. This is unsurprising given the small numbers involved.

	Free-edit group	*Based-on group*
No. models	5	6
Mean no. total phrases	49	67
Mean no. distinct symbols	26	29
SD total phrases	26	25
SD distinct symbols	11	8

Table 6.7 Free-edit versus Based-on model building

Although the quantitative data does not validate the hypothesis, observation of the experiment did suggest some interesting results. Modellers in the Free-edit group experienced evident difficulties in entering features in the form required by the edit fields of the laddering dialogues, not just because of the formal syntax (i.e. `Attribute=Value`) but also because they found it hard to think of suitable names for attributes and values. These difficulties hardly applied to the Based-on group, for whom the

[13] See Technical Note 2 of Appendix 6.

laddering dialogues elicited feature information by means of prefilled popup menus.

However, a flaw in these dialogues became apparent. Figure 6.5 shows how such a dialogue was *intended* to appear: unfortunately in the classroom trial the menu on the right actually showed the combined value sets for all attributes, not just those for the attribute currently selected in the menu on the left. Pupils were therefore required to make a potentially confusing selection from a lengthy menu containing mostly unrelated items.

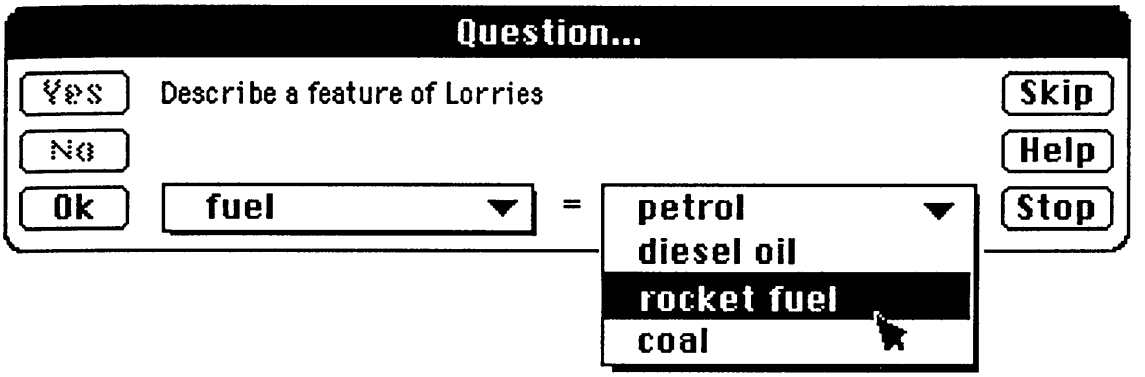


Figure 6.5 Selecting feature information from popup menus

Another difficulty affecting the Based-on pupils was that they did not know the convention of shift-clicking to select multiple menu items from a scrolling menu. As a result, faced with a laddering dialogue like the one shown in Figure 6.6 pupils would typically identify only a single subclass instead of several. To provide the missing information they would eventually have to interrupt the laddering interview and restart with the Q/A tool on the parent class node.

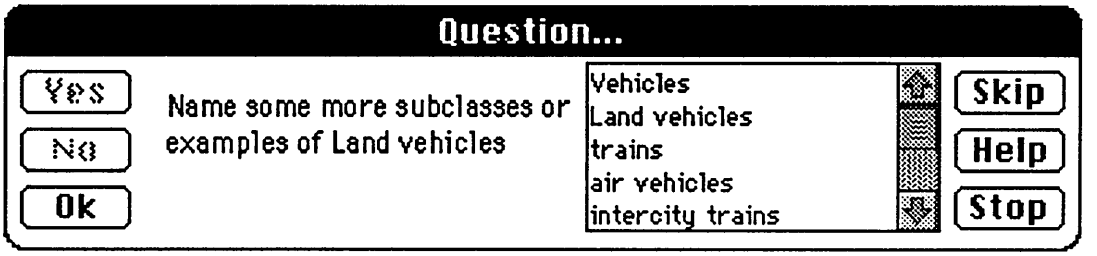


Figure 6.6 A scrolling menu where shift-clicking is useful

6.3.5 Assessment and future development

The idea of an analyser that can identify differences between different children's models, and explain these difference in a way that might stimulate discussion leading perhaps to a change in belief, is an attractive one which we have explored in only a very limited way. More research will be necessary to produce an analyser that is good enough to enable the teaching and learning implications to be properly investigated. Our experiences with the prototype PMC tool in small-scale classroom trials has clarified some main issues that should be addressed.

An important question concerns the extent to which it is justified to constrain the model-builder to a preconstructed lexicon. Such a constraint makes automatic difference analysis rather straightforward, as we have shown, and our classroom observations did suggest that an imported lexicon could act as a form of scaffolding. On the other hand, it might be argued that with such a lexicon the model builder is given rather too much of the solution. Relieved of the requirement to describe the world in personal terms, the model builder's task becomes more like assembling someone else's jigsaw. Furthermore, Haggith's argument for tolerance of a pluralist interpretation of domains (Haggith 1995, cited previously) seems hardly consistent in spirit with an externally imposed curtailment of what the model builder is permitted to express.

If teachers are content for some purposes with lexically constrained modelling then for those purposes an evolutionary refinement of the PMC may suffice. The design has considerable benefits: it is fully automatic, it needs no reference models nor other forms of domain knowledge, and it is robust.

Without lexical constraint it seems highly unlikely that these qualities can be equalled. As with the PMA, the choice may be between a willingness to invoke human assistance in identifying correspondences between models on the one hand, and settling for very simple domains but with more fully automatic functioning on the other. Natural language processing techniques may be available which can contribute to a system that is more

autonomous in larger domains. If the favoured choice is a design which partly relies on human assistance then a challenge will be to design questions that children can understand and answer.

Another issue for further research is in the explanation of differences that any comparator might discover. We detected signs that the PMC's explanations were not fully satisfactory. Future research might investigate alternative methods, perhaps referring to the source model representations. Some of the ideas of explanation generation reviewed by Valley (1992) should be useful.

6.4 Summary

This chapter has defined measures of model quality which, crucially, can be applied to any model, regardless of whether its representation is as a decision tree, factor table, classification tree, or rule set. We have shown that semi-automatic application of these measures is both desirable and feasible. In fact, when compared to the difficulties uncovered by researchers in the automated program analysis community with languages such as Pascal and Lisp, the analysis of Primex models is rather straightforward. We have demonstrated that our analyser, the PMA, is highly useful as a research tool and there is clear potential for classroom use of such tools by teachers.

An important issue has been the degree to which model analysis should be automatic. In our second application for model analysis, namely model comparison, we aimed for a fully automatic system in recognition of the fact that the tool was to be used by children. We achieved this objective, but at a cost: models must use the same symbols to denote the same concepts. We have shown how in practice a model builder can be provided with a preconstructed lexicon which constrains their representation. It is a matter for future research to say how far this solution is acceptable.

Finally, the chapter has established one clear advantage of the new knowledge acquisition modelling tools. By comparison to rule-based models constructed in the PKRL, models developed with PDT, PFT and PCT are easier to analyse. The explanation for

this is that the new knowledge acquisition tools clearly expose and distinguish the information that is needed to characterise a classification model. Information about categories, attributes, and values is explicit in each model and indeed is captured by the normal form representation that is part of the model file format. In a PKRL model on the other hand this information may be embedded in the rule set in a way which cannot easily be extracted, reflecting the fact that the PKRL is a general-purpose rule language and not one that has been specialised for classification tasks.

Chapter 7 The school trials

This chapter covers the main empirical results obtained from the large-scale school trials. We describe the design of a modelling course which involved children in building models for a variety of domains. The modelling course produced 632 models representing the work of 82 pupils from five secondary schools. These models were analysed using the techniques described in Chapter 6. Since each child typically built two or more models for the same domain, using different tools, we were able to use the results of the analysis to compare the effectiveness of the established rule-based shell with that of the new knowledge acquisition tools. Other data came from children's recorded times for model building, teacher questionnaires, pupil questionnaires, pre-tests and post-tests. We use this data to test four hypotheses relating to model quality, build times, motivation and representational skill.

7.1 Research strategy

As explained previously, the main hypothesis investigated by this research is that knowledge acquisition systems will enable more successful model building by children than has been possible with the EMYCIN-type shells. In this chapter we present relevant evidence from large-scale school trials of the modelling software. To motivate the presentation, it will be helpful at the outset to state four specific hypotheses that the school trials were intended to illuminate:

- Models built with the knowledge acquisition tools (PDT, PFT and PCT) are of higher quality than those built with the original rule-based shell (i.e. with the PKRL).

- Models are built by children more rapidly with PDT, PFT and PCT than with the PKRL.

- PDT, PFT and PCT motivate children to a greater extent than does the PKRL.

- As a result of working with our modelling tools, children develop their representational skills.

Following the presentation of the evidence, we shall return to evaluate these hypotheses in the final section of the chapter.

7.1.1 Selection of schools

In Chapter 3 we explained why it was considered appropriate to base the research method on large-scale teacher-led trials rather than tightly controlled experiments. Teacher-led trials seemed better suited to the exploratory nature of the research and also more likely to give feedback on how our systems would interact with real school contexts. The participation of several schools has benefits apart from the obvious one of potentially generating the large quantity of data which is needed in the present research. It also provides a form of triangulation whereby evidence from different sources can be compared. Furthermore, it offers some insurance against the risk (one that is difficult to quantify) that some data might be lost or for some reason be unusable.

Teacher-led trials cannot match the controlled conditions expected of laboratory-style experimental research. Accepting that, however, we certainly do not advocate a research approach that is laissez faire. Strong coordination is necessary to ensure that the conditions under which data is obtained are constrained to a reasonable degree. To achieve a high level of commonality, we adopted the following measures:

- All pupils followed the same modelling course, consisting of teacher exposition, demonstrations, worked examples, tasks, tests, and questionnaires. The course was written by the researcher and is described below. All schools were provided with a standard set of course materials.

- Pupils were drawn from the same stage of schooling and had

similar backgrounds in IT. Each class contained a broadly comparable range of pupils according to their teacher's predictions of attainment.[1] No pupil had previously used any of the new modelling tools, but some had a small amount of prior experience of the PKRL.

- Schools were invited to participate by the researcher, the main criterion being the presence of a teacher who was known to be sufficiently experienced and reliable to act as the course coordinator. All the invitees (identified in Table 7.1) occupied specialist IT roles in their respective schools.[2] All were familiar with the established rule-based Primex shell (i.e. with the PKRL). Teachers were extensively briefed by the researcher and it was felt that a high level of shared understanding was achieved on the aims and methods of the research.

School	*Location*	*Teacher in charge*
Balwearie High School	Fife	John Mason
Beeslack High School	Midlothian	Neil Livesey
Linlithgow Academy	West Lothian	Ann Cole
Musselburgh Grammar School	East Lothian	Luci Jones
Prestwick Academy	Strathclyde	Robert Grant
Trinity Academy	Edinburgh	Andy Pendry

Table 7.1. Participants in the large-scale trials

The use of IT specialist teachers had several advantages. An important one was that the researcher did not need to provide them with extensive individual training in the use of the new tools: teachers mainly taught themselves. Furthermore, it is very doubtful that teachers other than IT specialists could have justified the time (7-8 hours) required by pupils for the trial and neither would they have had ready access to the necessary computer systems. It is not claimed that these teachers are in some sense typical of teachers in general.

Neither is it claimed that the selected schools are

[1] All classes were at the mid-way point of a Standard Grade Computing Studies course. The vast majority of pupils were 15 years of age. Predicted attainment in all cases was for General or Credit levels (bands 1-4), i.e. the upper two of the three levels available for this course.

[2] All schools are state-run secondary comprehensive schools.

homogeneous. Notwithstanding our efforts to achieve a high level of commonality we anticipated that there might be differences in model building performance between schools, and indeed differences between pupils within each school, due to factors which we cannot control. To try to avoid effects arising from these differences, our experimental approach is to look for differences *within* pupils rather than *between* pupils. More specifically, our evidence is mostly based on patterns of differences that are found between a group of models built by the same child, using different modelling tools, for a given domain. Essentially, this style of experiment uses each subject as its own control. A possible disadvantage is the requirement to counteract order effects, as explained later.

Importantly, our research strategy attempted to marry the aim of situating the modelling activities in authentic school conditions with the aim of achieving a reasonable degree of experimental rigour. The marriage requires a willingness to compromise.

7.1.2 The modelling course

The modelling course had two main aims. First, it should provide a context for gathering the necessary experimental data. Second, it should provide children with an introduction to the representations and tools that would be worthwhile in its own right. The second aim was considerably important. Teachers were supportive of the research per se but their participation was stimulated by a shared belief that children would enjoy and benefit from the experience of the course.

Table 7.2 shows the stages of the modelling course. As can be seen, the course is structured around a series of domains. Two domains are used solely by the teacher for expository purposes and in each of a further six domains children generally build three or four models using different tools.[3] The domains are described in Appendix 12. The sequence in which tools are used within a domain is variable between children, as described shortly.

---

[3]The Horses and Darts domains were intended mainly as 'practice' domains for learning the tools rather than as sources of experimental data.

Stage	*Activity*	*Domain*	*Models*	*Size*	*Structure*
1	Pre-test	—	—		—
2	Teacher exposition	Car wash	PKRL, PDT, PFT	18/13	Flat
3	Pupil model building	Horses	PKRL, PDT, PFT	18/13	Flat
4	Pupil model building	Flags	PKRL, PDT, PFT	38/20	Flat
5	Pupil model building	Widgets	PKRL, PDT, PFT	40/33	Flat
6	Teacher exposition	Postage stamps	PCT	24/15	Hierarchical, singular inh., no exceptions
7	Pupil model building	Darts	PCT	27/17	Hierarchical, singular inh., no exceptions
8	Pupil model building	Boings	PKRL, PDT, PFT, PCT	31/22	Hierarchical, singular inh., no exceptions
9	Pupil model building	Pondoids	PKRL, PDT, PFT, PCT	31/22	Hierarchical, multiple inh., exceptions
10	Questionnaires	—	—	—	—
11	Post-test	—	—	—	—

Table 7.2 Stages of the modelling course

The course seeks to develop concepts progressively. Early domains have a flat structure and model building is limited to three representations whereas later domains have a hierarchical structure and model building includes all four representations. Models generally become larger, as shown by the fifth column of Table 7.2 which gives the total number of phrases/distinct symbols contained within an expert's performance model. In the final model, children are expected to describe a hierarchical structure which incorporates multiple inheritance with exceptions.

Some justification should be offered for the quantity and choice of domains. Classification domains are diverse and ideally a large variety of domains should be sampled. However, discussions with teachers had indicated that no more than eight hours would be available for the course in any school. Pilot tests had suggested that models for the chosen domains required 10-30 minutes for their construction. Not all of the course time would be occupied by pupils in model building and we calculated that the available time would allow for the construction by each pupil of no more than 18 models. Since our experimental method required that children

should construct several models for each domain (i.e. with different tools), it seemed that five or six domains would be the maximum that could be covered.

We designed domains to be varied, not only in size and structure but also in mode of specification and difficulty; some are specified in a mainly graphical way, others by text, and all were piloted to ensure that they presented tasks that were feasible in the time available. We avoided authentic curriculum topics and chose instead 'contrived' material. Authentic topics might have better represented future applications of the modelling tools, but their use would have complicated the research substantially because of the need to take account of varying levels of prior domain knowledge between pupils. Contrived domains have the advantage that they are new to all pupils. Also, contrived domains are easy to specify and assess: it is straightforward to construct reference models for them.

Arguably, in view of the results which we present later in the chapter, our domains were too simple; perhaps more challenging domains, if tackled by enough students, would have produced stronger experimental results. It is not clear however that harder domains would have been compatible with the aim of providing a worthwhile modelling course in the authentic school conditions of the trials. We claim that the domains used were justified by the limited time available for the course and with children who were novices in this form of modelling. Furthermore, our domains did give significant results, as we shall show.

Within each domain, the sequence in which the different tools were used was varied between children. For example, at stage 4 in which each child was expected to build three models in the Flags domain, one-third of children started with the PKRL, one-third with PDT, and one-third with PFT. This was achieved by printing the required sequence individually on each pupil task sheet in such a way that each of the six possible sequences was equally represented within the sheets dispatched to schools. These variations in sequence were intended to counterbalance any effect that might arise from the order in which models were built within domains. Should it transpire that (say) the first model built was likely to be more fully correct than subsequent models, then that

effect should not favour any particular modelling tool overall since each tool is used first with equal frequency.

7.1.3 Implementation and returns

Course materials were dispatched to schools in the first week of May 1996. As mentioned previously, teachers were supplied with a full package of course materials, including overhead projector slides, pupil handout and task sheets, prepared model files, and briefing notes. The latter are contained in Appendix 15. Teachers were asked to return to the researcher pupil's model files on floppy disks, together with the completed pre-tests, post-tests and questionnaires, by the end of the summer term.

During the period of the trial the researcher was in regular contact with the participating schools. Four of the six schools had email links: communication also used the postal service, telephone, and face-to-face meetings. The most serious problem that arose was that it became apparent that the course consumed more class time than had been anticipated. The problem was compounded by the fact that some teachers found their available class time squeezed by a variety of summer term activities. It was agreed that if cuts had to be made then in the first instance stages 9 and 5 should be dropped. In two schools more drastic cuts were made.

There was one major disappointment. During the trial the teacher coordinating the course at Beeslack High School left unexpectedly to take up a new position elsewhere. In consequence the course was abandoned at Beeslack. No data was forwarded from that school and the school does not feature in the analysis that follows. The other five schools all submitted data as arranged.

Table 7.3 shows the number of pupils participating and the model files returned from each school. In total 632 models were returned representing the work of 82 pupils, giving an overall mean of 7.7 models per pupil. This shows a considerable shortfall over the number of models that would be returned from a fully completed course, although there was considerable variation between schools, as the table shows. One school alone provided 39% of the total number of models returned. The table under-

represents the extent of pupils' model building insofar as not all models were successfully saved to and recovered from the computers of their schools. It is clear also that schools differed in the amount of time that they gave to the course.

School	*No. pupils*	*No. models*	*Mean models per pupil*	*Percentage of all models*
Musselburgh Grammar	12	124	10.3	20%
Balwearie High	20	248	12.4	39%
Linlithgow Academy	23	62	2.7	10%
Prestwick Academy	9	36	4.0	6%
Trinity Academy	18	162	9.0	26%
Total	82	632	7.7	100%

Table 7.3 Participation of pupils and return of models

7.2 Results of model analysis

Of the 632 returned models, 568 were successfully analysed. Almost all of these analyses were completed with the Primex Model Analyser (described in the previous chapter) but a small number required manual analysis where the PMA was unable to interpret the model. The remaining 64 models were rejected as unsuitable for analysis.

7.2.1 Rejected models

We do not regard the quantity of rejected models (representing 10.1% of the returns) as excessive. Models were not rejected lightly or on an ad hoc basis but because the quality measures could not (confidently) be applied to them, whether by human or machine. Rather than risking an unsound analysis we preferred to remove such problematic models from the sample. There were six categories of rejected model: those that were irrelevant, duplicate, grossly incomplete, contained syntax errors, were inaccessible, or were excessively idiosyncratic. These categories are described below.

1. Irrelevant models referred to domains that were never intended for analysis, such as the Car Wash domain that was used for expository purposes at stage 2 of the modelling

course.

2. Duplicate models were those in which the same student supplied two or more files with different file names but identical file contents: these were filtered so that only one copy remained.
3. Grossly incomplete models were those that clearly represented unfinished work on the student's part: deciding on this required judgement that made use of (but did not wholly rely upon) the start time/stop time information supplied by students.
4. Models containing syntax errors were those that would be rejected by the appropriate compiler. It was tempting to 'repair' models which contained only slight syntax errors, but the repaired model is no longer the student's own work and may not reflect his or her own intentions. In any case it is not always clear whether a particular error is 'slight'. Therefore it was decided that the presence of any syntax error would cause a model to be rejected. Special cases were 16 factor tables with incorrectly formatted columns (the rightmost and leftmost columns were transposed). Although almost all of these models were otherwise correct, their runtime behaviour was eccentric. We decided to treat this type of error (which we anticipated in section 5.2.1.1) as a syntax error and accordingly rejected the models. Had we not done so, we would have been forced to choose between overlooking the error as a slip (and applying the correctness measure to the models as if they were properly constructed) or assigning a correctness value of zero to all of them: we regard both these choices as too extreme.
5. Inaccessible models were those affected by disk media errors or incorrect save formats. Where the former occurred we first tried to recover the model using a disk repair utility. Incorrect save formats occured when non-PKRL models were saved without ensuring that the model window was at the front at the time of the save: by that means the source model was lost.[4]
6. Excessively idiosyncratic models were those that described

[4]This is probably a design weakness. See technical note 2 in Appendix 6.

their domains in obscure terms such that the researcher was unable confidently to determine the intended meaning. Some students used adjectives or nouns that are not in the researcher's vocabulary. Others used familiar language in an ambiguous or otherwise unclear way.

7.2.2 Distribution by domain and model type

Of the 568 models that were accepted for analysis, Table 7.4 shows the distributions by domain (shown in rows) and by model type (shown in columns). As can be seen, the proportion of classification tree (PCT) models is low but the other three model kinds are quite evenly represented. The distribution of domains reflects the shortage of time to complete the course and unfortunately there are fewer complete sets of models than had been expected, even for the earlier stages. For example, complete sets of (three) models are available for only 33 children in the Flags domain and for only 12 children in the Widgets domain.

	PFT	*PDT*	*PCT*	*PKRL*	Total	Total %
Horses	52	61	0	54	167	29%
Flags	50	48	0	49	147	26%
Widgets	17	21	0	22	60	11%
Darts	0	0	30	0	30	5%
Boings	32	37	36	36	141	25%
Pondoids	4	3	7	9	23	4%
Total	155	170	73	170	568	100%
Total %	27%	30%	13%	30%	100%	

Table 7.4 Distribution of models by domain and model type

7.2.3 Correctness

Table 7.5 presents the mean correctness measures for the 568 models grouped by domain and by model type. As can be seen, correctness is at very high levels in all domains except Boings and Pondoids. The overall mean correctness of PKRL models is the lowest of the four types although the differences are very small. Within some domains, however, a few larger differences between certain model types are apparent.

	PFT	*PDT*	*PCT*	*PKRL*	All
Horses	99%	94%	-	99%	97%
Flags	97%	97%	-	92%	96%
Widgets	88%	98%	-	86%	91%
Darts	-	-	86%	-	86%
Boings	63%	61%	90%	67%	70%
Pondoids	68%	61%	79%	55%	65%
All	89%	88%	87%	86%	88%

Table 7.5 Mean correctness of models by domain and model type

To test within domains the null hypothesis that the type of modelling tool has no effect upon the correctness of the model, we extracted correctness data for matched pairs of models. This is illustrated by Table 7.6 for the case where the matched pairs each contain a PFT and a PDT model built (in either order, recalling that the sequence was counterbalanced) within the Horses domain by the same pupil. A pair was excluded if data for one model is missing. We then applied the Wilcoxon signed ranks test to the correctness levels of the matched pairs. This was repeated for domains and model kinds selected on the basis of an exploratory data analysis, including the use of Friedman tests.[5]

	Horses/PFT correctness	*Horses/PDT correctness*
Pupil 1	95%	90%
Pupil 2	100%	85%
	.....	.....
Pupil n	90%	92%

Table 7.6 Organisation of data for the Wilcoxon test

Table 7.7 shows the results where differences significant at the 5% level or better were obtained. For example, the penultimate row shows that in the Boings domain, 27 matched pairs of PCT and PFT models were tested. The very low p-value strongly suggests that the null hypothesis can be rejected. The difference in correctness values is a significant one and PCT has the higher mean correctness of the two.

---

[5]Initially we used repeated measures ANOVA tests followed by selected matched-pairs t-tests. However, the correctness data is distributed with a left skew and so the analyses were repeated with the non-parametric Friedman and Wilcoxon tests which do not depend upon an assumption that the data has a normal distribution.

Domain	*Model types compared*	*No. of pairs*	*Z=*	*p =*	*Type with higher mean correctness*
Flags	PFT vs PKRL	39	-2.2600	0.0238	PFT
Boings	PCT vs PKRL	28	-3.9281	0.0001	PCT
Boings	PCT vs PFT	27	-3.8605	0.0001	PCT
Boings	PCT vs PDT	28	-4.4177	< 0.00005	PCT

Table 7.7 Domains and model types showing differences in correctness

These results provide some evidence that the correctness of a model is affected by the modelling tool, although the effects are not established for all domains and tools. An examination of students' models makes clear that in the hierarchical Boings domain, models in PKRL, PFT and PDT usually neglected to describe the non-leaf categories of the hierarchy. By contrast, PCT models almost invariably did define these non-leaf categories and this accounts for much of the superior correctness of these models. In the Flags domain an examination of PKRL models shows that they contain slightly more errors (we suspect, slips) than do PFT models. We note that the simple and regular structure of a factor table is well matched to the structure of the Flags domain and it may be that compared to a set of rules, a factor table is easier for students to check for correctness.

7.2.4 Efficiency

Table 7.8 below presents the mean efficiency measures. There are some large differences, with PKRL models obtaining relatively low mean scores in all domains and PCT models obtaining relatively low mean scores in the hierarchical domains. As before, we extracted data for particular kinds of matched pairs of models. We then selectively applied the Wilcoxon signed ranks test to the efficiency levels of the matched pairs. Table 7.9 shows the results where differences significant at the 5% level or better were obtained.

	PFT	*PDT*	*PCT*	*PKRL*	All
Horses	100%	99%	-	57%	86%
Flags	94%	93%	-	60%	82%
Widgets	75%	100%	-	55%	76%
Darts	-	-	44%	-	44%
Boings	79%	100%	50%	62%	73%
Pondoids	72%	83%	40%	45%	53%
All	90%	97%	46%	58%	77%

Table 7.8 Mean efficiency of models by domain and model type

Domain	*Model types compared*	*No. of pairs*	*Z=*	*p =*	*Type with higher mean efficiency*
Horses	PFT vs PKRL	43	-5.6454	<0.00005	PFT
Horses	PDT vs PKRL	48	-5.8942	<0.00005	PDT
Flags	PFT vs PKRL	39	-5.1594	<0.00005	PFT
Flags	PDT vs PKRL	41	-5.3731	<0.00005	PDT
Widgets	PFT vs PDT	15	-2.2014	0.0277	PDT
Widgets	PFT vs PKRL	14	-1.9775	0.0480	PFT
Widgets	PDT vs PKRL	15	-3.4078	0.0007	PDT
Boings	PFT vs PKRL	26	-3.4356	0.0006	PFT
Boings	PDT vs PKRL	28	-4.6226	<0.00005	PDT
Boings	PCT vs PKRL	28	-4.0305	0.0001	PKRL
Boings	PCT vs PFT	27	-3.6758	0.0002	PFT
Boings	PCT vs PDT	28	-4.6226	<0.00005	PDT
Boings	PDT vs PFT	28	-3.5740	0.0004	PDT

Table 7.9 Domains and model types showing differences in efficiency

These results provide fairly strong evidence that in the flat-structured domains (Horses, Flags and Widgets) students' PKRL models are inefficient compared to models constructed in PFT and PDT, and of the latter two types the result in the Widgets domain favours PDT. In the hierarchical Boings domain, PFT and PDT models perform best and PCT models are even worse than PKRL models.

The explanation for the inefficiency of students' PKRL models becomes clear from an examination of these models. Analysis of the Primex inferencing strategy shows that to be efficient, in general a PKRL program should use parameters and multiple-choice menu questions. For the other modelling tools, the tool compilers automatically generate this form of code but of course in a PKRL model the code is hand-crafted. Although the

necessary techniques were included in the modelling course, over 90% of students' PKRL models are coded in a naïve propositional syntax that makes no use of them. Furthermore, propositions are routinely expressed in an inconsistent way (i.e. unequal symbols are used at different points in the model to express what is evidently intended to be the same proposition). At runtime, such a naïve and inconsistent model will typically generate a large number of yes/no questions, most of which are actually redundant. The size of the problem is indicated by the fact that over half of students' PKRL models for the Flags domain generate between 51 and 66 questions to elicit all categories: an optimally performing model generates 28 questions. The problem is not one that can be remedied by any simple revision of the PKRL compiler.

The inefficiency of PCT models however should be relatively easy to rectify. As explained in section 5.3.3.1, it was decided to base runtime interpretation of these models on a bottom-up traversal of the classification tree. This process is inherently inefficient. By rewriting the PCT compiler to exploit a top-down interpretation strategy, students' models should be made to perform much better. The source-level representation of the model is unaffected. However, the task is not merely a technical one: it will be important to implement the interpretation strategy in a way that is not only efficient but also transparent to users, as discussed previously.

Models in PFT showed a very large variance in efficiency in the Widgets and Boings domains. Inspection of these models indicates that the explanation lies in the use of the asterisk character which acts as a wildcard or 'don't care' entry for factor table values. Wildcards affect efficiency because they indicate questions that can be skipped. Although the wildcard's proper use was covered in the modelling course, many children seem not to have properly learned the technique. Some appear not to know that the wildcard exists: they have constructed excessively large factor tables which redundantly enumerate all combinations of factor values.[6] Other students know that a wildcard exists, but not

[6]Interestingly, these factor tables would have run efficiently under the PFT's induction compiler. However, for the purpose of the large scale trials PFT was configured to use the naïve ('Left to right') compiler, for reasons that were explained in section 5.2.3.

which symbol to use: their factor tables contain underscores or hyphens instead of asterisks.

7.2.5 Conciseness

Tables 7.10 and 7.11 presents the mean conciseness and economy results. For completeness, we present in Appendix 16 the results where differences significant at the 5% level or better were obtained. However, these results seem less interesting than those for correctness and efficiency. Means are generally very high[7] and variances rather low, presumably reflecting the limited scope that exists for alternative approaches to representation within the trial domains.

	PFT	*PDT*	*PCT*	*PKRL*	All
Horses	100%	99%	-	103%	101%
Flags	95%	95%	-	118%	102%
Widgets	78%	100%	-	109%	97%
Darts	-	-	113%	-	113%
Boings	83%	99%	95%	100%	95%
Pondoids	88%	89%	120%	101%	103%
All	92%	98%	105%	107%	100%

Table 7.10 Mean conciseness of models by domain and model type

	PFT	*PDT*	*PCT*	*PKRL*	All
Horses	100%	98%	-	104%	100%
Flags	100%	98%	-	101%	100%
Widgets	98%	100%	-	113%	104%
Darts	-	-	105%	-	105%
Boings	104%	100%	106%	115%	106%
Pondoids	83%	88%	92%	94%	91%
All	98%	98%	104%	106%	101%

Table 7.11 Mean economy of models by domain and model type

The very high levels of conciseness of PKRL models has at least two sources. First, a small minority of students used a highly compact form of coding. To illustrate, consider the performance model of the Flags domain, which happens to be a model in PDT.

[7]Figures over 100% show that some of our performance models were not really optimal in conciseness and economy. As explained below, there is some tradeoff between these measures and efficiency: our performance models were selected to prioritise the latter.

The normal form generated for this model contains ten rules each of four phrases. The following is an example:

```
scouts if
  Stars = 0  &
  Colour = grey &
  Border = thick.
```

In their exceptional PKRL models, the students in question expressed the rule as a parameterised assertion:

```
Team : 0 grey thick scouts.
```

— which we regarded as just one phrase. With this form of code the entire domain can be described in only ten such phrases plus one five-phrase 'control' rule, giving an extremely high conciseness score of 40/15 = 267% relative to the performance model.[8]

A second explanation for the conciseness of PKRL models is the practice of some students of conflating two or more rule conditions into one. An example (relating to the same domain as above) is:

```
ADVISE The team is the Scouts IF
    The flag is grey with no stars AND
    The flag has a thick border.
```

This representation uses three phrases compared to the four taken by the performance model. However, it achieves conciseness at the expense of efficiency. This is because at runtime, if the user denies the rule's first condition then only two flags out of ten have been eliminated. In contrast, with the non-conflated (but unparameterised, and so still not ideal) rule:

```
ADVISE The team is the Scouts IF
    The flag is grey AND
    The flag has no stars AND
    The flag has a thick border.
```

[8]It could be argued that the parameterised assertion shown should be taken as several phrases. A problem is that our definition of a phrase in section 6.1.3 relates to normal form, which is difficult to reconcile with the parameterised PKRL code used in these exceptional models.

— denying the first condition eliminates five flags.

Inspection of models suggests that high levels of economy within PKRL and PCT models are largely due to the many instances in which students failed to distinguish syntactically between attributes and values. Within PCT models this could happen because as explained in section 5.3.1.3, the tool tolerates a relaxed syntax in which a feature need not be entered in the form Attribute=Value. An expression such as `They are red` is admitted by the tool and compiles to the condition `They are red=true`. A student who uses this form may produce the following (normal form) rule for the Boings domain:

```
The creatures are Cards if
        They are red = true &
        They have smooth skin = true &
        They live on Blip = true .
```

The corresponding rule from the performance model is:

```
The creatures are Cards if
        colour = red &
        skin texture = smooth &
        Home planet = Blip.
```

Here, the student rule uses only five distinct symbols compared to seven for the performance model's rule, so the student's representation is more economical. Again however, to fail to separate attributes from values will gain economy generally at a cost in efficiency.

7.3 Build time

Table 7.12 shows the mean build times in minutes for models by domain and model type. Perhaps the most practically important result is that the overall mean build time is less than 14 minutes, suggesting that model building with these tools can be a task of appropriate duration for a single school lesson period of time, providing domains are not too complex. We note that in the regularly structured flat domains (i.e. Horses and Flags), models are most rapidly constructed by PFT, whilst in the irregularly structured flat Widgets domain PFT models are not so quickly

built as models in PDT or PKRL.[9]

Since children were not invited to build their models speedily we do not wish to give these results too much weight. With more practice with a tool, build times may fall. This is suggested by the data on PCT models: although these models have the greatest mean build time, the time for the second PCT model (Boings) is considerably less than the time for the first (Darts) even though the Boings model is no less complex.

	PFT	*PDT*	*PCT*	*PKRL*	All
Horses	7.7	11.6	-	10.5	10.0
Flags	11.2	19.9	-	16.5	15.9
Widgets	19.3	13.3	-	15.0	15.6
Darts	-	-	25.7	-	25.7
Boings	11.9	12.9	13.6	11.0	12.4
Pondoids	14.3	7.0	18.0	13.4	14.6
All	11.2	14.4	19.4	13.2	13.8

Table 7.12 Mean build times (in minutes) by domain and model type

7.4 Pupil questionnaires

As noted previously, at the conclusion of the modelling course pupils completed a questionnaire. This is contained in Appendix 17. Children were asked to say whether they had enjoyed and understood their model building activities with each modelling tool, how easy they had found the tool to use, and whether they would like to use the tool again. Responses were on a seven-point scale, with the midpoint representing neutral feelings.[10]

Completed questionnaires were obtained from 67 pupils, representing a return rate of 82%. Table 7.13 shows the number of responses at each scale point for each tool. Figure 7.1 summarises the data by showing the mean responses in a barchart.

---

[9]Paired-sample t-tests confirm these results at the 1% level of significance. Other differences in timing are not significant at this level.

[10]The questionnaire was designed to use clear, simple language, and is consistent in locating positive responses in the right hand columns. We accept that this design incurs a risk of 'cueing' positive responses. However, the findings reported here are generally consistent with other feedback we obtained, including that which we report in the next chapter from children in the focussed studies.

	Disagree ←			Neutral		→ Agree
PCT						
I enjoy building these models	6	3	8	12	27	6	3
I understand the ideas behind them	1	4	3	9	26	15	7
I find the software easy to use	2	6	8	12	18	14	5
I would like to use this software again	12	1	4	14	9	18	7
PDT						
I enjoy building these models	4	4	2	7	15	19	15
I understand the ideas behind them	0	1	1	6	15	27	16
I find the software easy to use	0	2	6	8	17	13	20
I would like to use this software again	6	4	0	12	13	12	19
PFT						
I enjoy building these models	4	4	9	12	13	14	11
I understand the ideas behind them	0	2	2	13	13	18	19
I find the software easy to use	0	4	4	15	13	14	17
I would like to use this software again	8	4	6	14	12	11	12
PKRL						
I enjoy building these models	8	7	1	17	16	12	6
I understand the ideas behind them	4	2	2	10	17	16	16
I find the software easy to use	2	3	3	14	18	14	13
I would like to use this software again	11	7	3	21	6	10	9

Table 7.13 Responses to pupil questionnaires

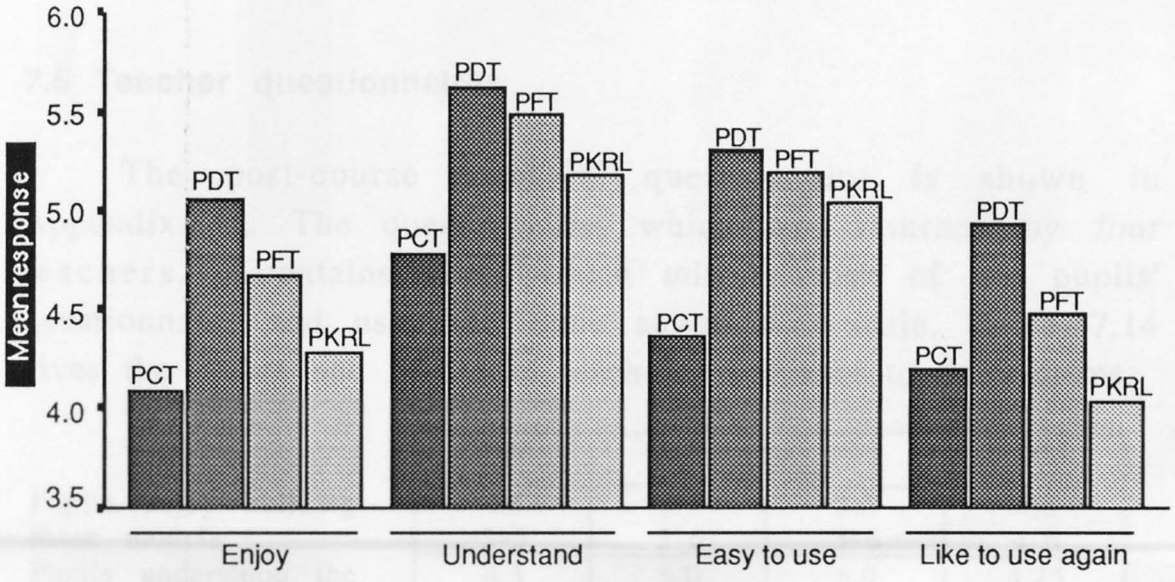


Figure 7.1 Mean responses to pupil questionnaire

As can be seen, PDT gains the most favourable mean response in all four aspects, followed consistently by PFT. The tool which pupils say they least enjoy, least understand, and find least easy to use is PCT, whilst PKRL is the tool which they would least like to use again. However, pupils' responses vary between schools and the differences may be large, as Figure 7.2 indicates. The

boxplots show the distributions of responses to the 'understand ideas' item of the questionnaire from pupils in the three schools which contributed the greatest quantities of models. Although in the case of PCT the distribution of responses is broadly similar for all schools, that for PKRL varies markedly. We cannot be sure why this is so: certainly the mean values for correctness of PCT and PKRL models do not differ significantly between the three schools.

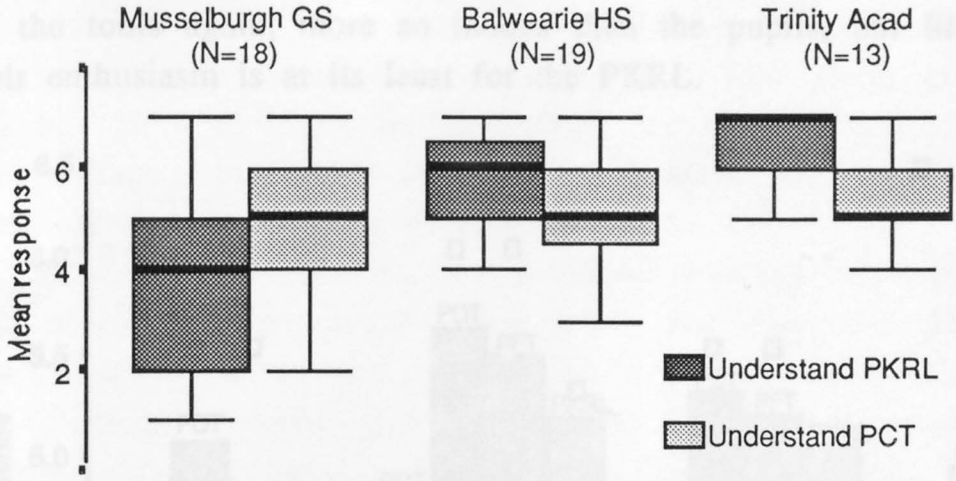


Figure 7.2 Boxplots showing responses grouped by school

7.5 Teacher questionnaires

The post-course teachers' questionnaire is shown in Appendix 18. The questionnaire, which was returned by four teachers,[11] contains items which mirror those of the pupils' questionnaire and uses the same seven-point scale. Table 7.14 gives the means and ranges of teachers' responses to these items.

	PCT	PDT	PFT	PKRL
Pupils enjoy building these models	4.5 3-5	5.5 5-6	5.5 5-6	4.5 4-6
Pupils understood the ideas behind them	4.5 2-6	6.0 5-7	6.0 5-7	5.25 4-6
Pupils' models were of high quality	4.25 3-5	5.5 4-6	5.5 4-7	4.5 4-5
I would like to use this software again	6.5 6-7	6.5 6-7	6.5 6-7	5.75 4-7

Table 7.14 Means and ranges of teachers' questionnaire responses

[11] The fifth teacher wrote a letter which expressed views consistent with those reported here.

Figure 7.3 shows teachers' mean responses as white squares superimposed on the barchart of the pupils' questionnaire results. From this it can be seen that the teachers, in making estimations of pupils' enjoyment and understanding, rank tools in almost the same order as do pupils, although teachers may be rather optimistic about the extent of pupils' enjoyment. Likewise, the tools which pupils regard as easier to use are identified by teachers as giving higher quality models. Teachers are keen to use all the tools again, more so indeed than the pupils, but like pupils their enthusiasm is at its least for the PKRL.

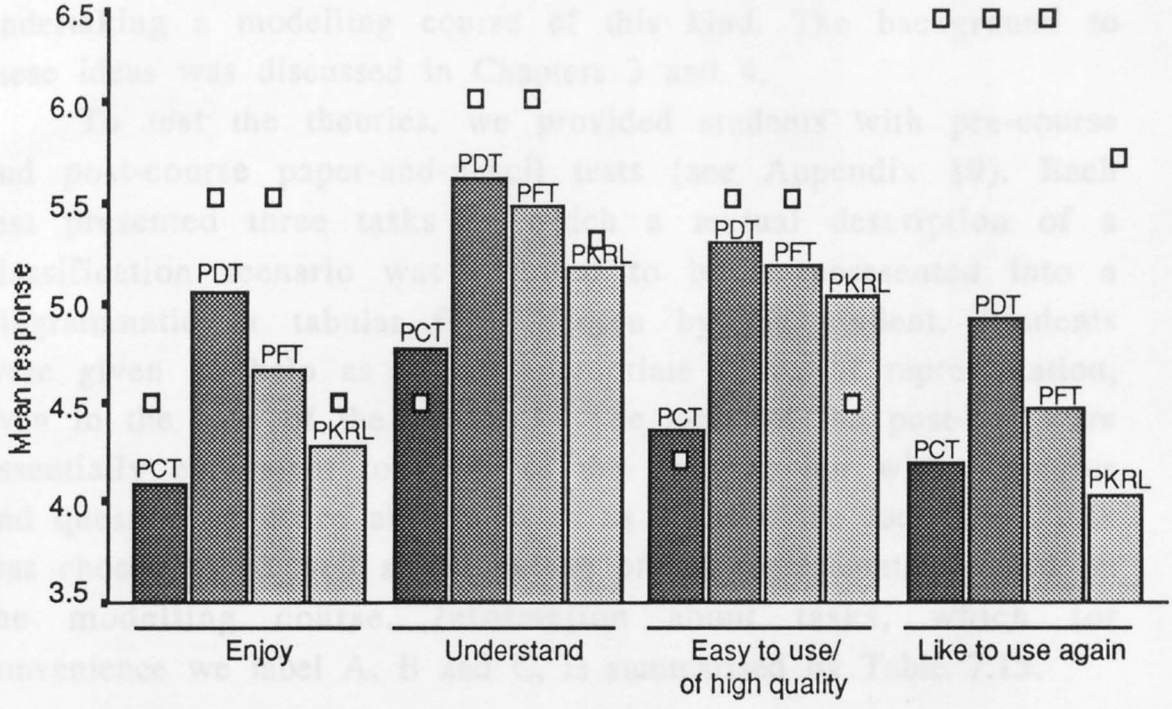


Figure 7.3 Teacher's mean responses (squares) on pupils' barchart

Teachers' written comments supplied in response to the questionnaire are reproduced in Appendix 18. These mostly confirm or slightly elaborate upon the structured responses, but some identify specific problems and sources of satisfaction. The problems mainly relate to technical difficulties with the software, including bugs and excessive load times. Sources of teachers' satisfaction included:

> *Seeing the great satisfaction of some pupils for models other than rule models (LJ).*
> *When offered a free choice on last period of term (after trial was complete) some pupils wanted to complete or begin more Primex models (RG).*

Pupils working well — concentrating and enjoying the tasks (AC).

7.6 Representational skills

The modelling course seemed to provide a useful opportunity to test two theories. First, that in selecting decision trees, factor tables, and classification trees as the representational bases for the new knowledge acquisition tools, we have made good choices; good in the sense that children have some prior familiarity with these representations and find them accessible. Second, that children develop their representational skills by undertaking a modelling course of this kind. The background to these ideas was discussed in Chapters 3 and 4.

To test the theories, we provided students with pre-course and post-course paper-and-pencil tests (see Appendix 19). Each test presented three tasks in which a textual description of a classification scenario was required to be re-represented into a diagrammatic or tabular form chosen by the student. Students were given no help as to the appropriate forms of representation, even in the case of the pre-test.[12] The tasks of the post-test were essentially equivalent to those of the pre-test, but with scenarios and question sequence altered so as to disguise the fact. Each task was chosen to be well suited to one of the representations used on the modelling course. Information about tasks, which for convenience we label A, B and C, is summarised by Table 7.15.

Task	*Pre-test qn. no.*	*Post-test qn. no.*	*Type of information in classification scenario*	*Modelling course representation*
A	1	3	Flat structured, regular, declarative	Factor table
B	2	1	Hierarchical	Classification tree
C	3	2	Flat structured, irregular, procedural	Decision tree

Table 7.15. Contents of pre-course test and post-course test

[12]In the researcher's discussions with teachers prior to the trials there was a general apprehension that pupils would flounder with the pre-test. It was agreed that if this transpired, teachers would stop the test early. The results suggest that the anxiety was misplaced.

A total of 79 completed pre-tests were returned. These were analysed to identify the representations that had been used for each scenario. The results are summarised by Table 7.16. As can be seen, factor tables, classification trees and decision trees featured very prominently and taken together these forms comprised 70.4% of all student responses. We take this as a strong endorsement of the first theory cited above. That is, the representational forms upon which we based the design of the new knowledge-based modelling tools are ones with which many students already have some familiarity.

Form of response	*Task A*	*Task B*	*Task C*	*Total*	*Total %*
Factor table	51	3	1	55	23.2%
Lookup table	9	0	0	9	3.8%
Other kind of table	3	26	3	32	13.5%
Classification tree	0	43	0	43	18.1%
Decision tree	9	0	60	69	29.1%
Other kind of tree	2	4	1	7	3%
Flow chart	0	0	7	7	3%
Cartesian graph	1	0	0	1	0.4%
Unrecognised	1	2	4	7	3%
Left blank	3	1	3	7	3%
Total	79	79	79	237	100%

Table 7.16. Students' pre-test representations

We repeated the analysis for the post-test, for which returns were obtained from 67 students. Table 7.17 shows the results.

Form of response	*Task A*	*Task B*	*Task C*	*Total*	*Total %*
Factor table	48	3	3	54	26.9%
Lookup table	0	0	0	0	0%
Other kind of table	0	4	0	4	2.0%
Classification tree	0	53	1	54	26.9%
Decision tree	5	2	61	68	33.8%
Other kind of tree	1	4	0	5	2.5%
Flow chart	0	0	0	0	0.0%
Cartesian graph	1	0	0	1	0.5%
Rules	5	0	0	5	2.5%
Unrecognised	1	0	0	1	0.5%
Left blank	6	1	2	9	4.5%
Total	67	67	67	201	100.0%

Table 7.17. Students' post-test representations

Compared to the pre-course, the percentages of answers that feature factor tables, classification trees and decision trees have

increased: other forms of representation show a reduction. This suggests that students after the course have responded in a more focussed way. One possible explanation is that the modelling course has strengthened students' understanding of the representations that were encountered (or at least, the course has raised awareness of their utility).

However, focussing on particular forms of representation need not indicate a real change in understanding: the close proximity of the test to the completion of the course may simply have cued students towards forms of representation that they would have been capable of producing before the course began. To test whether a change in understanding had occurred, we looked for evidence of a change in the *quality* of the representations, independently of what forms of representation were used.

Our method involved rating the quality of each of the 414 student representations appearing in pre-tests and post-tests. Quality was assessed on a three-point scale, as follows:

3 The representational notation is definitely consistently applied and all information is definitely correctly specified.
2 The representational notation may be inconsistently applied or some information may be missing or some information may be incorrectly specified.
1 The notation is definitely inconsistently applied or some information is definitely missing or some information is definitely incorrectly specified.

Of course, it would be possible to design a scale with more numerous and finer distinctions, but we would have felt less confident that such a scale could be applied reliably to the students' representations. With the scale as specified we were able to assign ratings rather confidently. We then compared these ratings pairwise by student, as shown (for the case of Task A) in Table 7.18, using Wilcoxon signed ranks tests. These showed that the differences were not significant for Task A. However, significant differences were identified for both Task B ($Z=-2.7163$, $p=0.0066$) and Task C ($Z=-3.0400$, $p=0.0024$).[13] We claim that

[13]The result for Task A could be affected by an error that occurred in the third question of the post-test (see Appendix 19). Due to the error, the scenario described was actually non-deterministic (in contrast to the corresponding pre-test question). Some pupils' responses in the post-test clearly showed that the error had confused them.

these differences provide strong supporting evidence for the second theory described above: that as a result of the modelling course, students developed their abilities to construct good representations of the types of information described by these scenarios.

	Task A pre-test rating	*Task A post-test rating*
Pupil 1	2	3
Pupil 2	3	3
	.....	.....
Pupil n	1	2

Table 7.18 Organisation of data for the Wilcoxon test

7.7 Summary

This chapter has described the research approach of the main school trials. A large quantity of empirical data obtained from the trials has been presented and discussed. It will be helpful now to summarise the main findings by reference to the hypotheses which were declared at the start of the chapter.

7.7.1 Quality of models

The hypothesis was:

- Models built with the knowledge acquisition tools (PDT, PFT and PCT) are of higher quality than those built with the rule-based shell (i.e. with the PKRL).

We compared models of the four types using our formal quality measures for correctness, efficiency, and conciseness. In terms of correctness, we found domains in which PFT and PCT models proved superior to PKRL models. We found no domains where PKRL models gave the best results among tools for correctness. In terms of efficiency, PFT and PDT (but not PCT) models were superior to PKRL models in all domains. In terms of conciseness, the figures for PKRL models were rather good; however we recognised that this was sometimes a reflection of naïve representations which were concise at the expense of

efficiency. In addition to the formal quality measures, questionnaires provided teachers' own assessments of quality: teachers believed that pupil's PDT and PFT models were generally of good quality but they were more doubtful about the quality of models in PKRL and PCT.

It will be clear that these results do not justify a simple, unqualified confirmation of the hypothesis. However, it seems safe to say that if children are to be helped to build correct, efficient classification models then they should generally be given one of the new knowledge acquisition tools rather than the PKRL. The selection of tool will depend partly on the nature of the domain. More research is needed to say much more than this, but the present data at least allows us to make some plausible suggestions: for hierarchical domains a good choice may be PCT, providing this tool's compiler can be rewritten for more efficient code; for flat structured domains good choices may be PFT or PDT, with the latter being favoured when the structure is less regular or the domain specification is procedural.

7.7.2 Build times

The hypothesis was:

- Models are built by children more rapidly with PDT, PFT and PCT than with the PKRL.

We found that in the flat-structured domains, models were built most quickly with PFT or PDT. In the hierarchical domains we did not find significant differences. However, it should be noted that children in the trials were not asked to race against the clock. The more important practical result is that the overall mean build time for all models and domains is less than 14 minutes, suggesting that model building is an activity that may be accommodated within the duration of a typical school period.

7.7.3 Motivation

The hypothesis was:

- PDT, PFT and PCT motivate children to a greater extent than does the PKRL.

Pupils' questionnaires confirm this hypothesis for PDT and PFT. Compared to the PKRL, children say that they enjoy PDT and PFT more; they believe that they understand these tools better; they say that they find them easier to use and they are keener to use them again. Teachers' questionnaires confirm the success of PDT and PFT in motivating children. In the case of PCT however the evidence does not support the hypothesis. Children rate this tool less highly than the PKRL in terms of their enjoyment and understanding. They regard PCT as more difficult to use. However, less time was occupied in working with the PCT than with any other tool. Teachers were very keen to return to PCT in future, more so than the PKRL; children were less keen than their teachers but given a choice between the two, their preference would be the same.

7.7.4 Representational skills

The hypothesis was:

- As a result of working with our modelling tools, children develop their representational skills.

The evidence with which to assess this hypothesis comes from pre-tests and post-tests which required pupils to select and construct paper-and-pencil representations of specified information. The pre-tests immediately proved something important: that the representational forms upon which we based the design of PCT, PFT and PDT are ones which students partly know already. An analysis of post-tests revealed that students had become more focussed on the forms of representation used in the modelling course. More importantly, we found that their use of representations had risen in quality. We take this as strong evidence in support of the hypothesis.

Chapter 8 Children building models

Compared to the previous chapter, this chapter offers an evaluation that is more qualitative and less formal. Using evidence from videotapes, dribble files, and informal observation, we provide detailed reports of episodes in which a small number of children build models with our tools. Each episode is interpreted and discussed. Individual children's comments are presented as are the views of the class teacher. These small-scale focussed studies reveal some of the difficulties that children experience, including difficulties in understanding domains, selecting representations, and constructing models. In the course of the chapter it becomes clear that the new modelling tools, although quite usable, could be substantially improved. We distinguish improvements which could be made almost immediately from those which will require further research. We also make suggestions about how teachers could design and supervise modelling activities in the classroom.

8.1 Process-oriented, qualitative research

The previous chapter presented the findings from the large-scale school trials, including statistics derived from an analysis of children's models. These findings provide answers to several of the research questions which we posed originally in Chapter 3. However, they leave largely unanswered the question

What problems do children have in using the tools?

— which we also posed as a research question. Finding the answer to this question is important not only to inform changes that might be made to the design of tools in the future. The answer will be useful also to teachers seeking to make the best use of the tools as they presently exist. In the long term — and beyond the scope of this research — fine-grained information about the problems that children have with the tools may contribute to a cognitive theory of how children build models. In this chapter, we describe how we investigated the question.

8.2 Method and organisation

In Murray's (1993) categorisation of research methods, the methods appropriate for investigating behavioural phenomena are mostly based on observation. Relevant techniques include task analysis, tape-recorded problem solving sessions, interviews, and protocol analysis. Essentially a research approach is called for that is process-oriented and qualitative, rather than the mainly product-oriented and quantitative approach of the previous chapter. Accordingly, we planned a session of in-depth observation of a small group of children using the modelling tools.

The session was conducted in Moray House Institute of Education with a group of a dozen ten-year old children from a state-run Primary school in East Calder, West Lothian. The children were all from the same Primary 6 class. The children's teacher, Dorothy Johnstone, was an experienced Primary school teacher and also a student on the M.Ed. Computers and Learning programme at Moray House. She had taken part in a course in knowledge-based modelling at which she had become interested in the new Primex tools. Prior to the day's session she had introduced her class to Primex as part of their work on the theme of 'Diet and Digestion' in Environmental Studies. All children had therefore spent a few hours working with the tools.

Compared to the pupils involved in the large-scale trials, this group of children were much younger. This did not seem disadvantageous. There was no intention to correlate formally the data between the two contexts and the present session offered an opportunity to learn how Primary age children would cope with the software. Arguably, this group of children — who were regarded by Dorothy Johnstone as spanning a fairly wide range of attainment levels — constituted a more exacting test of the software's usability than might have been provided by an older group with more prior experience of Information Technology.

The session took place in a computer room containing twenty Power Macintosh computers each fitted with the Primex modelling tools. We also installed a specially developed utility that created dribble files to dynamically record children's progress in

model construction.[1] At three computers, video cameras and sound recording equipment were installed. The programme for the session is shown in Table 8.1.

Time	*Item*	*Comment*
1230	Welcome	Children welcomed to Moray House by researcher
1232	Review	Researcher uses LCD projection equipment to review the construction of PDT, PFT, PCT and PKRL model kinds with simple examples. Children are provided with tasksheets and given some advice about how to start.
1245	Task 1	Children working as individuals build PDT, PFT, and PKRL models of the Horses domain (see Appendix 12).
1340	Break	Lemonade & crisps
1400	Task 2	Children working in pairs build models of various domains in Environmental Studies; tasks are more loosely specified, books are provided as source material
1435	Conclusion	Children thanked and arrangements made for issue with certificates of participation
1440	Finale	Children leave. Dribble files and children's saved models are retrieved from hard disks. Videotapes removed from cameras.

Table 8.1 Programme for the observation session

Ideally we would have liked to have situated the session within the children's school but the organisational demands made this impractical. Instead, children were collected from their school in a minibus and brought to Moray House. There is little doubt that they were at first much affected by the novel circumstances of the session and by the large amount of special attention which they received. However, children became noticeably more relaxed after the first half hour or so.

In addition to the researcher and Dorothy Johnstone, also present to provide support were Helen Pain and Judy Robertson from the Department of Artificial Intelligence. All were briefed in advance on the aims and methods of the session. They were asked

[1]We investigated a commercial product, 'CameraMan' from Motion Works International, part of their 'Multimedia Utilities' package, but found that it created disk files that were excessively large, redundant, and inconvenient to analyse, since images were saved at fixed time intervals in bitmapped format. Our utility, written as a Primex extension in Prolog++, saves models in a much more efficient (object) format at intervals determined by significant user events, and provides convenient playback facilities.

to adopt a reassuring and friendly style but to provide assistance only sparingly, for example where there was a risk of children floundering. This is a difficult balance to achieve.

In the sections below we reconstruct some individual children's experiences in model building. The reconstructions are based upon the evidence that comes from videotapes, dribble file analyses, and the researcher's own records and observations. Having reviewed all the material, we focus on a few episodes selected for three main reasons. First, we have fullest evidence for these episodes, including videotape recordings of fair quality. Second, they are sufficiently diverse to suggest the wide range of different experiences that children had with the tools. Third, it seems more appropriate here to give deeper accounts of a small number of cases than shallower accounts of a greater number.

Even so, the evidence we have on the selected episodes is not perfect: for example the videotapes were pointed at computer screens, so children's faces and body movements are not recorded, and children's talk is sometimes rather indistinct against the background hubbub. Inevitably, the descriptions that are provided here are simplifications of events that were clearly highly complex. We aim not to provide exhaustive detail but to highlight the main qualities and key moments of each episode. In addition to the description of an episode, each section also contains a discussion that interprets the children's behaviour and suggests implications that could be important from the perspectives of tool design and classroom practice. For brevity, we refer to the adult helpers by their initials throughout.

8.3 Duncan

We first describe Duncan's experiences between the start of the session and the break.[2] During this time Duncan attempts to build first a PDT and then a PKRL model for the Horses domain (described in Appendix 12).

In Dorothy Johnstone's opinion, Duncan often works at a

[2] Duncan is not his real name. Throughout this chapter, children's identities have been disguised.

level below that of the other children in the class.[3] However, she had noticed that during the previous class work on 'Diet and Digestion', Duncan had worked keenly with Primex.

8.3.1 Duncan's PDT model

1245 Duncan arrives at his computer.

1247 Although children have been told that Primex has been pre-loaded, Duncan does not seem to recognise that the menubar on his computer already shows the Primex menus. His first click on the desktop area causes these menus to be replaced by those of the Macintosh Finder (i.e. the operating system interface).

1248 Duncan pulls down various Finder menus before asking for help. An adult helper (TC) arrives. He brings Primex back to the front and shows Duncan how to create a new decision tree window via the File menu's Extensions submenu. The helper then departs.

1249 Duncan creates and names a new PDT window. He uses the pen tool successfully to relabel the root node.

1250 Duncan selects the New Nodes tool and clicks on the root node. The click does nothing: the cursor was not accurately located within the node. The second click succeeds and adds two nodes.

1251 Duncan edits the labels of the newly created arcs and nodes, working in breadth-first order. He does not use the shortcut provided by the 'Go to Node below' button of the edit dialogue.

1252 Duncan adds three further nodes to the tree and labels each arc, working from left to right. The tree at this point appears as in Figure 8.1.

---

[3]In terms of the government's curriculum guidelines for the 5-14 stages.

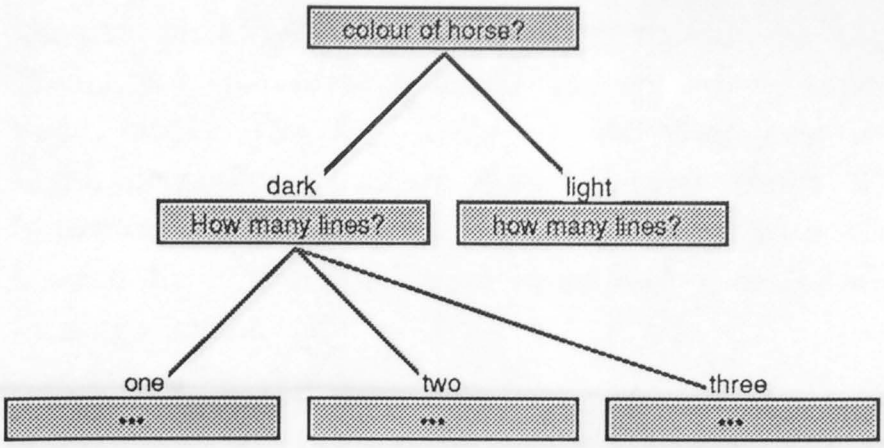


Figure 8.1 Duncan's PDT model at 1252

1253 Duncan clicks with the pen tool on the leftmost node at the third level. He pauses before typing 'Wanderer' into the edit field. At this point Duncan makes his first serious mistake: he clicks 'Ok' with the dialogue's default 'Question' setting still in force (see Figure 8.2). In consequence PDT appends a question mark to the name and 'Wanderer?' is displayed (in normal type, not bold) in the lower left node.[4]

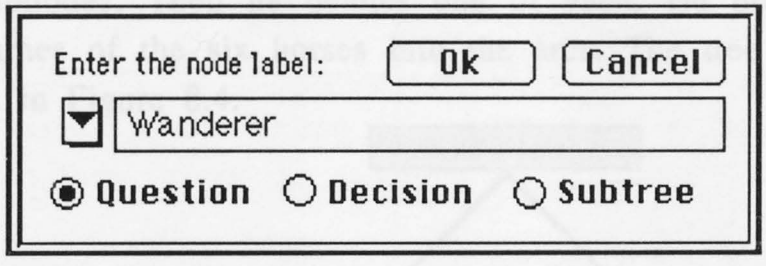


Figure 8.2 Duncan's response

1255 Following a pause, Duncan selects the eraser tool and deletes the node that he has just edited. He then creates a replacement node which, because it appears to the right of the two remaining third-level nodes, he relocates with the drag tool to the position of the node he has just deleted. He then labels the node's arc to 'name'. The tree now appears as in Figure 8.3.

[4]In PDT, nodes that represent decisions must be explicitly declared by the user. Their labels are then shown in bold type. See section 5.1.1.1.

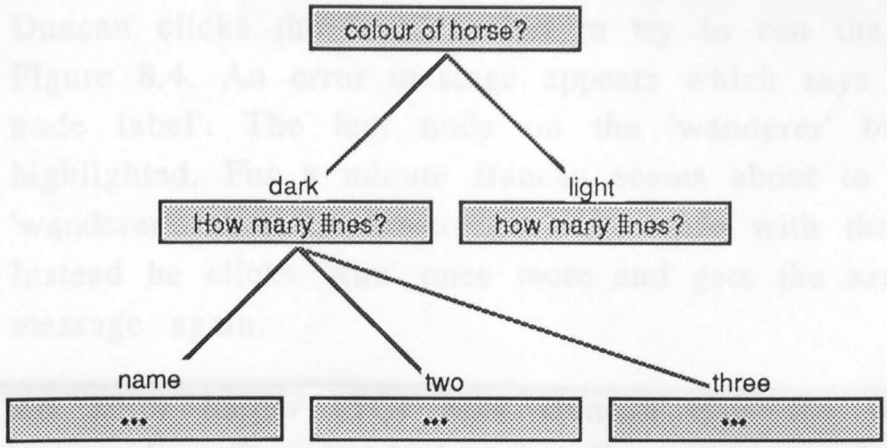


Figure 8.3 Duncan's PDT model at 1255

1256 Duncan edits 'name' to 'one'. He is now back to where he had been four minutes earlier (see Figure 8.1).

1258 Duncan adds three subnodes to the second level right hand node and labels their arcs 'one', 'two', 'three'. He then labels all six leaf nodes to 'name of hores', using the dialogue's popup menu to replicate the misspelt label in each node, each time leaving the dialogue's default 'Question' setting untouched. To each leaf node he rapidly adds two subnodes. Then he deletes one of each. He now enters the names of the six horses into the arcs. The tree now appears as in Figure 8.4.

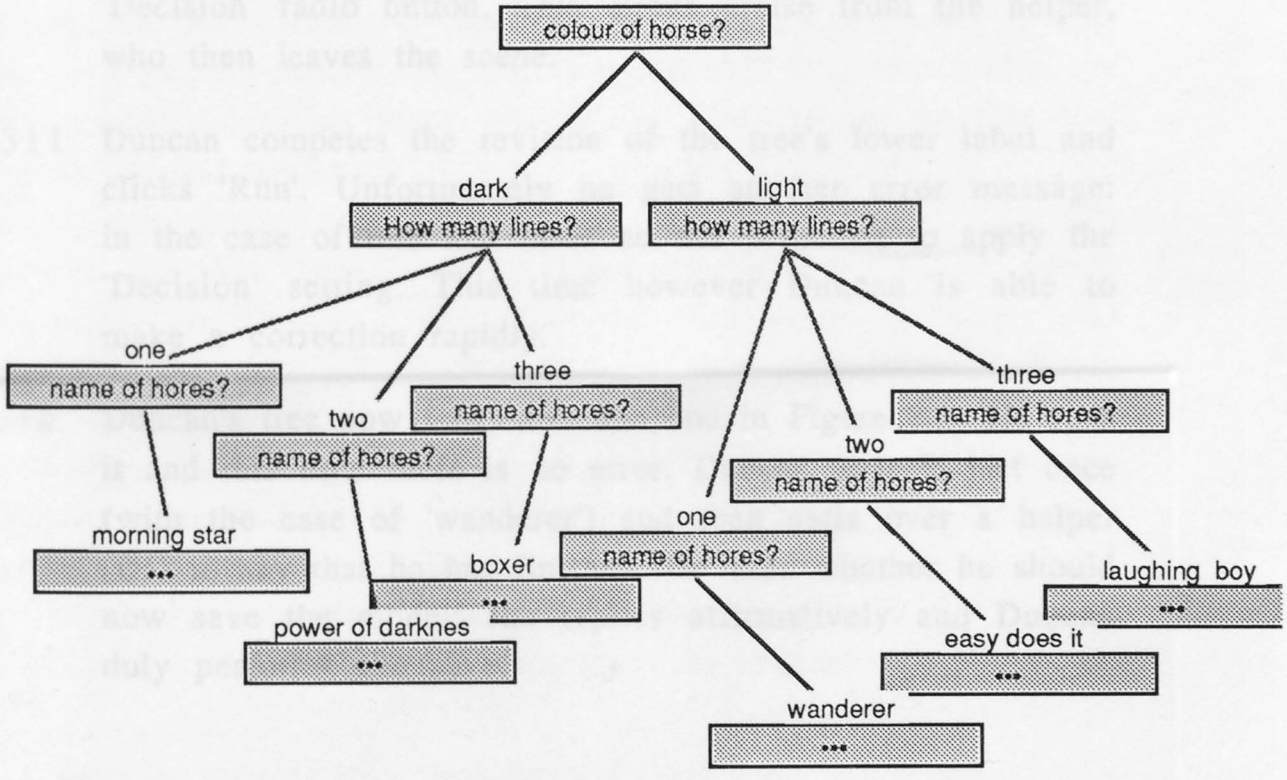


Figure 8.4 Duncan's PDT model at 1304

1304 Duncan clicks the 'Run' button to try to run the tree of Figure 8.4. An error message appears which says 'Missing node label'. The leaf node on the 'wanderer' branch is highlighted. For a minute Duncan seems about to edit the 'wanderer' label: he hovers over the node with the mouse. Instead he clicks 'Run' once more and gets the same error message again.

1306 An adult helper (TC) asks Duncan how the model is progressing. 'I've finished it but I'm not sure what to do with these bits', Duncan replies, pointing vaguely at the bottom of the tree. 'I've done it but I might have done it wrong'. A brief pause follows as the helper inspects Duncan's model. 'There's very little wrong with it', he declares. He traces with his pen down the tree's leftmost branch: 'If the colour is dark ... and the horse has one line ... then don't we *know* that it's name is Morning Star? So ...'. The sentence is completed by Duncan: 'So we don't need these boxes!', he says, and begins to delete the redundant nodes. Duncan then edits the label of the leftmost second-level node to 'morning star'. Without any prompting from the helper he remembers to click on the edit dialogue's 'Decision' radio button. This elicits praise from the helper, who then leaves the scene.

1311 Duncan competes the revision of the tree's lower label and clicks 'Run'. Unfortunately he gets another error message: in the case of one leaf node he has forgotten to apply the 'Decision' setting. This time however Duncan is able to make a correction rapidly.

1312 Duncan's tree now looks like the one in Figure 8.5. He runs it and this time there is no error. Duncan tests it just once (with the case of 'wanderer') and then calls over a helper (DJ) to say that he has finished. He asks whether he should now save the model: she replies affirmatively and Duncan duly performs the save.

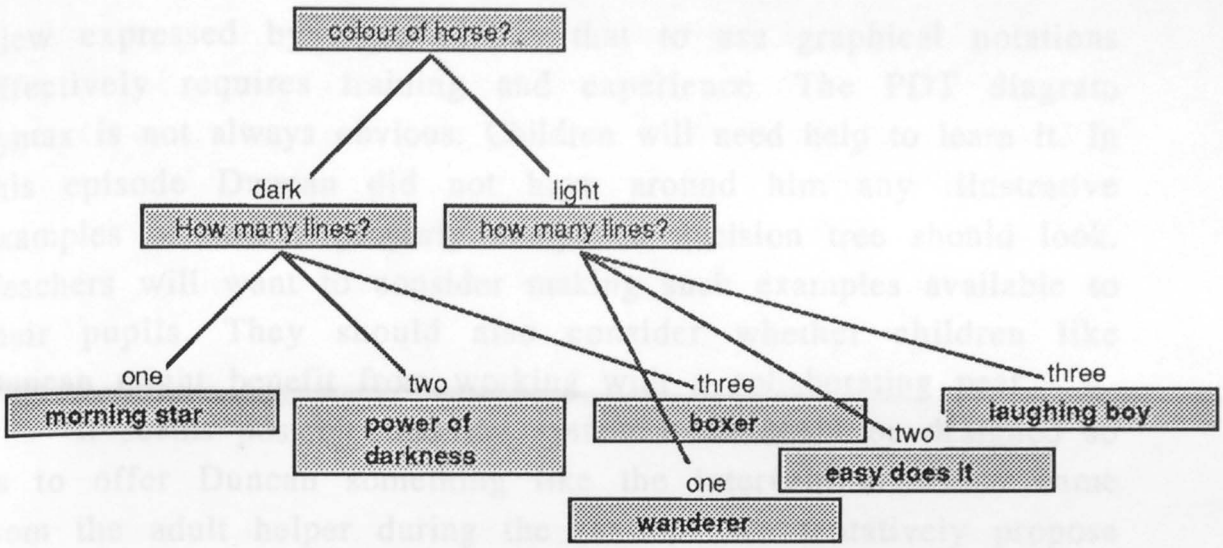


Figure 8.5 Duncan's PDT model at 1312

8.3.2 Discussion

It seems that Duncan lacks familiarity with the syntax of a PDT diagram. Specifically, he has trouble with the convention whereby decisions are expected to appear in specially designated nodes at the ends of branches. Duncan almost knows this rule: the intervention at 1306 is enough to put him back on track. One suspects that the difficulties would never have arisen had he only managed to click the 'Decision' button at 1253, after correctly typing the text 'Wanderer' into the edit field. Presumably it was the strange appearance in the diagram of that text, with its system-added question mark, that alerted Duncan to the fact that something was amiss.

Even expert users of software can forget to set a radio button; but for them it is an innocuous slip which is easily retrieved. In Duncan's case however the error at 1253 was retrieved only after a lengthy struggle that exposed a seriously weak conception of how diagrams should be constructed. Yet the sequence of events between the error and the intervention suggests that Duncan is tenacious and resourceful: he invents a syntax of his own — the one represented by Figure 8.4.[5] Unfortunately, it is not a syntax that is recognised by PDT.

From a pedagogical standpoint, the episode confirms the

---

[5] Possibly he invented two, since Figure 8.3 looks like the beginnings of a second syntax, though one that Duncan did not pursue.

view expressed by Petre (1993) that to use graphical notations effectively requires training and experience. The PDT diagram syntax is not always obvious. Children will need help to learn it. In this episode Duncan did not have around him any illustrative examples of how a properly completed decision tree should look. Teachers will want to consider making such examples available to their pupils. They should also consider whether children like Duncan might benefit from working with a collaborating peer.

It seems possible that the system itself could be designed so as to offer Duncan something like the intervention which came from the adult helper during the session. We tentatively propose that a student modelling system[6] (SMS) which traced Duncan's progress should be capable of detecting some of the symptoms of difficulty, such as the circular course taken between 1252 and 1256, the presence of decision tree nodes with only one child node at 1258, and the attempt to run the same buggy tree twice in succession at 1304. These symptoms can be detected without knowledge of the domain for which the tree is being constructed. Previous attempts at student modelling have often encountered major difficulties, arguably because developers' goals have been too ambitious (Self 1990). We are suggesting here the possibility of an SMS of very limited intelligence, one that has enough knowledge of PDT diagram construction to be able to offer occasional help of a domain independent and strategic kind. Systems of this type have been advocated by Cumming and Self (1990) who stress the benefits of envisaging an intelligent educational system in terms of a 'task layer' and a (lightly coupled, SMS-based) 'discussion layer'. It is a matter for future research to investigate what can be achieved by adding a discussion layer to PDT.

In the shorter term, observations of Duncan's difficulties with PDT have justified the consideration of some changes that might be made relatively easily. Examples are: configuration of the Macintosh operating system to prevent task switching of the kind that occurred at 1247; giving more obvious access to new model windows (1248); implementing a more forgiving algorithm for

---

[6] In the sense of a system which builds a model of a student's knowledge, not in the sense of a system with which students build models.

interpreting tool clicks (1250); providing support for the breadth-first development method preferred by Duncan, not just for depth-first development as at present (1251); redesign of the edit dialogue to highlight the significance of the choices offered by its radio buttons (1253), or more radically, review of the current PDT diagram syntax conventions; and more helpful error messages (1304).

8.3.3 Duncan's PKRL model

Duncan's task now is to make a rule model. The domain is the same Horses domain as before. Anticipating that this task will be difficult for Duncan, an adult helper (TC) has given him a sheet containing a listing of a PKRL model. The listing relates to a different domain but in size and complexity it is close to what Duncan is required to produce.

1314 Duncan immediately gets assistance from a helper (HP) in closing down his tree window and in creating a new PKRL window. The helper says: 'Now you've got a window there for doing the rules. Do you know how to do the rules?'. Duncan says he does. The helper leaves.

1317 With slow, hesitating keystrokes, Duncan has typed the following lines into the window:[7]

```
ADVICE   You have a dark horse AND
Has three lines it must be boxer.
```

He moves the cursor down two line and types the word ADVICE. Then there is a long pause. The cursor returns to the lines he has already typed. Duncan makes some small changes to these lines, then reverses them.

1321 The window now contains the lines:

```
ADVICE   You have a dark horse AND
Has three lines it must be boxer.

ADVICE You have a dark horse AND
```

[7]We reproduce Duncan's PKRL code in exactly its original form.

```
Has two lines it must be power of darkness.
```

Duncan clicks the Run button. The system highlights the first rule and displays the message: 'Oops! Incorrect use of Primex language in marked section!'.[8] Duncan pauses, makes a small change to the text of the second line, and tries Run again. He gets the same error message once more. After another small change he tries Run for a third time, with the same result. A long pause follows. Duncan then begins to type in a further rule.

1332 The window now shows:

```
ADVICE   You have a dark horse AND
Has three lines it must be boxer.

ADVICE You have a dark horse AND
       Has two lines it must be power of darkness.
ADVICE YOU   have adark horse AND
       Has one line it must be  morning star.

ADVICE You have a light coloured horse AND
        has one lines it must be wanderer.

ADVICE You have a light coloured horse AND
       HAS two lines it must de easy doese it.

ADVICE YOU have a light coloured horse AND
       HAS three lines it must be laughing boy.
```

Duncan clicks Run. As before, the system highlights the first rule and displays the message: 'Oops! Incorrect use of Primex language in marked section!'. Duncan emits a loud sigh.

1336 Duncan's cursor is hovering over the first rule. He inserts a few spaces, pauses, then deletes them again.

1338 In the background an adult can be heard calling the children to break for lemonade and crisps. Duncan departs. No further work is done on the model, which has still not run.

---

[8]Rules in the PKRL have syntax that admits the form:

```
ADVICE <Conclusion> IF <Condition> AND ... AND <Condition>.
```

so if the rule's `AND` was changed to `IF`, it would be legal (but not correct). Note that upper-case is mandatory for PKRL keywords.

8.3.4 Discussion

As with his previous model, Duncan again experiences difficulty with syntax. He omits the necessary IF from each rule. His use of upper-case characters suggests that he has been misled by the PKRL model listing with which he has been provided. He correctly applies upper case form to the keywords ADVICE and AND but also unnecessarily sometimes uses this form for the non-keywords YOU and HAS.

Even more seriously, Duncan misunderstands the semantics of PKRL rules. Presumably he believes that (for instance):

```
ADVICE   You have a dark horse AND
Has three lines it must be boxer.
```

— is equivalent in meaning to the natural language construction:

> *if* You have a dark horse *and* It has three lines
> *then* it must be boxer

— whereas in fact, the PKRL interprets a rule's first phrase as the consequence for which the subsequent phrases represent conditions. This kind of misunderstanding, concerning the direction of reasoning that is specified by the text of rules, has been observed in previous studies of children using rule-based shells (Galpin 1989, Webb 1992, Wideman & Owston 1993).

Duncan was defeated by his combination of difficulties with syntax and semantics. Again, however, we can see that he is keen to succeed and resourceful. The syntax that he constructs is not correct but it does show consistency.

Also, Duncan shows metacognitive awareness. His use of testing at 1321, at a point when the model is far from complete, suggests that he recognises that his grasp of the PKRL is weak: he monitors his own progress and knows that by testing what he has written, the system may provide helpful feedback. Unfortunately, the error message he gets is not adequate to enable him to diagnose and correct his own misconceptions. For some reason he does he not seek human help and this time there is no adult

intervention. Thus there is only one thing left for Duncan to do: he carries on with the task of entering rules. Does he believe that the problem revealed by the test will go away? It seems unlikely, but we cannot be sure.

It is interesting to contrast Duncan's experiences with the environments of the PKRL and PDT. The former provides almost no structure or constraint: Duncan just gets a blank text window into which anything might be written. The latter provides a set of tools with which (basically) only one kind of structure, a decision tree, can be produced. Thus valid syntax is strongly supported in the PDT whereas in the PKRL it is not much more than a technical possibility. Furthermore, the form of representation provided by the PDT provides a conceptual framework — based around the relationships between questions, answers, and decisions — that is useful for building classification models. Duncan has some success with this framework. The conceptual framework of the PKRL is much weaker (more general-purpose, and hence less specific to classification) and Duncan struggles to relate it to the task in hand. Although he also got his PDT model wrong, and needed human help, in the end the model worked. In contrast, even if the adult helper had returned to Duncan's PKRL model at 1332 she would have been faced with a formidable task of explanation and the model would have necessitated extensive restructuring.[9]

How might system designers help Duncan to build better PKRL models? At the very least, syntax errors should provide more helpful error reporting. However, more radical measures are justified. If it can be shown that children would find this form more natural then rule syntax should be revised so that conditions precede consequence. Possibly too rule conditions should each take the form of a feature expression `Attribute=Value` in order to provide a stronger conceptual framework for describing classification models. This would serve also to make PKRL models more amenable to analysis by tools like the PMA (see section 6.2.3). A structure editor, like the one provided with Adex Advisor (Briggs 1987) for example, could ensure that the necessary syntax is respected. Such an editor could also help to reduce the amount

[9]Note also the difference that Duncan had no example of a PDT model to refer to, whereas he did have a sample listing of a PKRL model.

of inconsistency of representation that was identified in the previous chapter as a cause of inefficiency in PKRL models. Possibly too an intelligent compiler could optimise naïve and inconsistent code.

To say whether these measures can make rules an adequately usable representation for children will require research. In the meantime, teachers might do better to steer children like Duncan towards alternative representations.

The last words should come from Duncan himself. On returning to school after the session, Dorothy Johnstone asked the group to write a paragraph describing how they felt about the various modelling tools. Duncan wrote:

(PDT) *This was the best one because it did not involve a lot of tipeing and you had to think a bit more than the rest.*
(PKRL) *I did not like this because it involved a lot of tipeing but I liked it how it asked you questions.*

8.4 Ronnie and Bob

After the interval (which was cut short at the children's insistence) Ronnie and Bob were asked to work together to build a PCT model in the domain of insects. As source material they were given the book *Minibeasts* by Sally Morgan.[10] They were also given a list of names of fourteen insects that they should try to include within the model.

Ronnie and Bob are regarded by Dorothy Johnstone as high achievers. Ronnie likes challenges and mathematical problems. He is normally very quiet. Bob is described as good at problem solving. Both boys greatly enjoyed Dorothy's previous classes with Primex.

8.4.1 Ronnie and Bob's PCT model

1352 Ronnie and Bob arrive at their computer. They talk animatedly as they read the task. Conversation about insects seems to be interwoven with negotiation about how

[10]Published by Wayland Press, 1995.

they will collaborate on the model. It is decided that one will act as typist and the other as mouse controller.

1355 A new classification tree window has been created and a laddered grid interview is underway. The model looks as shown in Figure 8.6.

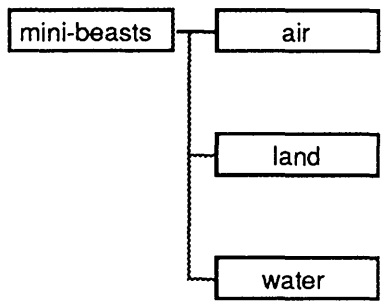


Figure 8.6 Ronnie and Bob's PCT model at 1355

1357 The system has generated the question:

Can you name some subclasses of air?

This provokes a discussion about whether the 'air' class is necessary. One of the boys is heard to ask of an adult helper (HP): 'Do we need air? Because, say, ladybirds stay on land'. The helper replies: 'It's up to you to decide, it's your model'. The air node is deleted with the eraser tool.

1359 When the interview is resumed at the land node, the question:

Can you describe some feature(s) of land?

— is generated.[11] One boy suggests a list of names: 'Spiders, beetles, ...', but the other contradicts him: 'No, they aren't features, they're subclasses'. They click no and get back:

So land is a kind of mini-beasts with no special features. Is that right?

This provokes more discussion, but they click 'Yes'. The

[11]PCT had been configured with the 'Ignore features on first pass' setting active, but by interrupting the interview to perform the deletion the boys have effectively ended that phase. See the discussion of PCT dialogue strategy in section 5.3.1.4.

system responds:

Can you name some examples or subclasses of land?

Considerable debate ensues during which four names are entered. The model now appears as in Figure 8.7.

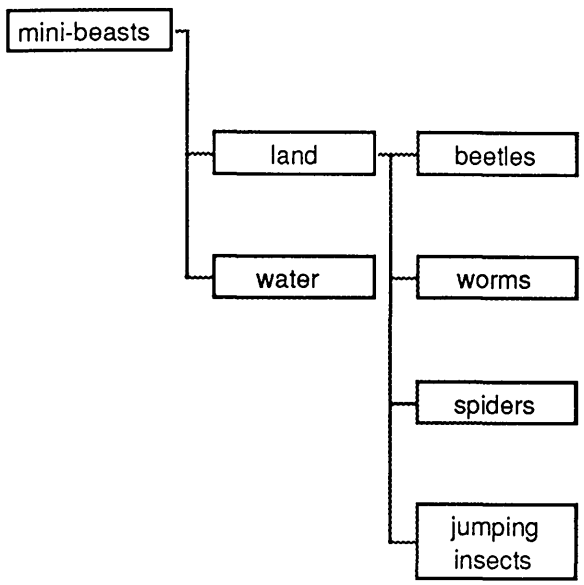


Figure 8.7. Ronnie and Bob's PCT model at 1359

1401 Ronnie and Bob have specified dung-beetles as a subclass of beetles. Asked by the system to describe a feature of dung-beetles they reply 'Gather dung'. This answer is made apparently without reference to the book. The boys seem amused when the system asks them to confirm that dung-beetles are a special kind of beetles that gather dung. The high spirits are evident also as they proceed to specify that features of click beetles and worms are 'make clicking noises' and 'gardener's friend'.

1406 The system asks whether 'gardener's friend' is true of spiders. One boy says not, but the other argues that it must be true because spiders catch earwigs and greenfly. This precipitates a lengthy debate. Finally they click 'No'.

1409 The boys swop places. The typist now becomes the mouse controller and vice-versa.

1415 The model now appears as in Figure 8.8.[12] One boy suggests that they run it in its present form but the other insists that first it should be saved to disk. When an adult helper (TC) passes by, they explain that on a previous occasion a classification tree model had been lost when the software crashed before the model had been saved. This time, the model having been saved then proceeds to run satisfactorily. Bob and Ronnie emit a loud 'Yes!' when the system's cursor briefly changes to an up-thumb.[13] When the system correctly classifies their test case of a wolf spider, they both let out a whoop.

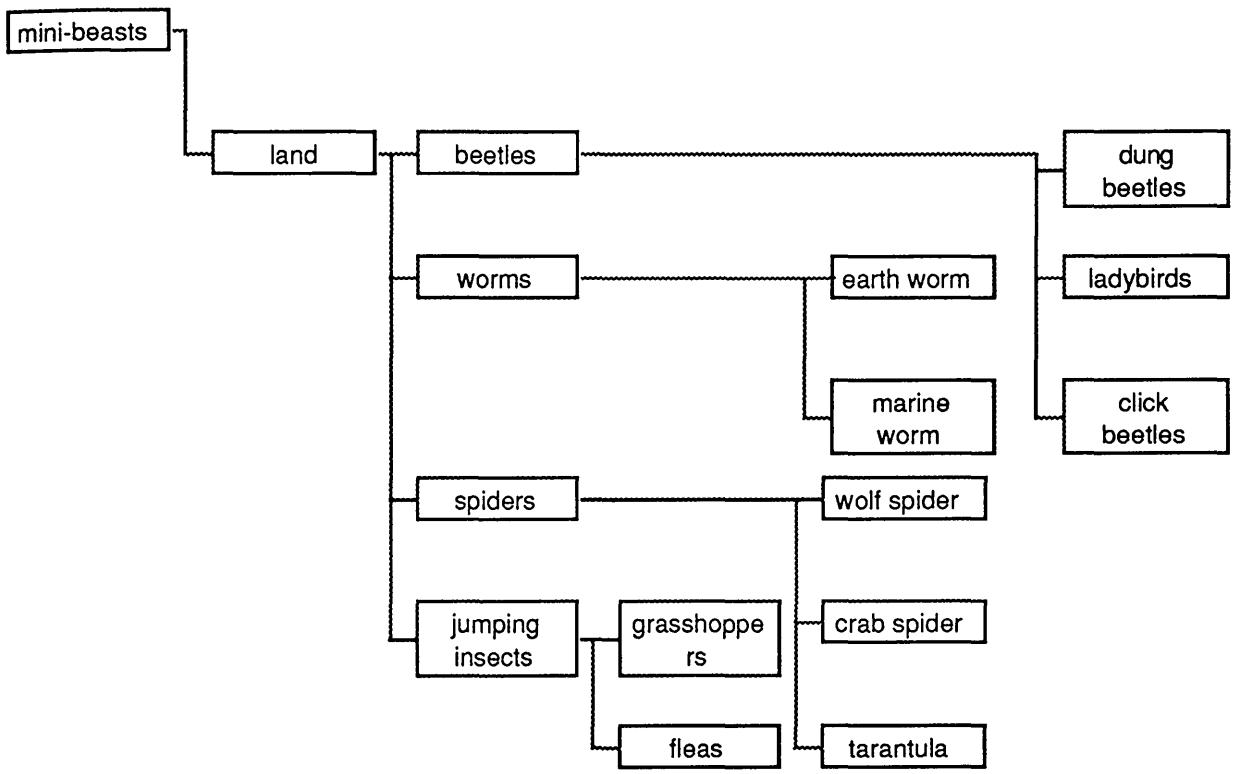


Figure 8.8. Ronnie and Bob's PCT model at 1415

1417 At the suggestion of an adult helper (TC) the boys invoke the Check tool.[14] A window appears which offers five suggestions on how the model might be improved. One of them is:

---

[12]The figure has been edited slightly to fit within the confines of a page.
[13]All Primex modelling tools use the up-thumb as a visual indicator of successful compilation of the model diagram.
[14]The anomaly checking tool described in section 5.3.3.2.

Give grasshoppers a feature to show in what way it is a special kind of jumping insects.

Ronnie and Bob decide to act on this suggestion by giving to grasshoppers the attribute 'has ears in its knees'. In the case of fleas they set this attribute to false. They also add the feature 'gives a painful itch' to fleas. These changes are made via the interview tool. The boys are evidently adept at clicking on the appropriate node and browsing through the laddering dialogue until a question is generated which allows them to supply the information that they wish to add.

1420 The boys reuse the Check tool. They pick out from among its suggestions the idea that marine worms should have a feature which distinguishes this class from earth worms. They add information to the model accordingly.

1423 An adult helper (TC) asks the boys to exchange tasks with another pair of children. Ronnie and Bob save the model and leave the scene.

8.4.2 Discussion

This episode shows that for Ronnie and Bob, PCT is a highly usable tool. They cope well with a task structure that is much more open-ended than any of those used in the large-scale trials described in the previous chapter. In half an hour the children have built a model that is larger than any of those constructed in the large-scale trials.[15] Their model provides a sound class hierarchy and its 'hidden layer' of feature information is quite extensive: 30 local features are defined, an average of almost two per class. Some of these features may owe more to the children's imagination than to the reference material provided, but since this material was new to them that is not too surprising.

---

[15]Ronnie and Bob's model contains 90 phrases and 32 distinct symbols. This compares to 40 phrases and 33 symbols for the largest model (Widgets) in the modelling course described in Chapter 7.

There is no doubt that Ronnie and Bob enjoy building the model. Their conversation is continuous and animated. Whereas Webb (1993) found that some novice users of the Expert Builder program devoted 48% of discussion to issues concerned with how to use the system, in this episode discussion is clearly dominated by the concerns of the domain. This may be because Ronnie and Bob are generally capable children and good collaborators. It may also be because the tool provides a good conceptual framework for the task and has an effective interface.

Certainly the children have almost no difficulties in making PCT do what they want it to do. They are especially proficient at starting and stopping the laddering interview at different nodes so as to develop the model according to their own plans. Ronnie and Bob are very much in control. This is clear evidence that our design for PCT substantially avoids the problems which we anticipated could arise in the use of contrived knowledge acquisition techniques (see sections 4.3.1 and 5.3.1.4).

The episode has several aspects that are relevant to a consideration of PCT's future development. First, we note that Ronnie and Bob make no use of the graphical editing tools: it is not clear however whether they were ignorant of their function or knew about them but just preferred to develop the model solely by interactive laddering. We implemented the graphical tools because we were uncertain about the method by which users would prefer to operate the tool. With more observation and discussion with users, this uncertainty might be removed. One possibility is that the graphical editing tools are mainly unwanted. If so, development effort could focus on methods of improving the laddering techniques.

Second, we note that the Check tool was effective in stimulating the development of the model but that the children did not invoke it until an adult helper at 1417 suggested that to do so might be useful. This is another context in which a domain-independent student modelling system (SMS), of the kind described above, might be able to provide occasional strategic help. It should be possible to embed the Check tool's functions within PCT; they could be called by the SMS and at judicious moments, the SMS could volunteer the suggestions that presently

can only be obtained on request. Research would be needed to find a procedure for identifying judicious moments for intervention.

Third, in describing feature information Ronnie and Bob always use propositions rather than expressions of the form `Attribute=Value`. As explained in section 5.3.1.4, the formative trials showed that the latter syntax caused difficulties and so we implemented a scheme in which propositions such as `Gather dung` were accepted and compiled internally into formulae such as `Gather dung=true`. Although this approach was useful to Ronnie and Bob, it makes for a rather verbose and inefficient model since propositions proliferate and at run time each proposition generates a yes/no question, not a menu question with multiple options. Related to this, we suspect that the tool makes it too easy for children to supply incidental features of classes, rather than prototypical features which differentiate between sibling classes. Some refinement to the dialogue strategy may be justified. It seems possible too that if PCT windows displayed feature information — with the option to turn it off for tidiness — then Ronnie and Bob would have been encouraged to be more thoughtful and concise in their description of the domain.

After the session, Dorothy Johnstone asked the boys to write down their impressions of the modelling tools. On PCT, Ronnie wrote:

> *This is really good, once you get the hang of it. Anyone could learn how to use this. It would be good for classifying animals and all sorts of things.*

Bob however preferred PDT. He wrote:

> *I found this easir to use because there was less typeing to be typed. This could be used to distinguish two look alike animals. I liked this one best.*

Neither boy liked the PKRL. Ronnie said:

> *This model was really boring. The only fun part was asking the questions. You could get some of the words mixed up and the computer would not understand it. You have to get used to this model before you use it.*

8.5 Julie and Angela

Like Ronnie and Bob, Julie and Angela worked as a pair after the interval. They were set the task of building a model that contained information about butterflies, using material from the book *Butterflies on my mind* by Dulcie Gray.[16] As well as being provided with this book, the children were given the names of seven butterflies that they should try to include within their model. In this case, the children were told that they were free to choose any of the four Primex model types. They were given no recommendation on which type might be best.

Dorothy Johnstone describes Julie and Angela as average pupils. Angela is extremely quiet. Julie is fastidious. Dorothy had noticed that in class, both girls had started hesitantly with Primex but later they had became more confident and enthusiastic.

8.5.1 Julie and Angela's models

1355 Julie and Angela arrive at the computer. There is a pause as they briefly review the task. 'So we can choose what kind of model?' says Angela. 'Oh, I like those .... classification trees', replies Julie. They immediately create a new PCT window and begin a laddering interaction.

1357 The tree's root has been given the label 'Butterflies'. In response to the system question:

Give some examples or subclasses of butterflies

— the girls slowly type in the names of the butterflies on the task sheet.

1401 The tree now appears as in Figure 8.9. Julie and Angela are asked:

Give some examples or subclasses of Silver Spotted Skippers

---

[16]Published by Angus and Robertson (1978).

They pause and consult each other, then click 'No'. The same happens for all the other butterflies on the tree.

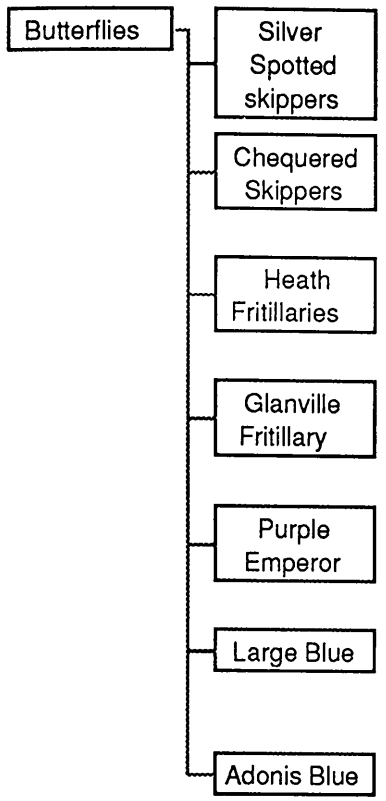


Figure 8.9 Julie and Angela's model at 1401

1404 The system asks:

Can you describe some feature(s) of Butterflies?

There is some debate about how this should be answered. Finally the children type 'They have colourful wings'.

1405 The system asks:

Can you describe some feature(s) of Adonis Blue?

The girls check the book before answering 'They are blue'. One girl wants to add 'They can fly' but the other disagrees and this information is not entered. The system next asks whether they can name some subclasses of Adonis Blue. The girls decline.

1406 The system asks whether the feature 'They are blue'

applies to Large Blue. The girls click Yes. The system then shows the dialogue box illustrated in Figure 8.10.

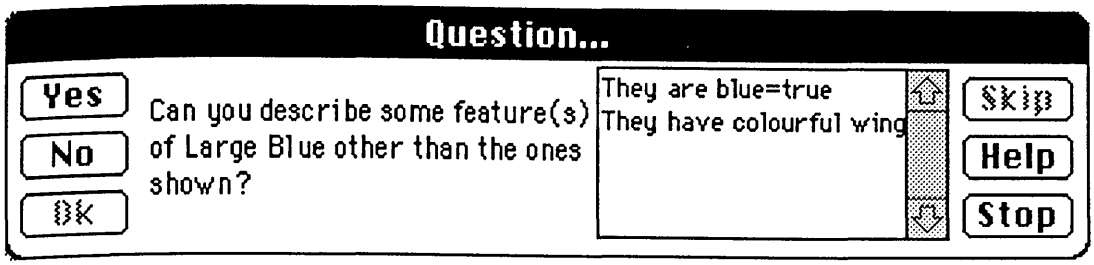


Figure 8.10. Julie and Angela's model at 1406

1407 A long pause follows. The dialogue is still on the screen. Julie says suddenly 'I don't understand this book. I honestly don't!'. She sighs loudly. 'Oh, this is a *hard* one!'. Her complaint attracts the attention of an adult helper (TC) who asks if he can assist. Julie explains: 'I don't understand it. They don't give you very many subclasses or examples in the book.' The adult helper expresses sympathy. 'Do you think that instead of a classification model, another kind of model would have been more suitable?'. Angela says 'Yes'. Julie says 'Maybe'. 'So do you want to switch to another kind of model?'. A pause follows. 'Don't do it because I'm saying it, but only if you think it's better', says the adult helper. 'I like the other ones better', replies Angela. Julie says nothing. The classification tree model is saved and the window is closed. The adult helper leaves the scene. Without further discussion, Julie and Angela create a new PDT window.

1411 Julie and Angela are unable to decide what to put at the root of the decision tree. They enlist the aid of an adult helper (HP). 'What sort of things help you to tell the difference between different sorts of butterflies?', asks the helper. 'Colour', says one girl. 'What they eat' says the other. 'Ok, so that's two questions you could ask. Now decide which one you want to ask first. That will give you somewhere to start, then you can decide what you want to ask next', says the adult helper, who then leaves the scene.

1413 Julie and Angela's decision tree appears as in Figure 8.11. However, neither girl seems confident that the approach is a good one. They talk quietly and indistinctly but it is clear that they are discussing whether to abandon the model. After a minute or so of conversation, they delete the decision tree model and immediately create a new window in PFT.

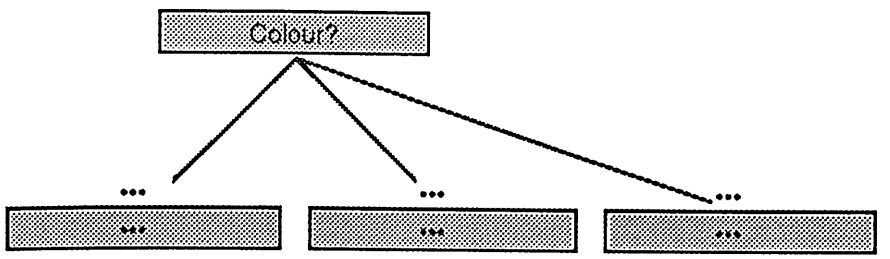


Figure 8.11 Julie and Angela's model at 1413

1419 The children's PFT window now shows the factor table illustrated in Figure 8.12. The children try to complete the leftmost column first. They have some difficulty with this because each time they click the Next button in the label edit dialogue (Figure 8.13), the cell that becomes highlighted is the one to the right of rather than the one below the current cell. So each time they have to cancel the dialogue and invoke the editor again from scratch.

Colours	Food	Habitat	Names
Blue			
Blue			
Purple			

Figure 8.12 Julie and Angela's model at 1419

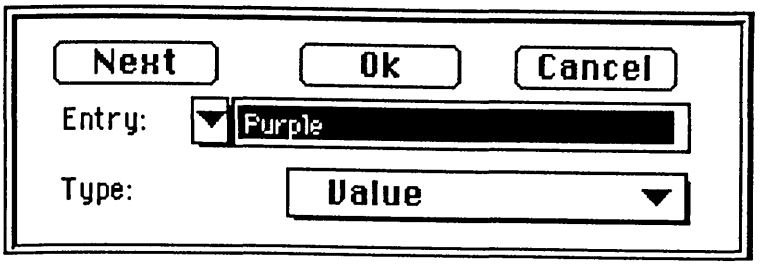


Figure 8.13 The label edit dialogue

1427 Having decided that it would be better to enter first the names of some butterflies, and then work backwards from there, the table now appears as shown in Figure 8.14.

Colours	Food	Habitat	Names
Blue	Wild thyme		Large Blue
Blue			Adonis Blue
Purple			Purple Emperor
			Silver spotted skipper

Figure 8.14 Julie and Angela's model at 1427

1437 The model has been developed to the state shown in Figure 8.15. Each entry has been the result of much discussion and consultation of the source material. At this point the adults call the class to order since it is time to go home. Julie and Angela save their unfinished model and depart.

Colours	Food	Habitat	Names
Blue	Wild thyme		Large Blue
Blue	Wild flowers	Chalk and limestone downs	Adonis Blue
Purple	Oak leaves	Top of oak trees	Purple Emperor
Mostly brown	Sheeps fescue	Chalk and limestone	Silver spotted skipper

Figure 8.15 Julie and Angela's final model

8.5.2 Discussion

Julie and Angela's experience gives us a lot to discuss. Below, we first attempt to interpret the process. Then we discuss some implications for system design and the practice of teachers. Finally, we report some views expressed by the children themselves.

8.5.2.1 Interpretation

As the children discovered, the source book does not classify butterflies hierarchically. It provides colour pictures of butterflies and textual descriptions of their features. The latter are somewhat scattered and the book lacks an explicit guide to butterfly recognition. Therefore Julie and Angela's task was by no means a simple one.

We should explain why we did not prescribe a representation. As noted previously in this thesis (see section 3.3), it seems likely that the ability to construct representations is generally important in problem-solving. From the large-scale school trials we found evidence that as a result of working with our modelling tools, children develop their representational skills. We considered that by observing children at work on a task in which the choice of represententation was left to them, insights might be gained into this process of skill development.

In hindsight, by asking the children to choose their own representation, with source material previously unseen and in a domain that they had not studied before, we probably asked too much. Nevertheless, the girls' tenacity is impressive. For over forty minutes they worked hard at the problem with only occasional adult help. After abandoning their first two efforts, they seemed in the final stages to be making real progress with their PFT model. One suspects that with a little more time, the PFT model would have been running satisfactorily.

Given the lack of hierarchical structure in the source material, the children's early choice of PCT was an unfortunate one. With PCT, the laddering interview asked Julie and Angela redundantly for subclasses and the more important feature

information is not made visible. It seems that this tool was selected on the basis that it was 'liked' by at least one child. Presumably for her an earlier experience with the tool had been enjoyable and perhaps successful. If we assume that Julie and Angela have not yet learned that different representations suit different domains, then theirs was a reasonable way to proceed.

In fact, although PCT is not ideal for the domain, the model could have been developed to completion in this tool. The girls' progress until 1406 was creditable. Why then did they abandon their PCT model? Julie's complaint at 1407 suggests that she was troubled by the mismatch between PCT's demands for subclasses on the one hand, and the non-hierarchical descriptions provided by the reference book on the other. The adult helper suggested a way to eliminate the mismatch: switch to a non-hierarchical form of representation. The children took the cue but their understanding of the justification for it is questionable. Angela's comment that 'I like the other ones better' is revealing. Julie, whose original idea it was to use PCT, expresses a reasonable doubt. Perhaps she recognises that the domain will be hard to understand regardless of what modelling tool is selected.

We do not know why Julie and Angela next selected PDT in preference to PFT or the PKRL. Unfortunately, this choice was possibly even worse than the original choice of PCT. The children knew of no procedure for classifying butterflies and to synthesise such a procedure from the book was a formidable task.

They quickly switched tool again, this time to PFT, and it is interesting that they did not consult the adult helper at 1411 on whether such a switch would be appropriate. Certainly she did not cue them to change tools. We suggest that the children, influenced by the previous adult intervention, have discovered a new tactic: when in difficulty, change tool. Although their present grasp of the properties of the various representations may be weak, the moment potentially constitutes a breakthrough. A willingness to explore different representations of a domain, if practised repeatedly over multiple domains, could conceivably develop an understanding of these properties as well as perhaps valuable habits and skills in reasoning and problem-solving.

In fact, there is some support for this idea in previous

research. In their study of how adult students (undergraduates) tackled analytical reasoning problems, Cox & Brna (1995) found that students did better at problem solving who fully understood the representations they used and it was helpful for students to have a wide conceptual repertoire of representations. Cox & Brna observed that switching of representations was relatively common during their construction, especially with more difficult problems. They suggest that subjects switched as an adaptive response to an awareness that their initial selection of representation has been poor. Although switching was associated with less rapid problem-solving, because of the time required to reconstruct the information in the new representation, it is an important tactic for resolving impasses in reasoning.

That the Primex tools — except the PKRL — all feature in Julie and Angela's episode is encouraging. It suggests that a productive process of exploration of representations may be enabled by these tools, even with children aged ten.

8.5.2.2 Implications

As in the case of Duncan's PDT model discussed in section 8.3.2, we note that the shortcut which we implemented in the PFT edit dialogue via the 'Next' button (shown in Figure 8.13 above) does not match the way that Julie and Angela prefer to work. A dialogue like the one shown in Figure 8.16, with arrow buttons to allow traversal of the table in any direction, might be more flexible. Designs along the lines of this one should be tested to see whether they are improvements in practice.

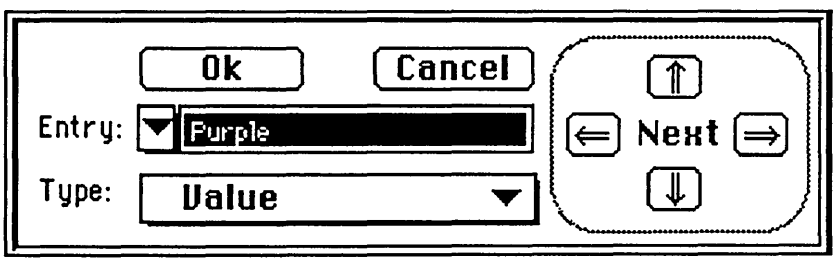


Figure 8.16 An alternative edit dialogue

Cox & Brna (1995) suggest that to help students to understand problems, intelligent learning environments might

provide a 'problem summary' window into which the student would post information. This might encourage the student to spend more time on the problem comprehension stage and facilitate self-explanation. Domain comprehension was a problem for Julie and Angela and it seems possible that they would have have been assisted by (say) making informal notes prior to selecting a Primex model type.

In most cases however, we do not expect that Primex modellers will be faced with completely novel domains as Julie and Angela were. We expect that children will have learned a good deal about a topic before they are asked to build Primex models: probably they will have previously constructed at least some informal representations, for instance by classroom writing or paper-and-pencil drawing. Even so, a problem summary window may provide a helpful further aid to comprehension at a low cost in implementation. Conceivably also, the system could make some practical use of the information, for example to prefill menus (although this would require some formal structuring of the summary window's contents, which to some extent may defeat their purpose).

We want children to learn how to match representations to domains, but we do not wish them to flounder excessively as a result of making a poor choice. Teachers should be prepared to intervene at the right moment: neither too early (lest children lose the experience) nor too late (lest they waste time in frustration). Furthermore, teachers should judge when children are ready for more open-ended modelling tasks.[17] It is not likely that all children of a given age will be equally ready to model the same domains with the same tools. Cox & Brna (1995) found large variation between their adult subjects in their preferred representations and suggest that a source of this variation may be individual differences in cognitive style, especially on a 'visualiser/verbaliser' dimension. They point out that this militates against prescriptive advice on how representational skills should be taught and learned. We expect that children too

[17]We do not claim that the task set for Julie and Angela was a model of good practice, and neither do we make such claims for the adult interventions that occurred during their session.

will differ in cognitive style, and teachers should be sensitive to these in setting tasks and in making interventions.

Might a student modelling system (SMS) based on the domain-independent (discussion-level) approach mentioned previously be useful to help students select or reselect an appropriate Primex representation for a domain? We are doubtful. The selection of a good representation depends on the features of a domain. Also, as just noted, the importance of individual cognitive style and prior knowledge should be recognised. Not least there is the difficulty of knowing when and how to intervene.

It seems more feasible that a system could offer some help *after* the child (perhaps with a teacher's guidance) has decided to switch to another representation. We notice that each time Julie and Angela switched tools, their previous models were lost without trace. It should be mainly straightforward to retain the names of classes, attributes and values when a student switches from one tool to another so that these names do not have to be retyped in the second model but can be selected from menus instead.[18]

More ambitiously, it may be possible to automatically construct a second representation from a first model created by the child. Cox & Brna (1995) refer to this as co-construction and recommend that it should be provided as a selectable option for the user. As mentioned in section 4.5.2, we did in fact consider co-construction in designing the Primex tools. We rejected it due to the implementation effort that seemed to be required. Also, there are some risks: it hardly seems likely that the cognitive benefits of an automatically constructed model will be the same as one that is constructed manually and the difference may not be to the child's advantage. This must be a matter for future research.

8.5.2.3 Children's views

As mentioned previously, following the session Dorothy Johnstone asked the children to write down their impressions of

[18]The more difficult cases are when the first model is developed in PDT or (worse) the PKRL, since these tools do not isolate the relevant names.

the modelling tools. On PCT, which had been her original choice for the Butterflies model, Julie wrote:

> *The classification tree is quite complicated and sometimes it gets mixed up and asks funny questions. It can only be used for some subjects like birds and fish.*

Angela is similarly ambivalent, mentioning that PCT 'is good for grouping animals and birds together'. It may be reading too much into these comments to detect a developing perception that PCT is more suited to some domains than others.

Certainly no such perception can be discerned in their comments about PDT. Although the girls did not succeed with this tool in the Butterflies session, they review it favourably:

> *I thought the decision tree was best of all ... The most enjoyable thing was having to get it all built up then seeing if it worked.* (Angela)
>
> *The decision tree is by far my favourite ... The reason I liked this was because there was not much typing involved.* (Julie)

Both children regard PFT as effective, but perhaps rather dull. For example, Julie wrote:

> *The factor table was a quick way of doing things but not as much fun as the decision tree.*

Julie and Angela are not much given to making adverse comment. The closest they come to criticism is when they write about the PKRL:

> *I thought it was good because the computer asked you questions and it was good when you got other people to see if they could guess what you had written. There was one thing I didn't like. That was: it was annoying when you didn't get it to run because you had the If's and the And's in the wrong place.* (Angela)
>
> *The rule based model language was quite complicated to understand but once you got used to it it was okay.* (Julie)

8.6 The views of the class teacher

As mentioned previously, the children's class teacher, Dorothy Johnstone, was an experienced Primary school teacher and also a student on the M.Ed. programme in Computers and Learning at Moray House Institute. As part of her M.Ed. she had participated in a course in knowledge-based modelling. The course was taught by the researcher and comprised eight three-hour classes followed by an assignment based on the development of a practical classroom application of knowledge-based modelling. Appendix 20 contains a copy of the assignment specification, the criteria for which indicate the course coverage.

For her assignment, Dorothy chose the topic 'The Digestive System and Diet'. This topic appears in the Environmental Studies area of the 5-14 curriculum. Prior to the assignment, the topic had been taught in Dorothy's school with a variety of methods including whole class teaching, group assignments, individual research work, TV programmes and various computer software packages. Dorothy wrote:

> *Although the present approach works, it is lacking in any real challenges to children's thinking power. Introducing Primex will hopefully provide an opportunity to fill this gap. Having used Primex at college, I know it can be challenging but stimulating too.*

Dorothy's introduction of the Primex tools was by way of an extension to, rather than a replacement for, the existing methods. She designed activities in which children towards the end of the topic built models in groups:

> *I would hope to use Primex as a means of revision and consolidation of the knowledge gained on diet and digestion. By building models in groups the children would be working collaboratively and gaining skills in planning, self-monitoring and evaluation.*

Dorothy's approach to introducing the Primex tools was influenced by a strategy proposed by Mary Webb (Webb 1994). In this strategy, children first explore an existing model, and then

extend or adapt an existing model, before building models of their own (working in groups). Also, children initially work with simple and familiar domains before attempting models in more complex or less familiar domains. Accordingly Dorothy created some simple PKRL, PFT and PDT models on the subjects of 'Eating places' and 'Picnics' which she demonstrated to the class. PCT was not used because Dorothy considered that it did not suit her domains. Children first used and later extended her models. Later they built models for domains that she designed: for example, one task was to build a PFT model which identified a range of mixed sandwich fillings as suitable or unsuitable for a variety of special dietary requirements (e.g. vegan, Jewish, vegetarian, allergic to dairy foods, etc.).

To summarise her experience with the tools, Dorothy wrote about the benefits and limitations of each. Her words are reproduced verbatim in Table 8.2.

Tool	*Benefits*	*Limitations*
PKRL	Children shared ideas and were able to plan as a team. When running models they were able to see mistakes and rectify them. Models could be used by others and built upon.	Lack of keyboard skills a drawback. Children became bored quite quickly. Frustration crept in when models wouldn't run.
PDT	Careful planning involved, higher level of thinking taking place, groups collaborated well, children motivated, enthusiasm shown by every child. Good graphics, and tools easy to use, especially the Tidy up tool which is quite a time saver. A model can be based on an existing one which saves typing.	Would agree with Doug Urquhart's comment (DAI Research Paper No 790, Feb 1996, Conlon & Pain): 'The Compare tool has limited use and would be more helpful if it could intervene intelligently during the tree construction process'. Models were difficult to see as they grew larger.
PFT	Easy to use. Less typing involved because of 'Next' facility. Quicker to build table compared with tree. Similar properties to above, e.g. planning, collaboration, higher order thinking skills, children motivated.	Boxes for typing in limited in size. Doesn't give an immediate picture of relationships like the tree above does.

Table 8.2 Dorothy Johnstone's summary of benefits and limitations

Dorothy's main reservation about knowledge-based modelling was her belief that many class teachers would need a

lot of support to make use of tools such as Primex. She recommends the provision of courses and packages of teaching materials. In conclusion, Dorothy wrote:

> *I think there is a place for knowledge-based systems in the primary schools. It has an enormous potential for developing children's thinking and learning skills.*

8.7 Summary

Generally speaking, it seems fair to say that these small focussed studies confirm the finding of the large-scale trials that the new modelling tools are quite usable. Yet children building models with them *do* experience difficulties, and the particular contribution of this chapter has been to offer some insights into why they happen and what might be done about them. Some of the difficulties can be addressed by still better tool design. Others are better addressed by looking to the wider classroom environment — and especially, to the teacher and to the model builder's classmates. Up to a point, of course, some difficulties are inevitable and even necessary. If there were none then model building would not offer the 'real challenges to children's thinking power' that Dorothy Johnstone identified as its main appeal.

The main hypothesis of this thesis is that the new modelling tools enable more successful model building by children than has been possible with the previous rule-based shells. The large-scale trials provided one way to test the hypothesis. Detailed observation of Duncan as he developed his models in PDT and the PKRL has now provided us with another. Although Duncan struggled on both occasions, it seems clear that PDT offers him a conceptual framework that he can understand. This tool constrains his representation in helpful ways. The PKRL's blank text window offers Duncan no such constraints and the rule language is to him an alien formalism. Duncan knows which tool he prefers.

The study of Ronnie and Bob gives a very useful picture of the worth of PCT. From the evidence of the large-scale trials, this tool's evaluation was mixed: generally, children rated it as relatively difficult to use. Here however we saw two ten-year olds

using PCT with ease. In a highly successful collaboration, they manipulated the interaction in a way which largely dispelled our fear that the laddered grid interview technique might assign too passive a role to the model builders. PCT is Ronnie's favourite tool.

How can the evidence gained by observing Ronnie and Bob be squared with that of the large-scale trials? We cannot be sure. Conceivably PCT's effectiveness depends upon high-achieving children or a collaborative approach. Perhaps the tool's novel features (e.g. hierarchical representation and indirect model construction) unsettled teachers who then communicated their unease to children. Whatever the explanation, Ronnie and Bob have shown that PCT justifies further development and evaluation.

The emphasis of this research has been on the representation of domains with particular formalisms. However, Julie and Angela's experiences remind us that when the modelling task is specified in a very open-ended way, the stages of domain comprehension and the selection of a suitable representational form may also be highly problematic. Admittedly, these children's situation was extreme: they were given difficult and unseen material relating to a domain that they had not studied, without advice on which tool to apply. Like Dorothy Johnstone, we expect that modelling will be more often used by teachers as a means of revision and consolidation of existing knowledge. What was most interesting about Julie and Angela's episode was their manner of selection and in particular their switching between tools. Their first instance of switching was prompted by an adult but the second was not. That the children felt able to construct three (albeit partial) models in different representations, in a forty minute session, is encouraging because it suggests that the tools have achieved a level of usability sufficient to provide support for learners in exploring the suitability of different representations for a domain.

Observations of children in these focussed studies have led to many suggestions about how the modelling tools could be improved. Many improvements could be made almost immediately, such as the redesign of the edit dialogues in PDT and PFT to give support to the ways in which children seem to prefer

to work. Others, such as our suggestions for domain independent student modelling systems that can offer strategic help along the lines of the proposal by Cumming and Self (1990), clearly require extensive further research.

Finally, we have been able to offer (or reflect) some suggestions about how teachers might use the tools in practice. These suggestions however place a considerable responsibility on teachers. For example, if teachers are to help children effectively to select good representations for a domain, as we proposed that they should, then clearly teachers will have to have this knowledge themselves. Dorothy Johnstone anticipates that the preparation of teachers will be the main barrier to their adoption of knowledge-based modelling as a classroom activity. Her own example suggests that the problem can be solved for individual cases, but a general solution will certainly be more difficult.

Chapter 9 Conclusion

This chapter summarises the results of the research and identifies some of its limitations. We then revisit the main research hypothesis and evaluate it in the light of our main findings. We compare the present research with two other related projects and identify the main original contributions of our research. Areas for future research are then described. Finally, we discuss the general prospects for knowledge-based modelling in education. Change in education is complex and influenced by many factors. Barriers to the uptake of new tools include the requirement for staff development and the need for additional research in teaching and learning. However, computer modelling in education seems to be at least partly in tune with some influential trends and developments. The strong focus within our research on teachers and children should ensure that our tools and findings are realistically grounded in classroom practice. The present thesis is seen as making a modest contribution to realising the enormous educational potential of knowledge-based modelling in education.

9.1 Summary of results

The main hypothesis of this research was stated in Chapter 3, where we also identified the main research questions that would be investigated in an effort to test the hypothesis. The hypothesis was:

> *... knowledge acquisition systems will enable more successful model building by children than has been possible with the EMYCIN-type shells.*

In the following sections we summarise the results and limitations that have been reported in previous chapters for each main research question.

9.1.1 Application area

In section 2.8 we observed that successful knowledge

acquisition systems are typically limited in their application area to a narrow range of tasks or domains. Therefore the development of successful educational modelling tools based on knowledge acquisition technology seemed to necessitate making a restriction in scope. The question was asked:

Which application area (task or domain) is suitable?

We addressed this research question in section 3.7. We collected evidence from teachers and curriculum documents which identified classification as important from an educational perspective. From KBS sources, tasks in classification were found to be tractable in a technical sense. Accordingly, classification was identified as a suitable choice of application area.

Although other application areas may also have been good choices, the suitability of classification has been confirmed by our research. We designed computer tools (Chapters 4 and 5) on the assumption that they would be used to build classification models. All of the models that featured in the large-scale trials (Chapter 7) and the focussed studies (Chapter 8) are of this type.

The commitment to a single application area was generally beneficial to this research. It focussed attention on applicable ideas and techniques of knowledge acquisition. It constrained design choices enough to prevent research effort from being too thinly spread, but not so much as to exclude interesting alternatives. It enabled a meaningful comparison between tools: for example, our definition of correctness in Chapter 6 depends on the assumption that models represent classification structures. It also made feasible certain techniques, such as the compilation of models into a 'normal form' as described in section 4.5.3, that were of great practical importance in the implementation of tools.

Nevertheless, the commitment to classification implies a limitation: our tools may not be flexible enough to build some of the models in which schools might be interested. In general, PDT, PFT and PCT are ill-suited to building models of other kinds of knowledge-based task (design, configuration, planning, or scheduling, for example). We did not attempt to investigate these non-classification tasks. Neither did we explore the idea that some

forms of tool design might benefit from commitment to an individual domain (see section 3.7.1).

9.1.2 Knowledge representation and acquisition

The question was asked:

What representation formalisms and knowledge acquisition techniques are available for the selected application area?

This question was addressed in Chapter 4 for the selected application area of classification. From education sources we found evidence that representations based on decision trees, classification hierarchies, and tables are useful for classification purposes, although they are not necessarily used by teachers in a systematic way. From KBS sources we distinguished three methods of classification problem-solving: heuristic classification, simple classification and systematic refinement. Among knowledge acquisition techniques, we identified as potentially applicable the contrived techniques, induction, and knowledge editors. These findings led to the outline specifications that were presented in section 4.4 and which subsequently were implemented, mostly successfully, as the tools PDT, PFT and PCT.

As discussed in section 4.2, we decided against attempting to give explicit support for heuristic classification. This problem-solving method is undoubtedly powerful. We reasoned however that simpler methods would suffice for most classroom purposes and that domain-independent tools for heuristic classification would probably be usable only by knowledge engineers.

It should be conceded that PDT, PFT and PCT are not well suited to building complex classification models. This is not only because of the lack of explicit support for heuristic classification. Even some tasks that would be regarded (e.g. by Clancey 1985; Tansley & Hayball 1993) as simple classification tasks will be impossible to specify with our tools, since they lack variables and arithmetic. Without the latter, some forms of abstraction are impossible. However, this was never raised as an issue by any of the teachers or students who took part in any of the trials. We do not think that it is likely to be a serious problem in the near

future.

Of the contrived knowledge acquisition techniques, we explored only laddered grids. Other techniques, such as card sort and repertory grid, were not investigated. As we observed in section 4.4, this part of our research uncovered what seemed to be an embarrassment of riches. It is quite possible that very good tools could have been constructed with the techniques that we did not explore.

9.1.3 Methodology

The question was asked:

What should be the approach to the development of tools?

Together with Helen Pain, we developed the Persistent Collaboration Methodology (PCM), a hybrid of user-centred design with action research. The PCM, which was described in section 3.6, is intended to provide a helpful framework for projects in applied AIED. It emphasises three specific dimensions of the development process: the importance of collaboration between teachers, researchers, and technologists; designs that are developed iteratively by formative testing in authentic user contexts; and a dialectical relationship between theory and practice. Our research was guided by PCM but the methodology should also be of wider interest.

As mentioned previously, the Persistent Collaboration Methodology requires development and independent evaluation through a range of other projects. Some limitations of the PCM were discussed in section 3.6.3. In addition to the problems cited there, it might be noted that we were not always able to realise the PCM ideal of utilising authentic classroom contexts — the focussed studies reported in Chapter 8 were situated within a college environment. Also, the episode cited in section 5.3.3.1, in which we abandoned the systematic refinement method of classification in running PCT models following teachers' feedback, shows that collaboration is not guaranteed to produce the best results in the first instance.

At a more technical level, we successfully used Extensible Primex (see section 3.8.1) as an experimental testbed for the development of new tools. The complexity of a large software engineering task was successfully managed with the aid of the language Prolog++ which was used throughout the project. We claim that by implementing PDT, PFT and PCT in Prolog++ as extensions to Primex, a shell already established in schools, we helped to secure teachers' acceptance of the tools and facilitated their rapid development. However, although Extensible Primex and Prolog++ served as effective experimental tools, they are less than perfect as a basis for production (i.e. release) versions of the new modelling tools. Especially if all the (numerous) software extensions are installed, the Primex system is large, takes a long time to load, occupies a lot of memory, and sometimes seems slow.

9.1.4 Design principles

The question was asked:

What main principles and issues should be addressed in the design of tools?

Leaving aside aspects related to our use of PCM and Extensible Primex, we identified and pursued in Chapters 4 and 5 three general principles that seem to be of central importance. First, modelling tools should offer children forms of representation with which (ideally) they are already familiar, or which are readily accessible to them otherwise. Second, these forms of representation should be ones which provide helpful conceptual frameworks for building models. Third, tools should provide convenient and transparent ways of building and interpreting representations.

We addressed all of three principles in the design of PDT, PFT and PCT. From the evidence of the pre-tests in the large-scale trials (section 7.6) and the focussed studies (Chapter 8) we claim that the forms of representation which we selected for these tools generally are known by or accessible to children aged ten and above. That our tools implement helpful conceptual frameworks and mainly good building environments is confirmed by the

overall high quality of models that children have constructed with them in the large-scale trials (section 7.2) and by the feedback that has been obtained from children and teachers (sections 7.4, 7.5 and 8.6). The evidence suggests that the runtime interpretation of PDT and PFT[1] models is suitably transparent, although the problem mentioned above with PCT's interpretation requires to be fixed.

Three other, more specific principles were advocated in section 4.5. First, a strong family resemblance between modelling tools is useful to support comparative evaluation and to assist children to learn the tools. Second, to help children to build large models and to work collaboratively it should be possible to combine models. Third, it is advantageous to use a consistent compilation scheme for all modelling tools. We successfully pursued these principles in the development of PDT, PFT and PCT, as summarised below.

(i) Family resemblance was achieved through the methods described in section 4.5.1. Although we cannot be certain how important this factor has been, we think it reasonable to suppose that had the tools been given markedly different interfaces then children would not have been able to construct so many models of different types in the large-scale trials (see Tables 7.3 and 7.4 in Chapter 7).

(ii) We developed and implemented techniques whereby models can be combined. These techniques, which are reported in sections 5.1.2, 5.2.2 and 5.3.2, were used only lightly during formative tests and not at all in the large-scale trials. However, there are indications that the ability to link models could be truly helpful to children who have become a little more familiar with the tools.[2]

---

[1]With the naïve compiler, which is the default setting for PFT.

[2]Following the focussed studies, Dorothy Johnstone (the teacher of the children involved) continued to use the Primex tools with her class. She sent to the researcher a printout of a PCT model of 'Minibeasts' developed by a collaboration of Bob, Duncan and Ronnie. The model, the largest we have seen from children with the new tools, defines 79 classes of insect. Its diagram fully covers two A4 pages which are printed lengthwise and joined with glue. This model has been constructed as a single tree. We consider it

(iii) We designed and implemented a three-stage compilation scheme which is common to all models. As described in section 4.5.3, all models are compiled to code which interprets the user's model as a classification expert system. An interim stage in compilation is the production of a 'normal form' that is useful in model analysis. Our scheme decouples the modelling tools from the Primex host shell, which is advantageous for the development of further tools as Primex extensions and also for the future portability of tools to other environments. A disadvantage is that compilation can be slow, especially with larger models on less powerful computers. In production (i.e. release) versions of PDT, PFT and PCT it may be possible to speed up compilation by bypassing the stage of generation of PKRL code.

Among other issues which we investigated was the potential for automatic or semi-automatic analysis of models. We designed and implemented the PMA to analyse the quality of the models that learners construct and the PMC to enable two learners' models to be compared (see Chapter 6). Such tools exploit the formal properties of knowledge-based modelling systems, including the fact that models can be reduced to normal form and the applicability of simple techniques in automated reasoning. Although the PMA and the PMC both have limitations, our evaluations of these tools demonstrate that semi-automatic analysis of models is possible and has considerable potential to enhance the usefulness of modelling tools.

9.1.5 Time required for model building

The question was asked:

How much time is required by children to build a model?

We showed in section 7.3 that the overall mean build time for all models and domains within the large-scale trials was less

---

highly likely that these children are ready to learn about model linking and would recognise the benefits of the technique.

than 14 minutes, suggesting that a typical school period could accommodate smale-scale model building activities. In the flat-structured domains, we found that models were built more quickly with PFT or PDT than with PCT or the PKRL. Within the focussed studies, the example of Ronnie and Bob showed that in half an hour some ten-year old children are capable of building with PCT a moderately large hierarchical model with unfamiliar material.

9.1.6 Children's difficulties with tools

The question was asked:

What problems do children have in using the tools?

First, taking a broad view, building a model of a domain requires understanding the domain, selecting a representation, and constructing a model. The example of Julie and Angela in the focussed studies (section 8.5) indicates that in open-ended tasks especially, children may experience difficulties in all three stages. We expect however that modelling will be used by teachers mainly as a means of revision and consolidation of knowledge. Teachers will use other methods and resources to help children to understand domains initially. Selecting representations is a skill that must be learned. Although using the modelling tools can help children to develop this skill, they will often need support. We expect that in many cases the selection of a representation will be made by the teacher.

Second, taking a narrower view of the question, we have identified numerous specific problems in using the tools to construct a model. For all tools there are difficulties associated with the requirement to construct correct syntax, understand the semantics of the representation, and express domain relationships effectively within the tool's conceptual framework.

Some of these difficulties could be alleviated by making further improvements to the tools. Table 9.1 summarises the relevant proposals which have appeared previously in the thesis (but does not amount to an exhaustive list). We distinguish three

broad categories of change: changes that are straightforward enough to be made immediately (labelled 'I'); those that require further analysis and appraisal (labelled 'A'); and those that justify more extensive research (labelled 'R').

Tool	*Development proposed (I=Immediate, A=Appraise, R=Research)*
All	1 Creation of new model windows to be via a dialogue accessed by the File menu's 'New' command (I) 2 Improved error messages (A) 3 A 'discussion layer' Student Modelling System for strategic help (Cumming & Self 1990) (R) 4 A 'problem summary' window into which the student would post information about the domain (Cox & Brna 1995) (A) 5 Support for constructing automatically a different kind of model, based on information in the current model (R) 6 'Save' command to offer a choice of save actions whenever more than one kind of model window is active (I) 7 'Undo' command to be extended to graphical editing (I)
PDT	8 A more tolerant algorithm for interpreting tool clicks in a model window (I) 9 Support for breadth-first in addition to depth-first development of the tree (I) 10 Redesign of edit dialogue to highlight the significance of the choices offered by its radio buttons (I) 11 Review of the current diagram syntax conventions (A)
PFT	12 Redesign of edit dialogue to permit traversal of the table in any direction (I) 13 Give table's rightmost column a distinctive appearance to indicate its special status as the decision column (I)
PCT	14 Improved criteria for determining when the first pass is complete while laddering with the 'Ignore features on first pass' setting (A) 15 Tool for optionally displaying feature information in the model diagram (I) 16 Refinement to the laddering dialogue strategy to elicit an improved quality of feature information (R) 17 Interpretation of models to be based on systematic refinement (A)
PKRL	18 Revision of rule syntax (A) 19 Structure editor for entering text of rules (A) 20 Compiler to optimise naïve and inconsistent code (A)

Table 9.1 Proposals to address children's difficulties with tools

Although all of the tools require development, it seems fair to say that children at present generally experience more serious difficulties in learning to use the PKRL than they do with PDT, PFT or PCT. The evidence for this comes from the observations of individual children (Chapter 8), feedback from teachers (sections 7.5 and 8.6), feedback from children (section 7.4 and Chapter 8) and the indirect evidence of the quality of models in the large-

scale trials (section 7.2). We have related the greater usability of the knowledge acquisition tools to the fact that they present children with more familiar or accessible forms of representation and provide conceptual frameworks that are helpful for building classification models.

9.1.7 Model quality

The question was asked:

How correct, efficient, and concise are the models?

Having defined these measures formally in section 6.1, we were able (with the aid of our semi-automatic model analyser, the PMA) to apply them in section 7.2 to the models built by children in the large-scale trials. Here we summarise only the most important comparative results. We found some domains in which PFT and PCT models proved superior in correctness to PKRL models, and none in which the converse held. In all domains, PFT and PDT models were superior in efficiency to PKRL models. In terms of conciseness, PKRL models were comparatively good; however we did not identify any domain or model type for which lack of conciseness indicated a serious problem.

9.1.8 Children's attitudes

The question was asked:

Do children enjoy and feel confident in using the modelling tools?

Pupils' attitudes towards the tools were elicited by questionnaire in the large-scale trials (section 7.4). Attitudes were generally positive but to different extents for different tools. Children generally indicated greatest enjoyment and confidence in PDT and PFT and least in PCT and the PKRL. In the focussed studies (Chapter 8) children's comments were mainly very positive about all tools except the PKRL, which was regarded as difficult to use.

9.1.9 Teachers' attitudes

The question was asked:

Do teachers regard the modelling tools as successful?

Questionnaire data from teachers involved in the large-scale trials was discussed in section 7.5. Teachers generally believed that children enjoyed and understood PDT and PFT and that children built high quality models with these tools. They were less sure that the same was true of PCT and the PKRL. Teachers were keen to use all tools again but their enthusiasm was least for the PKRL. The teacher of the Primary school class involved in the focussed studies, who had considerable experience with the tools, concluded that there was '... an enormous potential for developing children's thinking and learning skills' (section 8.6). She regarded the PKRL as having the most serious limitations.

9.1.10 Learning of representations

The question was asked:

Do children learn the representation formalisms that are embedded in the tools?

In the large-scale trials we used pre-tests and post-tests to assess children's ability to select and construct paper-and-pencil representations of specified information (section 7.6). The pre-tests revealed that the representational forms upon which we based the design of PCT, PFT and PDT were already fairly familiar. From the post-tests we discovered that children had become more focussed on these forms and that their use of them had risen in quality, showing evidence that representational skills had been learned. From the focussed study of Julie and Angela (section 8.5), we suggested that a productive process of exploration of representations may be enabled by the tools even with children aged ten.

9.2 Evaluating the research hypothesis

We can now review the original hypothesis. This was that:

... knowledge acquisition systems will enable more successful model building by children than has been possible with the EMYCIN-type shells.

In section 3.3 we declared our intention to judge 'successful model building' by using measures of four different kinds: process, product, attitude and learning outcome. Table 9.2 summarises some of our main research findings and shows that in important respects, the PKRL compares unfavourably to one or more of the new tools. We take this as fairly strong evidence in favour of the hypothesis.

Type of measure	*Summary of finding*	*See section*
Process	PKRL models take longer to build than models in PDT and PFT	9.1.5
Product	PKRL models are less correct than PFT and PCT models (in some domains)	9.1.7
	PKRL models are less efficient than PDT and PFT models (in all domains)	9.1.7
Attitude	Children are more positive about using PDT and PFT than they are about using the PKRL	9.1.8
	Teachers' enthusiasm towards the tools is at its least for the PKRL	9.1.9
Learning outcome	Children experience more serious difficulties in learning to use the PKRL than they do with PDT, PFT or PCT	9.1.6

Table 9.2 Evaluating the hypothesis with the research findings

However, the limitations of the research provide important qualifying conditions. For example, we have represented classes of tool with specific designs; we have confined our attention to a single application area; we have tested the tools with particular groups of children; and we have ignored some kinds of measure of success, such as domain learning. Therefore our research constitutes only a limited demonstration that the hypothesis is valid under certain conditions. In any case, as we explained in section 3.5, an absolute, incontrovertible validation of the

hypothesis is impossible. The success of any tool depends on the context in which it is used and educational contexts are infinitely variable; as are the philosophical stances of those who work in the field and who would each bring their own measures of success. The measures which we have adopted will not be universally acceptable. However, by making our measures explicit, together with the methods and data of our research, we have at least provided the basis for an informed discussion of our conclusions.

9.3 Related work

Below we compare our research to two other related projects, each of which has already been referred to earlier in the thesis. Both projects sought to design educational modelling tools using ideas drawn from research in knowledge-based systems.

9.3.1 Valley

Valley (1992) describes the development of an expert system shell for educational use. Her shell, named EXPLORES and implemented within the MS-DOS operating system, provides a representation language based on a frame-like structure which she calls a 'thing'. She gives the example of a thing shown in Figure 9.1.

```
Name:          spiny cactus

Has:           milky sap in stem = no
               spines = yes

Does:          grows best in May = yes

Relations:     has member => column cactus
               has member => goats horn cactus
               has member => rats tail cactus
               has member => peanut cactus
```

Figure 9.1 A 'thing' from Valley's EXPLORES shell

In this example, the thing called `spiny cactus` has features described by the `Has` and `Does` sections. The `Relations` sections

identify class membership relationships that at runtime are used to propagate inherited features through the model: `has member` is a predefined relation, as is `is an example of` which is its converse. In addition to things, the language also provides production rules. A template editor is provided to enable models to be created.

EXPLORES has some similarities to our own PCT. Like EXPLORES, a model in PCT is hierarchically structured from classes that define feature information. A difference is that a hierarchy in EXPLORES is restricted to singular inheritance (Valley 1996, personal correspondence). As discussed in section 5.3.1.3, we take the view that multiple inheritance is more expressive and can be implemented in a way that avoids some potential pitfalls.

Perhaps the most prominent feature of EXPLORES is that the system provides the user with an extensive set of question templates for querying a model. For example, the user can ask 'How are spiny cactus and column cactus related?'. The construction of such a question is supported by a menu-driven interface. For each question template, EXPLORES has a technique for answering the question automatically using the current model.

The question-answering techniques of EXPLORES are certainly interesting and in fact they influenced the development of PCT. As described in section 5.3.3.4 we implemented one of the techniques in PCT, together with a variation of it which seemed worthwhile. To incorporate the other EXPLORES techniques within PCT would be straightforward.[3] Indeed, because PCT unlike EXPLORES provides a graphical display of the model hierarchy it offers a superior environment for exploiting these techniques. Questions can be constructed partly by clicking on the model diagram and the diagram can help users to understand the answers.

A more important difference however is PCT's use of knowledge acquisition techniques, specifically laddering, to assist the user to build the model. Although PCT provides graphical

---

[3]That we did not do so reflects our balance of emphasis, which leans towards building rather than interpreting models. Recalling the distinction made in section 2.5 between the 'exploratory' and 'expressive' modes of learning with computer models, it could be said that Valley's research mainly relates to the exploratory mode whereas ours has prioritised the expressive mode, reflecting our main research hypothesis.

editing tools whereby classes can be created and feature information edited directly, we noticed that users generally preferred to construct the model by laddering. We suspect that the EXPLORES language and template editor does not provide enough support for children in building models. Since we are concerned to enable children to explore their own models, rather than only explore models preconstructed by others, we think that Valley's question-answering techniques should be complemented by a knowledge acquisition strategy, as in PCT.

Finally, we note a considerable difference in methodology. Our project's use of the Persistent Collaboration Methodology (see section 3.6) ensured a strong classroom focus at most stages of the research. Thus in the case of PCT, the tool's usability has been assessed empirically by studying children's performance with the tool. By contrast, EXPLORES was not used with children and there is no quantitative or qualitative data on its usability based on actual trials. This makes design decisions and claims about the shell's benefits difficult to assess.[4]

9.3.2 Webb

Webb (1994) describes a tool called Expert Builder, the design of which was partly motivated by an aim similar to our own: to improve upon rule-based expert system shells as classroom modelling tools. Her tool, which is implemented in Microsoft Windows, provides a form of representation somewhat similar to the AORTA trees used in the Transparent Prolog Machine (Eisenstadt & Brayshaw 1988).[5] As an example of a model in Expert Builder, Webb provides the diagram which we reproduce in Figure 9.2. Such a model is constructed with a set of direct manipulation tools. In its use of a graphical editor to create a tree structure, Expert Builder superficially resembles our own PDT.

---

[4] Of course, the present research has benefited greatly from the advances that have been made since the time of the EXPLORES project, including the much greater availability of reasonably powerful computers in schools.

[5] But unlike the TPM, Expert Builder does not support variables. Also, we ignore the fact that as Figure 9.2 shows, Expert Builder models may not be trees but tangled hierarchies.

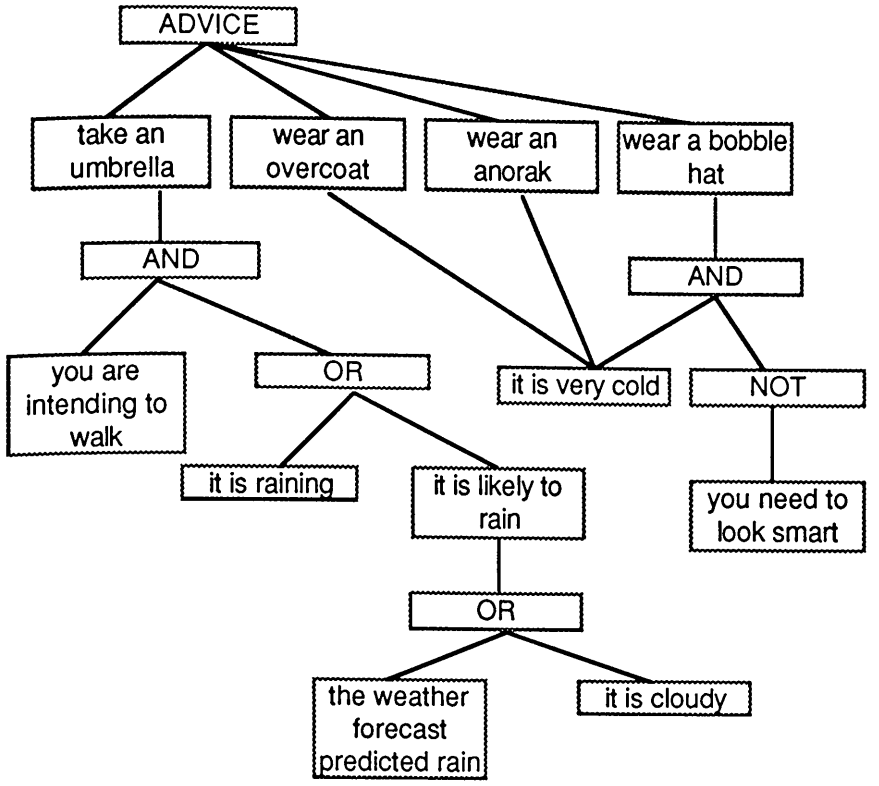


Figure 9.2 A model diagram in Webb's Expert Builder shell

When we demonstrated this example to a group of teachers who were participants on an M.Ed. course in Knowledge-based modelling, they found the diagram difficult to understand. Only when they realised that each subtree is a graphical encoding of a set of IF/THEN rules, with implicit IF operators, and with other logical operators following a prefix convention, did its meaning become apparent. For example, the leftmost subtree of the ADVICE node is equivalent to the following pair of PKRL rules:[6]

```
ADVICE take an umbrella IF
   you are intending to walk AND
   {it is raining
        OR
   it is likely to rain}.

it is likely to rain IF
   the weather forecast predicted rain
        OR
   it is cloudy.
```

[6]Note that the second rule is called by the first. In the PKRL, rules that are not top-level omit the keyword ADVICE.

The teachers challenged the assumption that Expert Builder's diagram provided a helpful representation. They argued that to comprehend the diagram requires first constructing mentally the equivalent rules and so it would be quicker to begin with the rules in textual form. However, this argument might be countered in at least two ways. First, inspection of the diagram enables some inferences to be made rather rapidly (for example, that 'it is cloudy' is a proposition that may somehow contribute to the evaluation of 'take an umbrella'). Second, even if it is true that the diagram is harder to interpret than the rules, it may be more convenient to *construct* the model diagrammatically.

In the present research we have argued that for usability, tools should offer children forms of representation with which (ideally) they are already familiar, or which are readily accessible to them otherwise; and that tools should interpret models in a transparent way. The M.Ed. teachers greatly doubted that Expert Builder's AORTA-like trees would be familiar to children and they questioned their accessibility.

The teachers also disputed that the program's method of interpreting models is transparent. At runtime, Expert Builder queries the user to supply true/false answers to the propositions specified by the leaf nodes of the tree, following a (basically) depth-first order of traversal. Although the tool helpfully animates the traversal by colouring nodes red or blue dynamically as they become evaluated to true or false, the teachers sometimes had difficulty in understanding why nodes became coloured and in predicting what question the system would ask next.

Such questions should really be tested empirically, as we have done for PDT. It will be recalled that although PDT is also based on a tree representation, its trees are decision trees with a syntax and semantics that differ greatly from Expert Builder's form of representation. We have strong evidence that decision trees are familiar to or accessible by children. Also, PDT's method of model interpretation is much simpler than Expert Builder's. In PDT, running a model involves the traversal of exactly one branch of the tree from root to leaf. Although there is no animation of the diagram, the system's behaviour is invariably easy to predict.

A further principle which we advocated is that tools should

reflect conceptual frameworks that are helpful for building models. In the case of PDT, the use of decision trees provides a framework of relationships between questions, answers and decisions that is widely found to be useful for describing classification procedures. Expert Builder provides only the weak (i.e. general) framework of (a subset of) propositional logic. It could be argued that this provides flexibility: according to Webb (1992), the tool has been been used to construct models of various types, including diagnosis and planning as well as classification. We make no such claim for PDT, which is essentially restricted to classification models. We have argued in this thesis that such a restriction is acceptable because it enables knowledge acquisition techniques to be used which empower children as model builders. We have willingly traded general-purposeness for usable and useful expressive power.

Expert Builder differs from PDT in many other, more minor, ways. We have already contrasted (in section 5.1.1.1) the two tools' very different methods for constructing trees graphically. Also, in Expert Builder a model must appear in a single window whereas in PDT a model can be composed of multiple linked trees distributed over several windows. Expert Builder, like EXPLORES, lacks model analysis tools that might be compared to the PMA and PMC. However, some of the interface features of Expert Builder are superior to PDT's. For example, the program has a 'long-range view' facility that is more flexible than PDT's viewer.

A significant difference again relates to methodology. Consistently with the methodology used in our research, it would be desirable to implement Expert Builder's AORTA-like trees as another Primex extension. The success of model building using this representation could then be compared to that with PDT, the PKRL, and so on, using our standard domains and measures. Although she did test Expert Builder with children, it appears (from Webb 1992, Webb 1994) that Webb used the system in isolation from other tools and her domains are not clearly specified. This makes it hard to evaluate some of the claims that are made for the system.

9.4 Contribution of this research

Compared to previous research, we summarise the main original contributions of this research as follows:

1. The research overall has investigated the contribution that can be made by knowledge acquisition technology to the design of educational tools for knowledge-based modelling. Knowledge acquisition research in KBS aims to enable domain experts to communicate their knowledge directly to a computer, bypassing the need for intervention by a knowledge engineer. We have demonstrated that it provides a valuable source of techniques for knowledge-based modelling in education.

2. We developed and evaluated PDT, PFT, and PCT, three specific designs for modelling tools which use knowledge acquisition technology. These tools are quite close to a state in which release versions could be produced and made available to schools.[7] (Chapters 4, 5, 7 & 8).

3. We described a set of measures, and questions relevant to the measures, for evaluating the extent of success of children's modelling. We showed that these measures can be implemented in practice. (Section 3.3, Chapters 7 & 8).

4. We gave formal definitions for a set of measures (correctness, efficiency and conciseness) for analysing the quality of children's models. We showed that these measures can be implemented in practice. (Sections 6.1 & 7.2).

5. We compared the extent of success of children's modelling using the new knowledge acquisition tools with that using a rule-based shell. We demonstrated that in important respects, the new tools are superior. (Chapters 7 & 8, section 9.2).

---

[7] As shown by the fact that a large Education Authority has recently paid £5000 to Moray House Institute of Education for a trial disk containing the *existing* versions of PDT, PFT and PCT to be distributed to all secondary schools within its area.

6. We explicated a research methodology for projects in applied AIED and illustrated the methodology by reference to our own project. (Section 3.6).

7. We investigated the idea that models built with the tools could be analysed semi-automatically and demonstrated that the idea is feasible. (Chapter 6).

8. We investigated two specific applications for automated or semi-automated analysis of models and developed tools for these applications. The PMA evaluates a model's quality and is useful in its current form. The PMC compares two children's models and is a prototype demonstration that suggests a new line of research in collaborative learning. (Sections 6.2 & 6.3).

9.5 Future research

We now summarise some possible directions for future research, for convenience identifying where in the thesis they have been discussed previously. We omit the proposals for tool development that have already been summarised in Table 9.1 above, except for those that are clearly of a research nature.

• Student modelling systems (sections 8.3.2 & 8.4.2): It seems likely that student modelling systems could provide 'discussion layer' strategic help, along the lines of the proposals by Cumming & Self (1990). For example, in PCT such a system might offer some of the suggestions that at present can be obtained using the 'Check' tool. Knowledge of the domain is not necessary to provide this kind of help.

• Automatic co-construction (section 8.5.2.2): The feasibility and benefits of a facility for constructing automatically or semi-automatically a different kind of model, based on information contained in the current model, should be investigated.

• Improved laddering dialogue (section 8.4.2): Although fairly successful, PCT's dialogue strategy should be improved to elicit a higher quality of feature information. At present, children usually specify features involving only boolean attributes and sometimes these seem to be incidental rather than prototypical features of the class concerned.

• Other contrived techniques (section 4.3.1): Laddering is not the only contrived technique for eliciting information in classification domains. Others, such as card sort and repertory grid techniques, may be equally or more successful and should be investigated.

• Other representations (section 4.1): As the example of Expert Builder's AORTA-like trees shows, alternative forms of representation exist that might be successful for building classification models. These could be implemented as Primex extensions and evaluated with the domains and measures used in this thesis for the Primex tools.

• Other tasks and domains (section 3.7): Classification is certainly not the only task that is of interest to schools. Other tasks such as design and planning should be investigated and knowledge-based modelling tools developed, based on suitable representations and techniques. Although we have managed to avoid making domain-based assumptions about the information that will be modelled, such assumptions may be justified if they enable methods to be used which empower children to build better models.

• Learning outcomes (sections 3.5 & 7.6): Research is needed into the learning that results from modelling with various tools, including learning of domain concepts, representational skills, collaborative skills and metacognitive skills. There is scope too for cognitive science research into the cognitive processes that enable (and are enabled by) model building.

• Automated model analysis (sections 6.2 & 6.3): Better tools should be developed than our PMA (which is not fully automatic) and our PMC (which is fully automatic but rather inflexible). Using

techniques in Natural Language Processing, it should be possible to improve upon the methods which these tool currently employ.

9.6 Prospects for knowledge-based modelling

In Chapter 2 we traced the development of knowledge-based modelling in education from the work that centred on the Prolog language in the early 1980's through to the deployment by schools of expert system shells in the last few years. We concluded that although the shells have secured a place in secondary schools, it is a very limited one that is mainly confined to the classrooms of specialist IT teachers. We constructed a table, reproduced below as Table 9.3, in which the gap between the current state of practice and our envisaged 'ideal state' is characterised by a much wider range of teacher-users and by an emphasis on learning outcomes which relate to the wider curriculum instead of being mainly concerned with skills in IT. It seems appropriate to conclude this thesis by discussing the prospects that exist for narrowing the gap.

	Current state of practice	*Envisaged 'ideal' state*
Users	IT specialist teachers	Subject teachers from many curriculum areas and stages
Intended learning outcomes	IT skills	Domain knowledge Skills in modelling Transferable skills
Models	Informal topics	Curriculum topics

Table 9.3 The gap between existing practice and an 'ideal' state

Our research has demonstrated (albeit with qualifications) that tools can be made available to teachers which enable more successful knowledge-based model building by children than has been possible with the tools that were available hitherto. Although this must improve the prospects for knowledge-based modelling, it would be naïve to expect rapid change on this account alone. As discussed in Chapter 2, change in education is complex and typically depends on a combination of new materials, new teaching approaches and new beliefs (Fullan 1991). New computer software, however good, may not stimulate significant change

unless other conditions are also favourable.

The charge of naïvety — of ignoring the actual approaches and beliefs of teachers — could reasonably have been levelled at the present researcher in his work a decade ago, when he visited a school weekly to teach a select group of children a course in micro-Prolog. That course, which was subsequently published in book form (Conlon 1985), was not much adopted by schools. Although the development of Prolog represented a milestone in computer science, the language was external to and never took root in the culture of the classroom.

In contrast to that work, a strength of the present research should be our PCM methodology which has provided a strong focus on authentic practice. Many of the findings which we report in this thesis are supported by trials conducted in schools by regular teachers working under normal classroom conditions. Our modelling tools PDT, PFT and PCT have been shaped by a vast quantity of interactions with teachers, children and students, not imported from a laboratory as was Prolog. It may not be too much to hope that the tools and findings of the present research are realistically grounded in the teaching approaches and beliefs of teachers, and hence might stimulate change.

Yet, although laboratories are not sufficient, they are necessary. It could be said that by embracing knowledge acquisition technology, schools would be catching up with KBS developments in the world outside. The EMYCIN-type shells which schools presently use are now very dated and their limitations have long been recognised outside of education. As discussed in Chapter 2, research in knowledge acquisition carries the main hope that domain experts can communicate their knowledge to computer systems without the intervention of knowledge engineers. It seems a natural step for educational developers to draw upon this research in making advances in knowledge-based modelling. What seems to be necessary is the best kind of collaboration between researchers, teachers, and technologists, to shape and reconstruct developments that can meet the authentic needs of teachers and children.

This thesis has already identified some of the barriers to a wider uptake. Staff development is necessary to help teachers to

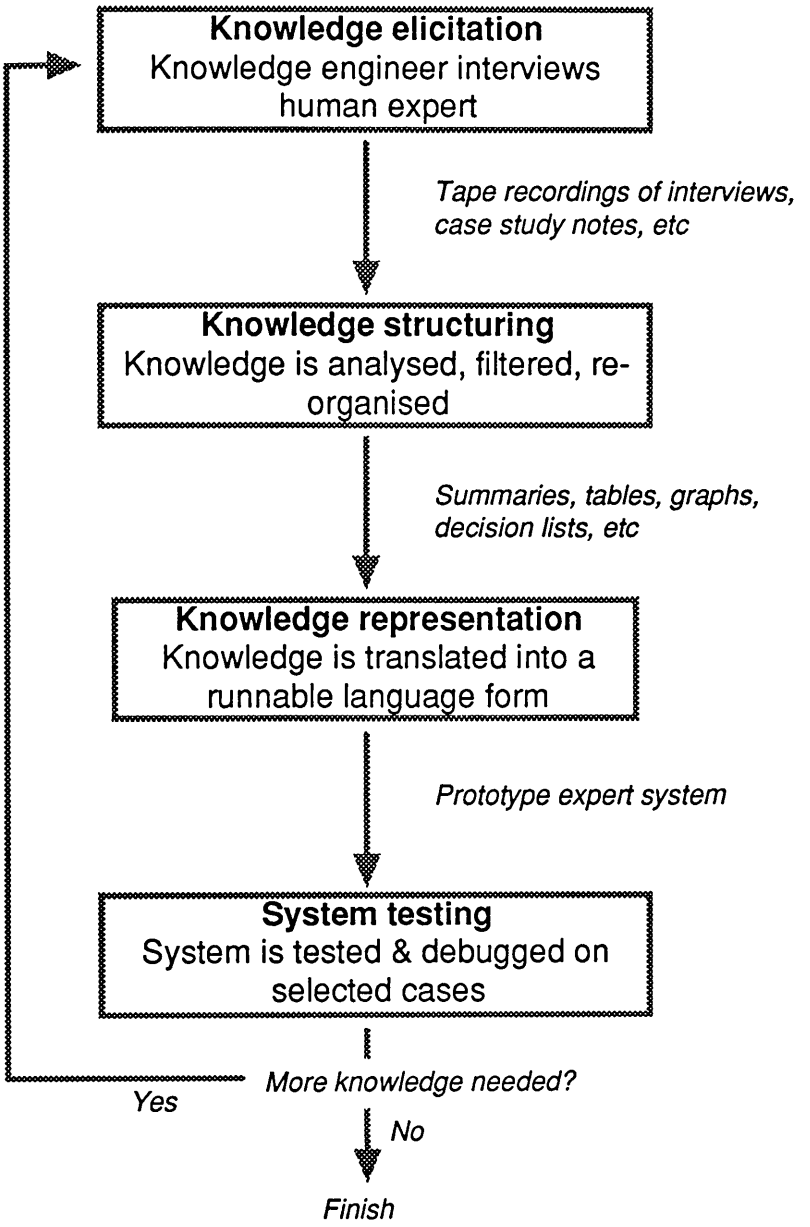
learn about the tools and the ideas of representation and inference upon which the tools are based. To support this, research is needed into forms of classroom practice that promote the most effective teaching and learning. This in turn requires that the relationship between modelling and learning be investigated in depth. It will be important also to ensure that curriculum policy makers become aware of the potential of the new technology.

In general, computer modelling seems to be at least partly in tune with some influential trends and developments in education. It can potentially offer valuable support to project work, thematic work, collaborative work, and the development of transferable skills, all of which have been widely advocated in education circles. Schools have already shown that they value flexible and expressive IT tools such as word processors, spreadsheets and graphics packages. Modelling tools are similarly flexible and expressive. Although schools (especially Primary schools) have often lacked sufficiently powerful hardware to run modelling software, that situation is changing as falling prices allow modern computers to be purchased in quantity.

The special contribution of knowledge-based modelling — as opposed to say, modelling with a multimedia package — stems from its use of explicit, formal structures of knowledge representation and inference. Not only can children benefit from working with and learning about these formal structures, which are various, but also their formal properties have many advantages: for example, models can be analysed for anomalies, compared to other models, validated against reference models, translated into other forms, and so on.

In conclusion, it seems clear that although problems remain, knowledge-based modelling has enormous potential to provide usable and useful classroom tools. This thesis has been a modest attempt to show that even in the shorter term, it should be possible to make quite significant advances.

Appendix 1: Evolutionary prototyping methodology



Appendix 2: Questionnaire on knowledge-based tasks

Dear Colleague,

As part of a research project on problem-solving I would welcome your help. My aim is to discover the extent to which various kinds of problem-solving task feature in classrooms at present. I also need to find out which tasks are particularly valued by teachers. I realise that you are very busy but if you could devote ten minutes of your time to completing the questionnaire below, I would be very grateful.

In completing the questionnaire, please try to relate to your own teaching experience as far as possible. For the middle column, use these codes:

1	2	3	4	5
never	seldom	occasionally	quite often	very often

Overleaf is an informal description of each task type. You may like to consult this whilst completing the table.

If you would like to attach a comment, please do so.

Thank you very much!

Tom Conlon
Moray House Institute of Education

Type of task	How often do your pupils encounter this type of task? Circle 1=never to 5=very often.	Would you like pupils to encounter this task more frequently? Circle yes or no.
Diagnosis	1 2 3 4 5	yes no
Verification	1 2 3 4 5	yes no
Correlation	1 2 3 4 5	yes no
Monitoring	1 2 3 4 5	yes no
Classification	1 2 3 4 5	yes no
Prediction	1 2 3 4 5	yes no
Repair	1 2 3 4 5	yes no
Remedy	1 2 3 4 5	yes no
Control	1 2 3 4 5	yes no
Maintenance	1 2 3 4 5	yes no
Design	1 2 3 4 5	yes no
Configuration	1 2 3 4 5	yes no
Planning	1 2 3 4 5	yes no
Scheduling	1 2 3 4 5	yes no
Modelling	1 2 3 4 5	yes no

Descriptions and Examples of task types

Note: most of the tasks are in fact *highly* general. The descriptions are informal and are intended to be indicative rather than strictly precise. The examples given are not necessarily representative of all the important forms of each task type. I don't want to 'cue' your response, and for that reason no attempt has been made to select examples which reflect specific areas of the curriculum.

Task	*Description*	*Example*
Diagnosis	Finding the cause of unexpected symptoms in a person or machine	Locating the cause of an illness, finding a fault in an electrical circuit
Verification	Determining whether a statement about an object or system is correct	Testing a claim about a car, eg 'the petrol tank is empty'
Correlation	Identifying similarities or differences between two or more objects or systems	Comparing two animals to see if they are of the same species
Monitoring	Using measurements of an ongoing system to check for correct functioning	Watching pots and pans on a stove to ensure correct cooking
Classification	Categorizing an object or system into one of a fixed set of classes	Categorizing shapes, leaves, cars, animals
Prediction	Determining what will happen next to an object or system	Determining the outcome of a collision or the future value of an investment
Repair	Replacement of a defective element in an object or system	Mending a bicycle puncture with a new inner tube
Remedy	Counteracting a malfunction in an object or system by initiating a new process	Counteracting a bicycle puncture by inflating the existing inner tube
Control	Modifying a system's behaviour with the aid of feedback	Setting the brightness level of a TV set
Maintenance	Ensuring that an ongoing object or system behaves correctly	Servicing a car
Design	Specifying the components and organisation of a new object or system	Designing a house, bicycle, computer program
Configuration	Assembling the elements of a system correctly	Connecting the parts of a new hi-fi system
Planning	Determining the actions required to meet a goal	Planning a shopping trip, planning a career, making a financial plan
Scheduling	Determining how actions should be sequenced to satisfy a set of constraints	Determining the arrival/departure times of an international flight to maximise profits
Modelling	Building a scaled-down representation of a real-world object, system or process	Building a model aircraft, a financial model, a population model

Appendix 3: Upgrade specification for Primex 2

LANGUAGE ENHANCEMENT

- brackets {} available for rule bodies
- variable convention changed to underscore
- new EXPLANATION keyword
- Startup graphics
- 'don't care' variables
- more powerful menu question processing
- new maths functions
- advice rules now callable

GENERAL HCI

- Automatic compilation
- explanation dialogues improved
- 'standard' and 'novice' configurations
- 'recordable' consultation
- selective inferencing tracing
- uncertainty handling improved
- Hypertext and balloon help

CONSULTATION

- 'What if?' command for rapid testing
- 'Recap' command summarises conclusions
- Database-style query interpreter

DOCUMENTATION

- Technical Guide documents new features
- Learner's Guide with graduated examples

INTEGRATION

- Import database files, e.g. from Claris Works
- Records automatically become KRL facts

PERFORMANCE

- Compilation much faster
- 32K limit on source KB files removed

IMPLEMENTATION

- Mac System 7 (24 bit)
- Object-oriented design
- Prolog++ implementation
- Extensible architecture

Appendix 4: Developing Primex extensions

Developers may wish to build their own Primex extensions. This appendix describes the requirements.

Development environment

Extensions should be developed in LPA MacProlog32 and Prolog++. It will be helpful to have the codefile `generic.XTN` installed since this makes available many methods that are likely to be useful to extensions (see below). The base Primex system should not be needed until the extension has been fully developed. Version numbers and/or version dates at the time of writing (September 1996) are:

LPA MacProlog32	1.09d/February 1995
Prolog++	1.0/January 1995
base Primex system	2.5/February 1995
generic.XTN	May 1996
Macintosh OS	B1-7.5.1

Extension codefiles

The extension should be implemented as a set of Prolog++ objects stored within a single codefile. Objects should be named differently from the name of any Primex object (see below). The codefile, which must have a name of the form `prefix.XTN`, should be located in the 'Extensions folder' which itself resides in the same folder as the Primex application file. It will be loaded automatically when the base Primex system starts running.

The interface object

If the codefile is named (say) `test.XTN` then one of the objects must be named `test`. This object provides the interface to Primex. It should have an `install/2` method which will be called automatically by Primex when `test.XTN` is loaded. The first argument of `install/2` should specify a list of atoms representing command names which will be added automatically to the `File` menu's `Extensions` submenu. For each of these names, the `test` object should have a zero-arity method of the same name: this method will be invoked automatically when the user selects the menu command. The second argument should specify a list of four-character atoms identifing the extension's filetype names. These filetypes will be automatically visible in future `File Open` dialogues. As an example, the decision trees extension codefile

`trees.XTN` defines the following object named `trees` as its interface object.

```
open_object trees.

install([New], [FileType]) :-
   New = 'New decision tree',
    tree_windows <- dtree_filetype(FileType),
    tree_initialiser <- setup,
    init_gensym('Decision tree ', 1).

'New decision tree' :-
    cursor(watch),
    gensym('Decision tree ', WName),
    tree_windows <- dialogue_new(WName).

close_object trees.
```

Printing an extension's windows

The Print dialogue automatically offers all graphics windows (except `usr`), so no special programming is needed for printing.

Opening and saving extension's files

When the user selects the Primex `File Open` command, the file dialogue will offer in addition to Primex's native types the file types which the extension declared via its interface object's `install/2` method. If the user selects a file name `F` of a type `T` then Primex broadcasts to all installed objects a message

```
O <- xtn_open(T, F).
```

The extension should have an object `O` (not necessarily the interface object) with an `xtn_open/2` method. The method should fail if `T` is not one of the extension's own file types. Otherwise it should arrange to open `F` and succeed.

When the user selects the Primex `File Save` command and a graphics window `GW` is at the front then Primex broadcasts to all installed objects a message

```
S <- xtn_save(GW).
```

The extension should have an object `O` with an `xtn_save/1` method. The method should fail if `GW` is not one of the extension's own windows. Otherwise it should arrange to save `GW` (and any other currently open windows owned by the extension) and succeed.

Responding to `Close` and `Restart`

An extension may have an object `O` with a method `xtn_close/0`. When the user selects `Close` or `Restart`, Primex broadcasts an `xtn_close` message to all objects. The behaviour of

`xtn_close` should be to close all the extension's windows (not Primex KRL windows) and succeed.

Object names used by Primex

Primex uses these object names which should therefore be avoided elsewhere:

answer_changer, answer_finder, balloon_help_resetter, browser_help, codestore, code_manager, compiler, compiler_utilities, consulter, dialogue_utilities, edit_menu, errors, event_handler, expert_menu, filetypes, file_menu, how_explanations, inferencer, inference_utilities, initialise_primex, input_parser, interaction_log, menumaster, primitives, printer, qn_dlogs, query_interface, recommendations, run_terminator, settings, startup_responder, term_converter, tracer, why_explanations, windows_hci, windows_menu, window_manager

Object names used by extensions

A convention used for extensions is that all extension objects have names based on the extension prefix name. For example, the decision trees extension which is implemented as a codefile `trees.XTN` has objects named `trees`, `tree_click_handler`, `tree_dlogs`, `tree_drag_tool_handler`, `tree_drawing_primitives`, etc. This convention should prevent any name clash between co-resident extensions.

Using `generic.XTN`

An extension may use any of the public methods defined by this special 'generic' extension. The objects defined by `generic.XTN` are:

generic	— interface object for generic.XTN
generic_command_manager	— creates and manages graphic window menu boxes
generic_dialogues	— defines dialogue-generating methods
generic_graphics	— computes graphic node descriptions
generic_krl_compiler	— compiles normalised models into Primex KRL
generic_model_manager	— manages database of normalised models
generic_settings	— reads and writes the 'Extensions settings' file
generic_zoom_tool_handler	— implements a zoom tool

Methods are summarised as follows. For details consult the source listing of `generic.XTN` which is heavily commented.

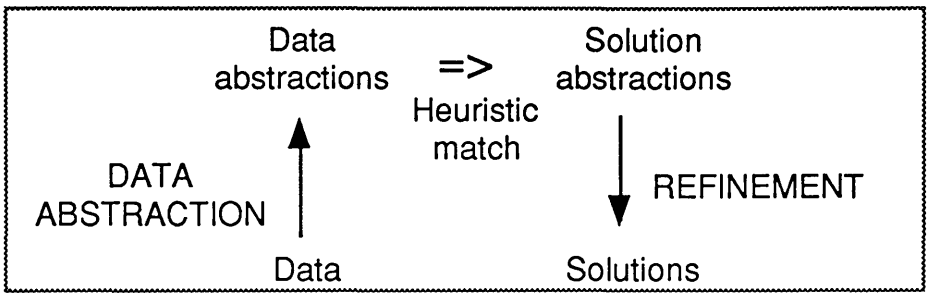
generic_ model_ manager	get_model(MName?, Model^) *Get a normalised model using its name* install_model(ModelFileName^, Result^) *Retrieve & store a normalised model from disk* clear_model(ModelName?) *Delete a named local (not disk) model*

generic_ command_ manager	add_hmenu(MItems?, Win?, MY?, MX?) *Horiz menu, MItems a list of item(Label, Handler)* add_vmenu(MItems?, Win?, MY?, MX?) *Vertical menu, MItems a list of item(Label, Handler)* pt_in_menuitem(Win?, Pt?, MenuItem^) *Return MenuItem containing Pt, update dribble file.* run_command(Win?, Mod?, MenuItem?) *Sends message: Handler <- run(Mod?, Win?)* help_command(W?, MenuItem?) *Sends message: Handler <- help(W?, H^) then displays H*
generic_ dialogues	menu_popup(GW?, Header?, PItems?, PreSel?, Selected^) *Make a popup menu in the top left corner of W* simple_edit(W?, Width?, Header?, Prefill?, Entered^) *Create a simple edit field dialogue in the topleft corner of W.* edit_with_btn2(W?, Prompt?, Prefill?, Label?, Typed^, Clicked^) *Create a dialogue with an edit field* display_scroll_text(Title?, Txt?) *Display Txt in a scroll field with Title at top* message(W?, Msg?) *Display a message Msg (an atom)* choose(W?, Msg?, Btn1Label?, Btn2Label?, SelectedBtn^) *Display a message Msg (an atom) in W with two buttons*
generic_ graphics	node_gdl((Y,X)?, Txt?, Font?, Colour?, Size?, Border?, Box^, GDL^) *Computes the GDL for a node with top left corner at (Y,X) containing centred Txt. No side effects.*
generic_ krl_compiler	cache_compile_model(W?, CompilerObject?, EditTime?, Model^) *Get a normalised model (a kb/3 term) for W, from this object's cache if a current one exists, otherwise by invoking the CompilerObject:do_compile/2 method* run_model(W?, Model?, LastEditTime?) *Generate KRL for Model, write it to a Primex KRL window, run it* primex_available *Succeeds if the call is made in the base Primex environment.* filter_sym(Sym?, Sym2^) *Sym2 is Sym, or is a version of Sym that avoids any clash with Primex reserved words. Assumes primex available.*
generic_ settings	read_settings_file *Reads the Extensions Settings file into this object's local database.* write_settings_file *Writes this object's entire local database back to the file.* write(Object?, Attribute?, Value?) *Writes an entry into the local database.* read(Object?^, Attribute?^, Value?^) *Reads an entry from the local database.*
generic_ zoom_tool_ handler	activate(W?) *Set the graphics cursor to show the activation of the tool* clicked_in_space(Modifier?, W?) *Respond to a click with the tool in a free area of graphics window W* help(ToolName?, Help^) *If ToolName is 'The Zoom Tool', return words/1 help on the tool*

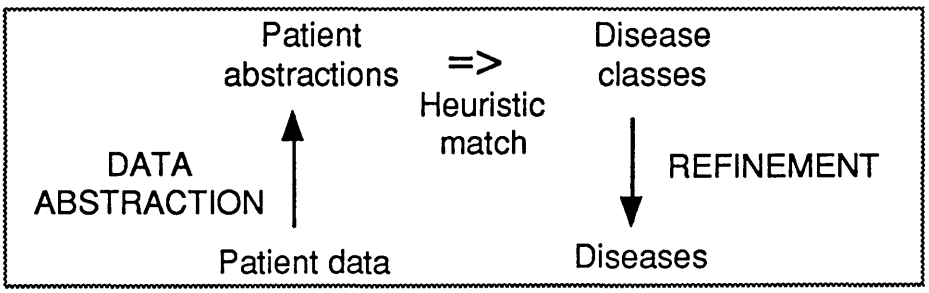
Appendix 5: Heuristic classification

The arrows in these diagrams show the flow of inference, but not necessarily the inference strategy, i.e. the order in which inference steps are made. Under a forward reasoning strategy, steps do follow the direction of the arrows. Backward reasoning proceeds in the reverse direction. Middle-out or bidirectional reasoning is also possible.

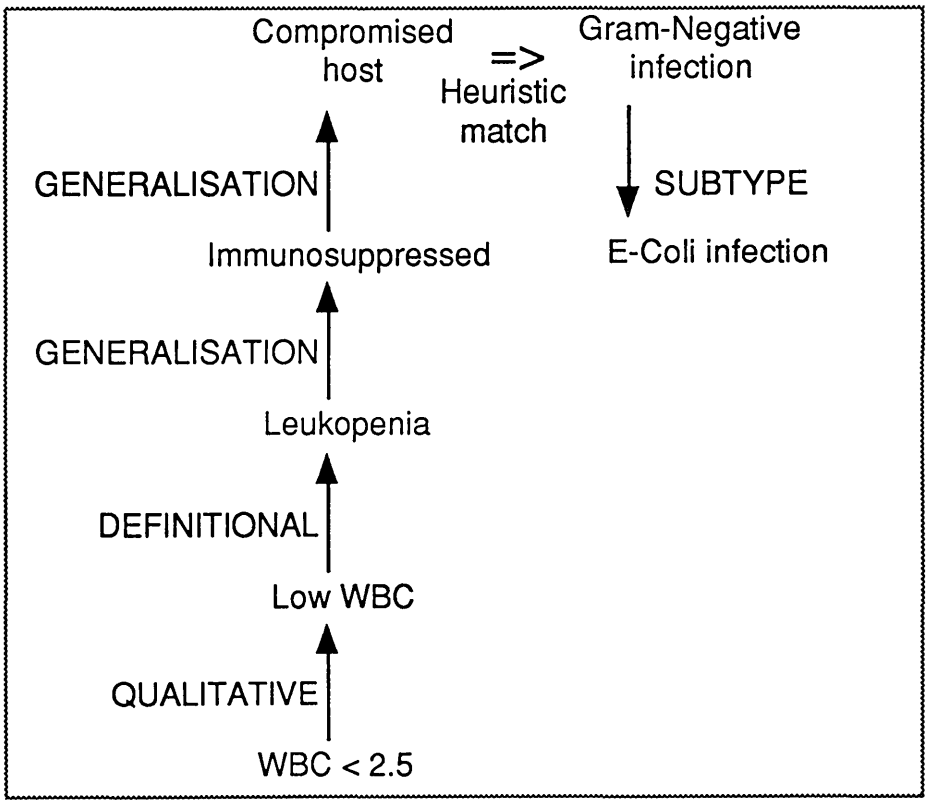
General inference structure



General inference structure applied to MYCIN



Example of MYCIN inferencing in one case



Appendix 6: Technical Notes

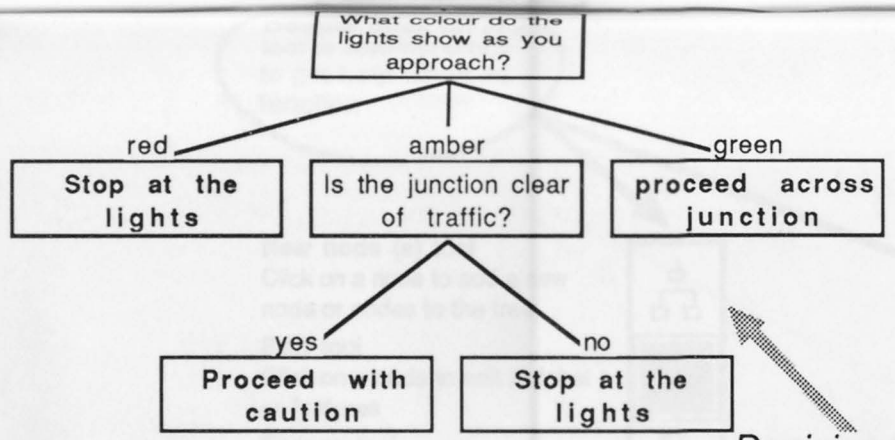
(1) The root of the problem was that LPA MacProlog at that time was not '32-bit clean'. The Macintosh OS up to and including OS 7.0 had supported two memory management systems: one for 24-bit programs and one for 32-bit programs. The former tended to be older programs like MacProlog. OS 7.5 removed the 24-bit option as part of Apple's general strategy of upgrading to exploit the characteristics of the PowerPC microprocessor, which had been introduced to replace the Motorola 68XXX family of processors used in all Macintosh computers up to that time. The result was that neither MacProlog nor Primex 2.0 would run under OS 7.5. LPA produced a 32-bit clean version, MacProlog32 v1.0, in 1994. However the development of MacProlog32 was also used by the company as an opportunity to bring the system more into line with the language specification of its Prolog for Windows system.

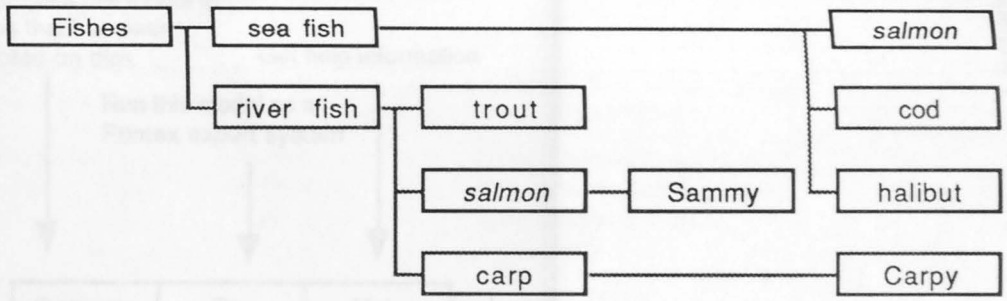
(2) When operating with extensions, the Primex Save command saves to disk all currently active windows that share the model type of the window that is at the front at the time of the Save. If the front window is not of type PDT, PFT or PCT then the system defaults to a PKRL-type save, even if the front window is a dialogue window that sits immediately above a PDT, PFT or PCT window. Unfortunately, this convention caused quite a few models to be incorrectly saved and lost. The design could be improved by offering users a choice of save actions whenever more than one kind of model window exists concurrently.

Appendix 7: User help sheets

This appendix contains help sheets intended for users of the Primex modelling tools. The sheets summarise information about the graphical commands, syntax and semantics of model diagrams, compositionality, and methods of running models. They were issued to teachers in the large-scale trials and were used frequently at other times in the research. The sheets are summarised in the table below.

Title of Sheet	*Contents of Sheet*
Knowledge-based modelling in Primex 1996	Shows all four model types (PKRL, PDT, PFT, PCT) and states how a model of each type can be created in Primex.
The Primex Decision Tree tool	Summarises PDT
The Primex Factor Table tool	Summarises PFT
The Primex Classification Tree tool	Summarises PCT
The Primex Rule Language	Summarises the PKRL





Decision tree models

File menu — Extensions submenu
— 'New decison tree' command

Classification tree models

File menu — Extensions submenu
— 'New classification tree' command

Knowledge-based Modelling in Primex

with 1996 Extensions

Factor table models

File menu — Extensions submenu
— 'New factor table' command

Water type	Light	Fish rising	Fly
river	dull	no	14 Black spider
river	bright	no	14 Butcher
river	*	yes	14 Dry Tups
loch	dull	*	12 B&P
loch	bright	*	12 Alexandra

Rule models

File menu — 'New' command

```
ADVISE You should wear trainers IF
       You intend to go outside AND
       The weather is good.

ADVISE You should wear stout shoes IF
       You intend to go outside AND
       There are signs of rain ahead.

ADVISE Put your wellies on IF
       You intend to go outside AND
       The streets are very wet.

ADVISE You should wear slippers IF
       NOT You intend to go outside AND
       You like to be comfortable indoors.

ADVISE Remember to clean your shoes regularly IF
       You want your shoes to stay smart.
```

Double click on any tool or command box to get **help** about its function

New node (s) tool
Click on a node to add a new node or nodes to the tree

Edit tool
Click on a node to adit its label or features

Eraser tool
Click on a node to delete it

Tree tidying tool
Click on the root to tidy the tree

Drag tool
Point at a node, press, and drag to move the node

Zoom tool
Click on the tree to enlarge or reduce the image

Compare this model to one that has been stored on disk

Get help information

Run this model as a Primex expert system

Compare | Run | Help

The root of the tree shows the first question.

What colour do the lights show as you approach?

Each arc is labelled with one way to answer a question. Answers are shown in blue type

red — **Stop at the lights**

amber — Is the junction clear of traffic?

green — **proceed across junction**

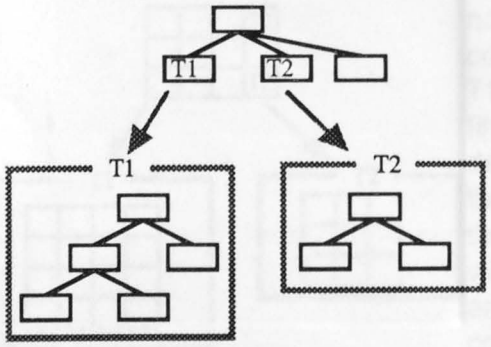
yes — **Proceed with caution**

no — **Stop at the lights**

Non-leaf nodes are questions. Text appears in normal type.

Leaf nodes are decisions. Text appears in **bold** type.

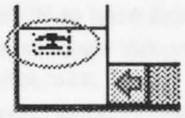
A big decision tree model can be composed of several **linked trees**, as shown here.



The top tree has 'subtree' type nodes labelled with names T1 and T2. These names identify the subtree windows.

Scroll the window with the scroll bars or the viewer

Dragging the grey rectangle in the viewer gives a **fast** scroll



Double click on any tool or command box to get **help** about its function

Run this model as a Primex expert system. Usually, the **fewest possible** number of questions is asked to identify the value of the outcome factor.

Fill the table with data read from a spreadsheet (ASCII Text) file

Compare this model to one that has been stored on disk

Get help information

Save the table to a file that can be opened by a spreadsheet or word processor

Compare | Run | Help | Import | Export

Edit tool
Click on a cell to edit its contents

Eraser tool
Click on a cell to delete its contents

Select tool
Click on a cell to set height and width or to delete or insert rows or columns

Names of **decision factors**. These are the items to consider when making a decision

The **outcome factor**. This column gives the decision for various values of the decision factors.

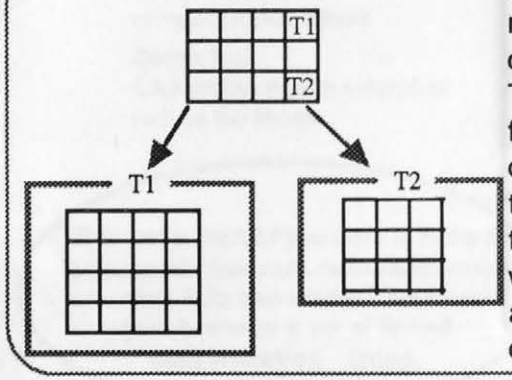
Water type	Light	Fish rising	Fly
river	dull	no	14 Black spider
river	bright	no	14 Butcher
river	*	yes	14 Dry Tups
loch	dull	*	12 B&P
loch	bright	*	12 Alexandra

An asterisk is the **wildcard** or 'don't care' value. When fishing a loch in dull light, use a 12 B&P regardless of whether fish are rising.

This row says: when fishing a river in bright light with no fish rising, use the fly called a 14 Butcher.

- Be sure to have rows that cover all possible combinations.
- It's OK to have two rows with the same factor values but different outcomes.
- Don't be inconsistent in your use of upper and lower -case letters.

A decision table model can be composed of several **linked tables**, as shown here.



The top table has 'table-name' type cells labelled T1 and T2. To find the decisions for these cases, the tables in windows T1 and T2 are consulted.

The Primex Classification Tree Tool

With the interview tool, the system will ask you questions. It will use your answers to build the model. A model built this way can still be extended or edited with the other tools.

Double click on any tool icon or command box to get **help** about its function

Check this model for missing information

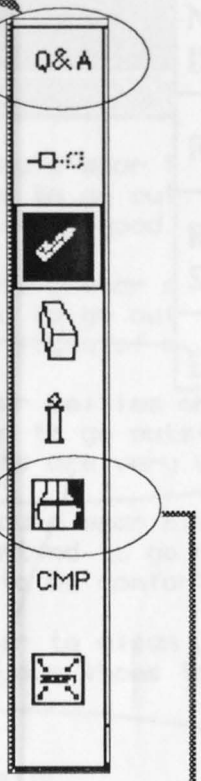
Get help information

Compare this model to one that has been stored on disk

Run this model as a Primex expert system

Set options that affect the behaviour of the system

Compare | Check | Run | Help | Options



Interview tool
Click on a node to begin or resume interview

New node tool
Click on a node to add a new subclass

Edit tool
Click on a node to adit its label or features

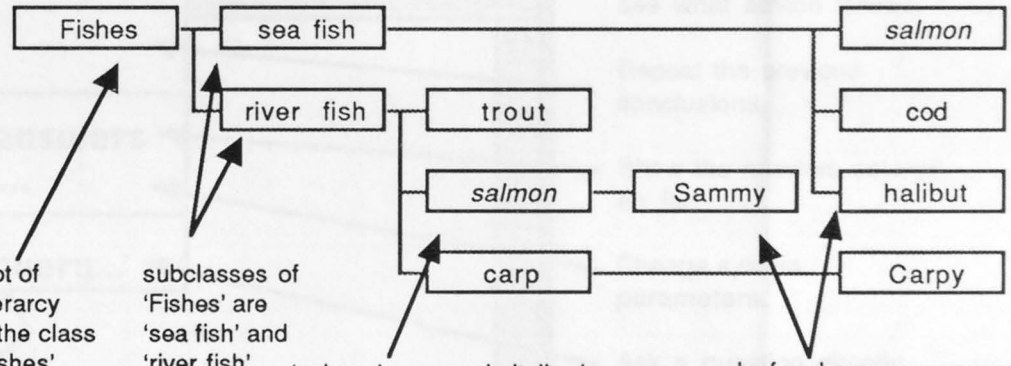
Eraser tool
Click on a node to delete it

Information tool
Click on a node to see information about it

New window tool
Click on a node to create a new window with node as root

Comparison tool
Click on a node to compare it with others

Zoom tool
Click on the tree to enlarge or reduce the image



root of hierarcy is the class 'Fishes'

subclasses of 'Fishes' are 'sea fish' and 'river fish'.

'salmon' appears in italics because it occurs twice on the tree.

leaf nodes may represent individuals ('Sammy') or subclasses that have not been further analysed ('halibut')

This tool is useful if you want to make a large model. You could define each major sub- class in its own window. Your model then becomes a set of **linked classification trees**.

Inheritance

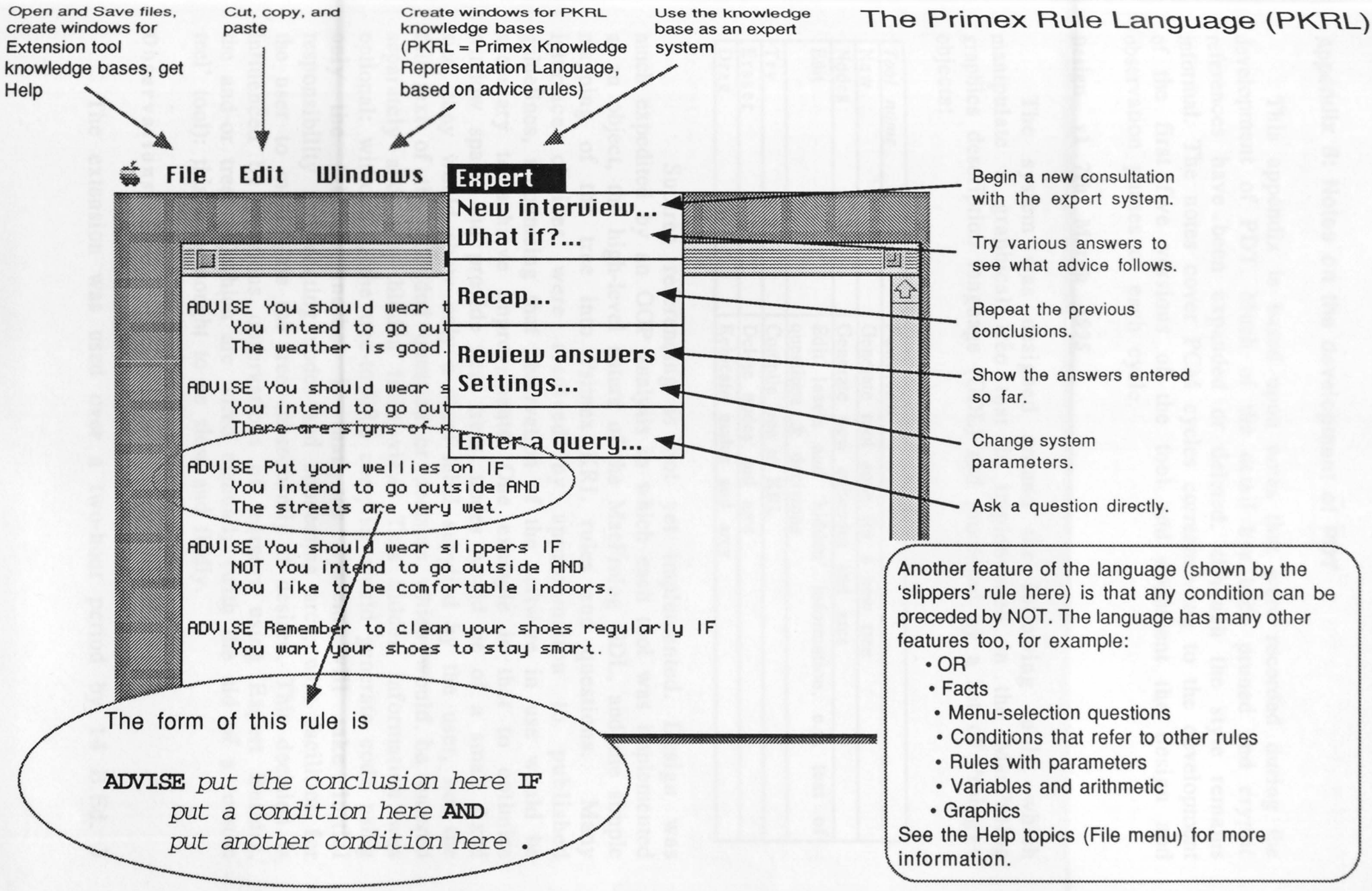
A class can have any number of local features, each of the form Attribute = Value. In addition it normally inherits the combined features of its ancestors.

direction of inheritance →

Fishes — sea fish — cod

Fishes	sea fish	cod
Local feature: *has fins = true*	Local feature: *hunter = trawlermen* *habitat = sea*	Local features: *food = sprats*
Inherited features: *none*	Inherited features: *has fins = true*	Inherited features: *has fins = true* *hunter = trawlermen* *habitat = sea*

The Primex Rule Language (PKRL)



Appendix 8: Notes on the development of PDT

This appendix is based upon notes that were recorded during the development of PDT. Much of the detail has been pruned and cryptic references have been expanded or deleted, although the style remains informal. The notes cover PCM cycles corresponding to the development of the first five versions of the tool and document the design and observation phases of each cycle.

Design v1 7th March 1995

The system was designed around the following tools, which manipulate a graphical tree that is implemented in the MacProlog graphics description language (GDL) and processed by a set of Prolog++ objects:

Tool name	*Function*
New	Generate root node for a new tree
Nodes	Generate new subnodes and arcs
Edit	Edit labels and 'hidden' information, e.g. text of questions & decisions
Try	Compile tree to KRL
Eraser	Delete nodes and arcs
Drag	Relocate nodes and arcs

Subtree referencing is not yet implemented. Design was much expedited by an OOP analysis in which each tool was implemented as an object, the high-level nature of the MacProlog GDL, and the simple mapping of the tree into Primex KRL rules and questions. Many interface choices were resolved by approximation to published guidelines, recognising that observation of the software in use would be necessary to achieve improvements. One example is that to optimise window space and provide neat trees, nodes would be of a small fixed size; they would show only a short label supplied by the user, not the full text of the intended question or decision which would be entered separately and kept hidden from view. This hidden information was optional: without it, the tree-to-KRL compiler would generate code using only the labels. Another is that the system will take initial responsibility for locating nodes and connecting arcs, with facilities for the user to manipulate the tree subsequently if desired. This decision is influenced by previous observations of learners using Expert Builder, the and-or trees of which are wired manually (with the aid of a 'cotton reel' tool): this was thought to be slow and fiddly.

Observations

The extension was used over a two-hour period by 14 B.Ed. 3

students on 7th March 1995. Students were given a twenty-minute demonstration of the software. They were then asked to build classification models in two domains (shapes and fish) for which specifications were provided. Most students were fairly successful in these tasks. The following observations were recorded which led to recommendations for revision.

Observation	*Comment*	*Options for v2*
1 Click on arc label with Edit tool occasionally selected a different arc.	Bug in edit tool.	Fix bug.
2 Period or KRL keyword in label causes Try tool to generate illegal KRL.	Bug in tree-to-KRL compiler.	Fix bug.
3 Users exhibit hesitation in selection of tools. New is sometimes used incorrectly instead of Nodes.	Tool pane has two columns — students expected an adjacent pair of tools to be functionally related (they aren't). New is the first tool users must select, but they may never need it thereafter!	Redesign tool pane, perhaps: a) only one column b) functional groupings c) Tools clearly distinguished by function (eg eliminate New?) d) Improved tool icons.
4 Users neglect to apply the Try tool after amending tree, then become confused when the shell's 'New interview' command produces an unchanged consultation.	A similar source/object code confusion affected Primex 1 (which had a 'Compile' command) and was resolved by automating that function. The present Try tool invokes the tree-to-KRL compiler. After generating KRL it reminds users to select the 'New interview' command.	One of these: a) Automate the Try tool but keep the reminder. Best option if the user is expected to be concerned with the KRL version of the tree. b) Automate the Try tool and link it directly to the shell's 'New interview' command. Relabel the Try tool to 'Run'. Best option if the user is not expected to be concerned with the KRL version of the tree.
5 Nodes tool asks users how many subnodes are required. They hesitate and then usually answer two. Tool sometimes locates nodes directly on top of existing nodes.	A restriction to binary trees would be unacceptable to more advanced users, but present method is slow. Adopting manual node location and arc connection would probably be much slower.	a) As a default, the tool could provide two subnodes. Only generate the dialogue if a modifier key is pressed. b) Keep the automatic placing but prevent concealment.
6 Users make no edits to the generated KRL and spend little time studying it.	Most users have little experience of using the KRL and editing it could introduce bugs.	Adopt 4 b)
7 Users apply Eraser tool to node in the middle of a tree and inadvertently delete subtree.	Warning could be given but repeated issue may irritate advanced users.	Give warning for non-leaf deletion but suppress if a modifier key is pressed.

8 Edit tool generates a menu that causes users to hesitate. The menu's options are 'Visible label', 'Full question', 'Decision', 'Tree name'. Each option leads to a second dialogue containing an edit field.	The distinction between visible and non-visible information is not well understood. 'Tree name' means nothing to users, who have not been told about the intention to provide decomposition support.	Eliminate non-visible text. Edit tool should lead immediately to the edit field dialogue with radio buttons for 'Question' (default) 'Decision' and 'Tree name'. Show full input on tree: implement variable-sized nodes.

Design v2 14th March 1995

The main changes relative to v1 are:

Aim of change	*See note*	*Nature*
Bug fix	1	Edit tool now processes arc clicks correctly.
Bug fix	2	Try tool now intercepts KRL reserved symbols and consistently transforms them into non-reserved symbols.
Bug fix	5	Nodes tool now checks for coincident nodes.
Usability	3	Tool pane now has only one column. The New tool is now redundant - on creation a decision tree window is given an (unerasable) root node and two subnodes. (This implies a commitment of one tree per window - not thought to be a serious restriction since the number of windows is limited only by memory).
Usability	4	Try tool is now 'Run' as per 4 b).
Usability	5 a	New tool generates a dialogue only if a modifier key is pressed.
Usability	7	As per recommendation.
Usability	8	As per recommendation.

Observations

This version was used over a two-hour period on 14th March 1995 by 10 B.Ed. 3 students, seven of whom had participated in the previous trial. Students were given a ten-minute demonstration of the software's revised features. They were then asked to build classification models in two domains. Students devoted between 30 minutes and 80 minutes to this task. All built successful trees.

Observation	*Comment*	*Options for v3*
1 Long questions are truncated in nodes.	Nodes are of fixed width and variable 1-3 rows.	a) Increase number of rows allowed, or b) Increase column width, or c) No change.

2 Users can't save the tree, only the KRL.	Primex Save command is part of the shell core.	Implement special load/save of tree.
3 Quotes in nodes/arcs generate illegal KRL.	Quotes are KRL constant delimiters.	Filter out or transform quotes.
4 Users confused by question marks in decisions when they did not type them.	Question marks are automatically added to question nodes. If these are changed to decision nodes, the question mark will remain unless edited.	Strip question marks from decision nodes.
5 User clicked with rubber tool on arc label. System silently ignored the click. User confused.	Rubber tool acts only on nodes.	Incorrect clicks should give warnings.
6 New windows coincide with old - user 'loses' a window.		Auto-shift new windows.
7 Run tool with default node attributes generates invalid KRL.		Reset default values or generate compile-time message.
8 Duplicate window names cause unexpected error.		Intercept and flag duplicate names.

Once again it was observed that users made little or no reference to the generated KRL (except where invalid KRL had been produced which led to error messages). In discussion, students said that they thought the KRL should be hidden or made visible only by selection (at present the KRL window is brought to the front when the code is generated from the tree).

Design v3 21st March 1995

Now implements subtree referencing. The other main changes relative to v2 are:

Aim of change	*See note*	*Nature*
1 Usability	1	Nodes are now 10% wider.
2 Usability	2	Trees can be printed and tree files can be loaded and saved.
3 Bug fix	3	Quotes now filtered correctly.
4 Usability	4	System now ensures that question nodes always have question marks - other kinds of nodes do not.
5 Usability	5	Incorrect clicks (eg with rubber tool outside of a node) now give warnings.
6 Usability	-	Draft User Notes produced.
7 Usability	-	At runtime, tree nodes are highlighted.

Subtree referencing was implemented by compiling the run tree with extended branches where Tree Name nodes were encountered.

Tree Names are expected to appear as window names for the tree's definition. Compile-time checks detect cases of a) Tree Name nodes referring to nonexistent windows b) Tree Name arcs that don't appear as decisions in the named tree, as well as c) A question node has no answers (i.e. is a leaf). Runtime node highlighting is generated by adding SYSCALL conditions to the KRL. The extensions codefile has grown to 177K. Primex would not load it until its Finder allocation was increased from 3200K to 3400K.

Observations

Students (8) from the same group as previously were issued with the Draft User Notes and given a 10-minute demonstration of the subtree referencing ('Forest') capability. This was motivated by claims that a) large trees are difficult to test, debug, maintain and understand and b) forests provide scope for collaborative work. Students were then invited to work in pairs or triads to build a forest. In fact all students worked in pairs. Each pair build a model in 35-60 minutes. Typically these had three trees, each containing 3-15 nodes, one tree being a small 'control' tree with leaves that named the other two. This was similar to the example that had been demonstrated (the demonstration did include a Tree Name node as a mid-tree node, with arcs labelled by subtree decisions, but none of the students used this style). Mostly individual subtrees were developed separately, tested, saved, and loaded into one computer for testing in combination. All ran correctly apart from one which failed to compile in combination ('evaluation stack full' - this was the biggest example). Students responded well to the task and spoke enthusiastically of classroom possibilities. Most asked for copies to try out on teaching practice or at home.

Observation	*Comment*	*Options for v3*
1 Evaluation stack full error.	Change to a larger stack size would require a change to Primex.	Make the change, plus others that are due, or change the predicate that overgrows the stack.
2 Window name with a question mark mismatched Tree Name node label. The two could not be renamed consistently. The user had to redraw the tree in a correctly named window.	No facility for changing a window name, and a Tree Name node isn't allowed to contain a question-mark.	Allow windows to be renamed or Allow Tree Name nodes to have a question-mark or Automatic creation of subwindows when Tree Name nodes are created (but should changing the node type later delete the window??)

3 Node highlighting fails to show when subtree window comes to front.	Highlighting is implemented by transient picture, ie is deleted on refresh.	Improve or remove highlighting (students unsure of its value, runtime shuffling of tree windows is slow and unsettling). NB SYSCALL makes KRL less readable and ties it to this XTN.
4 Print prints all windows	-	Prompt for choice of windows to print.
5 Windows reload to previous size even on smaller monitor computers.		Reload to screen size max.
6 Tree windows concealed by others on top.		Implement tree browsing tool - at least to follow links?
7 Tree size statistics must be read manually.		Implement autostats tool - maybe combine with tree renaming tool? or tool to measure KRL code size, so portable across XTNs?
8 Tree exceeds window size - students forced to scroll (slowly!) to inspect tree.	Larger nodes of present system make this more likely. A good sign that students produce big trees so quickly?	Zoom tool.

Node highlighting in its present form is not worth the implementation effort. It slows the consultation, can be distracting (especially when windows 'shuffle' with linked trees), transient highlights are sometimes wiped, and the KRL is less readable and less portable. Some students say that they are well able to relate inferencing to the trees without the support of node highlighting — it may even be desirable that they have to actively figure out why the system asks the questions it does.

Design v4 14th April 1995

Now implements an option for a domain lexicon (attribute and value names) to be imported and for comparison of knowledge models via a new 'Menu' tool. The other main changes relative to v3 are:

Aim of change	*See note*	*Nature*
Usability	1	Cuts added to compiler. Now uses evaluation space more efficiently - 'shopping' example compiles OK.
Usability	2	Tree name nodes may (but need not) now contain question marks.
Usability	3	Run time node highlighting removed.
Usability	4	Dialogue now offers selective window printing.

Usability	5	Window sizes now reduced on reload if necessary.
Usability	8	New tool scales drawing area.
Usability		Double-click help added to all tools and extended for many.
Usability		Default node and labels changed. Node labels were Qn_N, arc labels were yes/no. This was suspected of confusing or misleading users. All labels now use bullet strings '•••' by default.
Usability		Many tools given additional functions, mostly controlled by modifier keys. Eg: for Edit tool, shift-clicking cycles through yes/no on arc labels or through model arc/node labels if a model is installed; Arrow tool if clicked on tree name node or subtree root node now creates new window or wfronts named window.

In addition, changes have been made to Primex. The evaluation space is unchanged (see note 1 above) but the Finder space has been increased to 3500K. A new version of Primex has been generated.

Observations

The prototype was tested with two groups of PGCE students (Computing & Geography), each for about one hour. The Computing group were invited to save, load and compare models. Numerous problems were identified and recorded in the Menu tool. In spite of these the Computing students generally appreciated the motivation for the knowledge base comparison facility. Model comparison was not mentioned to the Geography group, who spent the time creating and running trees. The Geographers generally had little difficulty in creating trees and proposed some applications from their own subject. Tony Shallcross, the Geography tutor, was present and proposed that his materials on decision-making for a proposed second Forth Bridge could be one. The Compare tool was discussed with Mandy Haggith in DAI and compared to her metalevel framework for reasoning about inconsistency.

Design v5 1st May 1995

Design changes include better shell integration, removal of the distinction between the two types of decision tree file, KB comparison command now accessed via CMP tool, removal of 'Menu' tool and elimination of most redundant recompilation. This version has been demonstrated to Doug Urquart who will now conduct formative trials with pupils in his school.

Appendix 9: Induction versus naïve compilation of a factor table

The factor table below is based on an example in ecological modelling provided by Robertson (1992). In (1) we show the Primex KRL which is generated by PFT when the compiler uses the induction method (i.e. when the Options dialogue shows the 'Smart' setting). In (2) we show the KRL which is generated by the more naïve compilation method (i.e. the 'Left to right' setting). It can be seen that induction generates 8 rules compared to 18 for the naïve method. Also, the `Altitude` factor disappears entirely in the induction-generated rules, half of which also make no reference to the `Grazing` factor, whereas every rule generated by the naïve method makes reference in column order to all the table's factors. Both methods generate code that correctly classifies each example in the table, but the runtime behaviour differs markedly because of the differences in the PKRL representation. Compilation time is similar for both methods.

Altitude	Rainfall	Soil type	Grazing	Erosion hazard
low	low	sand	low	false
high	low	sand	low	false
low	med	sand	low	true
high	med	sand	low	true
low	high	sand	low	true
high	high	sand	low	true
low	high	sand	high	true
high	high	sand	high	true
low	low	sand	high	true
high	low	sand	high	true
low	low	clay	low	false
high	low	clay	low	false
low	low	clay	high	false
high	low	clay	high	false
low	med	clay	low	false
low	med	clay	high	false
low	high	clay	low	false
low	high	clay	high	true

(1) Primex KRL generated by induction-based compilation

```
ADVISE true IF
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Rainfall' [low, med, high] med.
ADVISE true IF
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Rainfall' [low, med, high] high.
ADVISE false IF
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Rainfall' [low, med, high] low.
ADVISE false IF
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Rainfall' [low, med, high] med.
ADVISE false IF
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Grazing' [low, high] low.
ADVISE true IF
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Grazing' [low, high] high.
ADVISE false IF
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Rainfall' [low, med, high] high AND
    Qn : 'Grazing' [low, high] low.
ADVISE true IF
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Rainfall' [low, med, high] high AND
    Qn : 'Grazing' [low, high] high.

Qn : _Prompt _Choices _Ans  IF
    SELECTION FROM menu : _Prompt _Choices IS _Ans.

QUESTION menu : _Prompt _Choices
ASK  '': _Prompt
CHOOSE ONE OF _Choices.
```

(2) Primex KRL generated by naïve compilation

```
ADVISE false IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] low.
ADVISE false IF
    Qn : 'Altitude' [high, low] high AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] low.
ADVISE true IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] med AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] low.
```

```
ADVISE true IF
    Qn : 'Altitude' [high, low] high AND
    Qn : 'Rainfall' [low, med, high] med AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] low.
ADVISE true IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] high AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] low.
ADVISE true IF
    Qn : 'Altitude' [high, low] high AND
    Qn : 'Rainfall' [low, med, high] high AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] low.
ADVISE true IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] high AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] high.
ADVISE true IF
    Qn : 'Altitude' [high, low] high AND
    Qn : 'Rainfall' [low, med, high] high AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] high.
ADVISE true IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] high.
ADVISE true IF
    Qn : 'Altitude' [high, low] high AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Soil type' [sand, clay] sand AND
    Qn : 'Grazing' [low, high] high.
ADVISE false IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Grazing' [low, high] low.
ADVISE false IF
    Qn : 'Altitude' [high, low] high AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Grazing' [low, high] low.
ADVISE false IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Grazing' [low, high] high.
ADVISE false IF
    Qn : 'Altitude' [high, low] high AND
    Qn : 'Rainfall' [low, med, high] low AND
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Grazing' [low, high] high.
ADVISE false IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] med AND
```

```
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Grazing' [low, high] low.
ADVISE false IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] med AND
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Grazing' [low, high] high.
ADVISE false IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] high AND
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Grazing' [low, high] low.
ADVISE true IF
    Qn : 'Altitude' [high, low] low AND
    Qn : 'Rainfall' [low, med, high] high AND
    Qn : 'Soil type' [sand, clay] clay AND
    Qn : 'Grazing' [low, high] high.

Qn : _Prompt _Choices _Ans  IF
    SELECTION FROM menu : _Prompt _Choices IS _Ans.

QUESTION menu : _Prompt _Choices
ASK  '': _Prompt
CHOOSE ONE OF _Choices.
```

Appendix 10: A laddering session with PCT

Below is the transcript of a laddering session in which PCT in its default configuration (i.e. 'Ignore features on first pass' and 'Depth-first development') interviews the user. The interview is conducted through a dialogue box (not shown) which contains Yes, No, Skip, Help, and Stop buttons, together with a text area in which the system displays questions, an edit field into which the user types answers, and a menu that may be prefilled with items by the system. Not all these items need be present or enabled in the dialogue box at any time. The dialogue box is adjacent to the model window in which the classification tree diagram is displayed and incrementally updated as the interview progresses. In the following, we show the interaction sequence including updates to the model diagram. Initially the diagram consists of a single node with the default label `Things`.

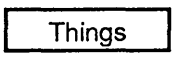


Figure A10.1

1. PCT (shows Figure A10.1)>> Name the class of things you wish to describe
2. USER (typing)>> Fishes

Figure A10.2

3. PCT (shows Figure A10.2)>> Can you name some subclasses or examples of Fishes?
4. USER (clicking)>> Yes
5. PCT>> Enter the name(s) of subclasses or examples of Fishes.
6. USER (typing)>> sea fish and river fish

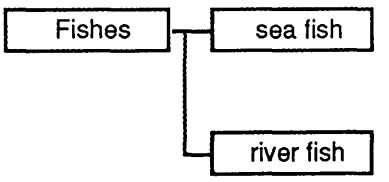


Figure A10.3

7. PCT (shows Figure A10.3, highlights sea fish)>> Can you name some subclasses or examples of sea fish?
8. USER (clicking)>> Yes
9. PCT>> Enter the name(s) of subclasses or examples of sea fish.
10. USER (typing)>> salmon, cod and halibut

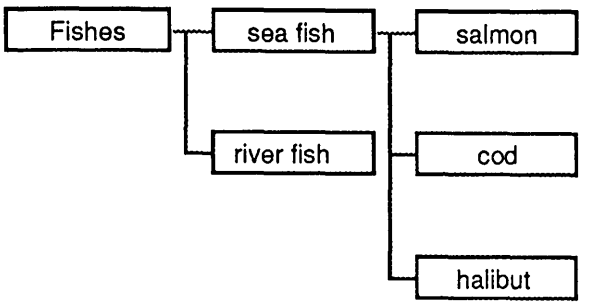


Figure A10.4

11. PCT (shows Figure A10.4, highlights salmon)>> Can you name some subclasses or examples of salmon?
12. USER (clicking)>> No
13. {Repeat interaction 11&12 for cod and halibut}
14. PCT (highlights river fish)>> Can you name some subclasses or examples of river fish?
15. USER (clicking)>> Yes
16. PCT>> Enter the name(s) of subclasses or examples of river fish.
17. USER (typing)>> salmon, carp, trout

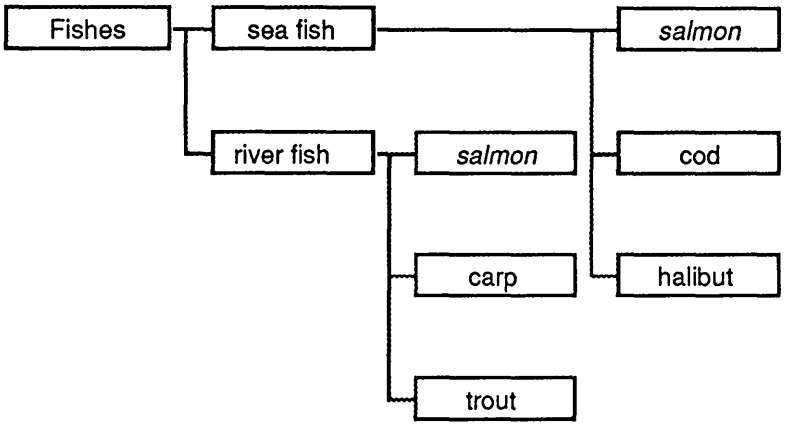


Figure A10.5

18. PCT (shows Figure A10.5, highlights salmon)>> Can you name some subclasses or examples of salmon?
19. USER (clicking)>> Yes
20. PCT>> Enter the name(s) of subclasses or examples of salmon.
21. USER (typing)>> Sammy

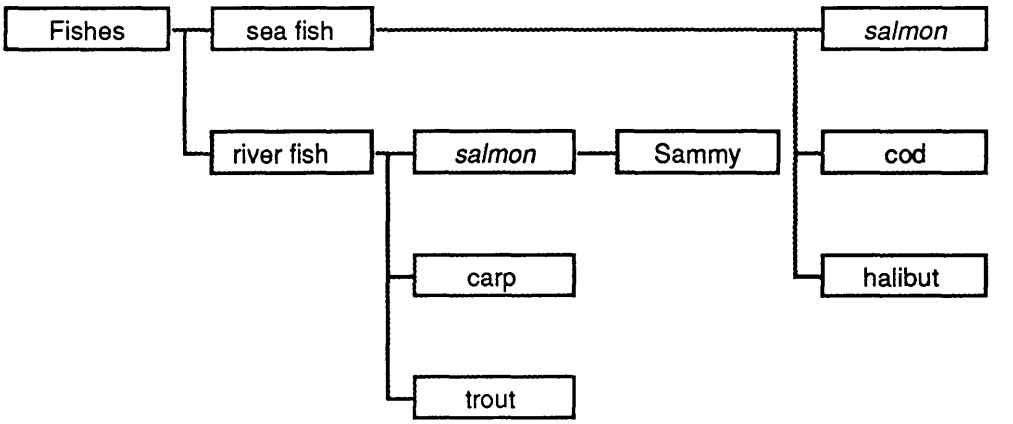


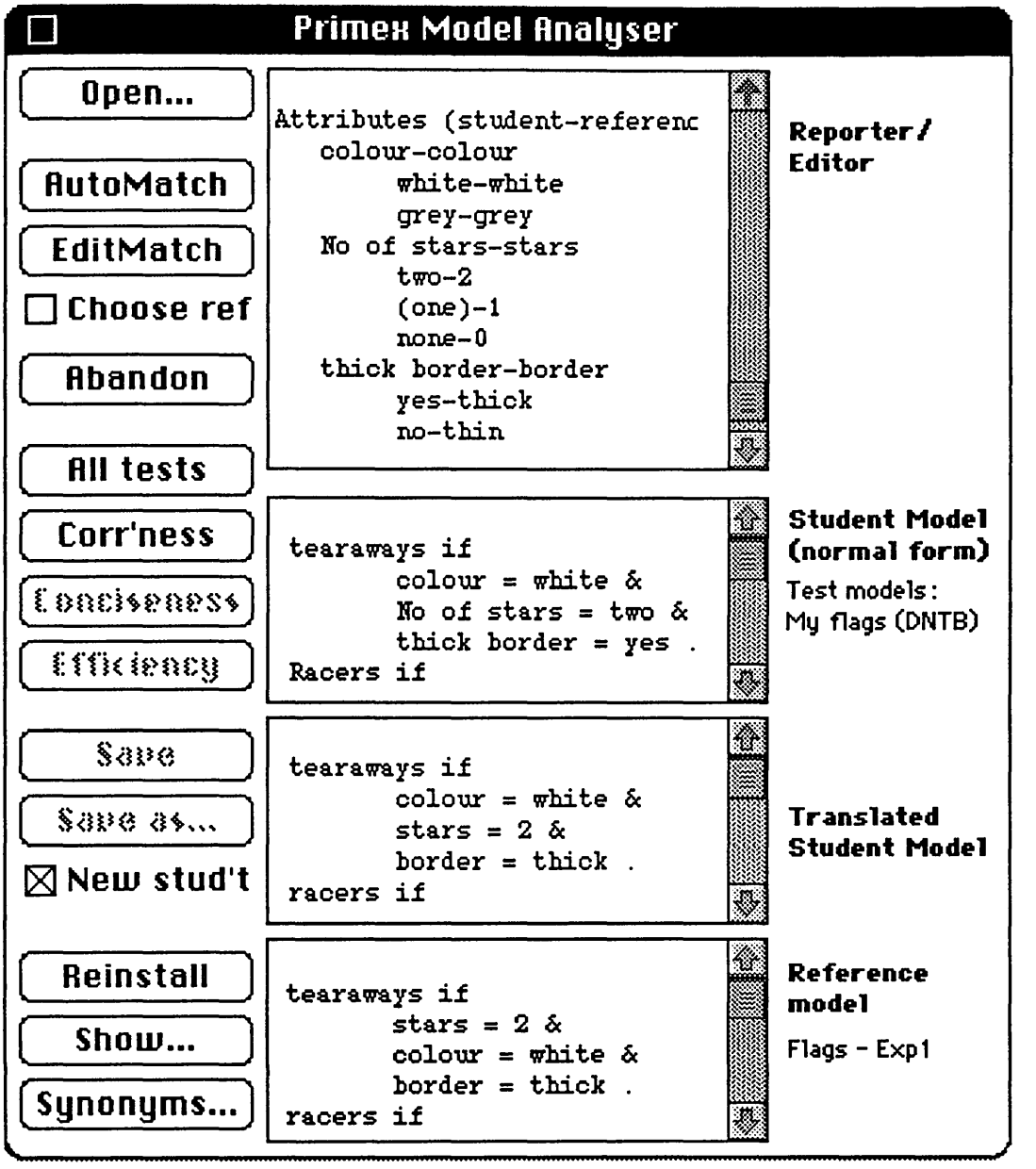
Figure A10.6

22. PCT (shows Figure A10.6, highlights Sammy)>> Can you name some subclasses or examples of Sammy?
23. USER (clicking)>> No
24. PCT (highlights carp)>> Can you name some subclasses or examples of carp?
25. USER (clicking)>> No
26. {Repeat 24&25 for trout}
27. PCT (highlights Fishes)>> Can you describe some features of Fishes?
28. USER (clicking)>> Yes
29. PCT>> Describe some features of Fishes.
30. USER (typing)>> habitat=water
31. PCT (shows river fish and sea fish in menu)>> Can you name some more subclasses or examples of Fishes other than the ones shown?
32. USER (clicking)>> No
33. PCT (highlights river fish, shows habitat=water in menu)>> Is this feature true for river fish?
34. USER (clicking)>> Yes
35. PCT>> Can you describe some features of river fish other than the ones shown?
36. USER (clicking)>> Yes
37. PCT>> Describe some more features of river fish.
38. USER (typing)>> water type = fresh and food = insects.
39. PCT>> So river fish is a special kind of Fishes that has water type = fresh and food = insects. Is that correct?
40. USER (clicking)>> Yes
41. PCT (shows salmon, carp and trout in menu)>> Can you name some more subclasses or examples of river fish other than the ones shown?
42. USER (clicking)>> No
43. PCT (highlights trout, shows water type = fresh and food = insects and habitat=water in menu)>> Are all these features true for trout?
44. USER (clicking)>> Yes
45. PCT>> Can you describe some features of trout other than the ones shown?
46. USER (clicking)>> No
47. PCT>> So trout is a kind of river fish that has no special features. Is that correct?
48. USER (clicking)>> Yes
49. PCT>> Can you name some subclasses or examples of trout?
50. USER (clicking)>> No
51. {Repeat 43-50 for carp}
52. PCT (highlights salmon, shows water type = fresh and food = insects and habitat=water in menu)>> Are all these features true for salmon?
53. USER (clicking)>> Yes
54. PCT>> Can you describe some features of salmon other than the ones shown?
55. USER (clicking)>> No
56. PCT>> So salmon is a kind of river fish that is also a kind of sea fish. Is that correct?
57. USER (clicking)>> Yes
58. PCT (shows Sammy in menu)>> Can you name some more subclasses or examples of salmon other than the ones shown?
59. USER (clicking)>> No
60. {Repeat 43-50 for Sammy}
61. PCT (highlights sea fish, shows habitat=water in menu)>> Is this feature true for sea fish?
62. USER (clicking)>> Yes
63. PCT (shows insects, 'Something else', and 'Does not apply' in menu)>> What is the value of food for sea fish?

64. USER (clicking)>> Does not apply
65. PCT (shows fresh, 'Something else', and 'Does not apply' in menu)>> What is the value of water type for sea fish?
66. USER (clicking)>> Something else
67. PCT>> Enter the value of water type for sea fish.
68. USER (typing)>> salty
69. PCT (shows water type = salty and habitat=water in menu}>> Can you describe some features of sea fish other than the ones shown?
70. USER (clicking)>> Yes
71. PCT>> Describe some more features of sea fish.
72. USER (typing)>> hunter=trawlerman
73. PCT>> So sea fish is a special kind of Fishes that has water type = salty and hunter=trawlerman. Is that correct?
74. USER (clicking)>> Yes
75. PCT (shows halibut, cod and salmon in menu)>> Can you name some more subclasses or examples of sea fish other than the ones shown?
76. USER (clicking)>> No
77. {Repeat 43-50 for halibut and cod}
78. PCT (highlights salmon, shows 'Keep only salty', 'Keep only fresh', 'Keep both', 'Supply a new value' in menu)>> There are two values of water type for salmon. What do you want to do?
79. USER (clicking)>> Keep both
80. PCT (shows food = insects, hunter = trawlerman, water type = salty, habitat = water and water type = fresh in menu)>> Are all these features true for salmon?
81. USER (clicking)>> Yes
82. PCT>> Can you describe some features of salmon other than the ones shown?
83. USER (clicking)>> No
84. PCT>> So salmon is a kind of river fish that is also a kind of sea fish. Is that correct?
85. USER (clicking)>> Yes
86. PCT (shows Sammy in menu)>> Can you name some more subclasses or examples of salmon other than the one shown?
87. USER (clicking)>> No
88. PCT>> No more questions!

Appendix 11: The Primex Model Analyser

The user interface to the PMA is shown below. The dialogue has been narrowed to fit within the confines of the page.



A typical session proceeds as follows.

(1) When the PMA is first installed it loads into memory from a designated folder the available Reference Models, Performance Models, and the Synonyms Database.

(2) The user selects via 'Open...' the model file for analysis. The Reporter/Editor field shows some file details, including the model's status as runnable/non-runnable. A model's status as non-runnable will be revealed by the absence of a normal form

representation within the model file. If such a representation is present, it is displayed in the Student Model field. The dialogue above shows that the model under analysis has the filename 'My Flags'. The model's file type is 'DNTB', revealing that the model was created by the Primex factor table tool (PFT).

(3) The user selects 'AutoMatch'. The PMA Model Matcher responds by trying to match the categories and features of the model against those of the installed Reference Models. During this process the user may be queried to supply certain correspondences (see the example dialogue below). In the dialogue shown above, PMA has identified 'Flags - Exp1' as the best-fit reference model. Attribute mappings, value mappings, and category mappings between this model and the student model are shown in the Reporter/Editor field. The Translated Student Model field shows the result of rewriting the Student Model using these mappings. Had the AutoMatch process failed, or if it succeeded but the user was not satisfied with the mappings, there is the option to use 'EditMatch'. With 'EditMatch' the user manually edits the Student Model in the Reporter/Editor field, prior to resubmitting it to the matching process.

(4) The user selects 'All Tests', or equivalently, uses the single test buttons individually to generate for the model the quality measures of correctness, conciseness, and efficiency. Sample output from these tests, which normally appears in the Reporter/Editor field, is shown at the end of this appendix.

(5) Optionally, the user clicks 'Save' to save the results of the analysis. The PMA maintains two files: a 'log' file containing copies of the information that appears in the dialogue fields and a 'data' file containing information in 24 fields, as shown in the table below. The table uses abbreviations as follows: RM - Reference Model; PM - Performance Model; SM - Student Model.

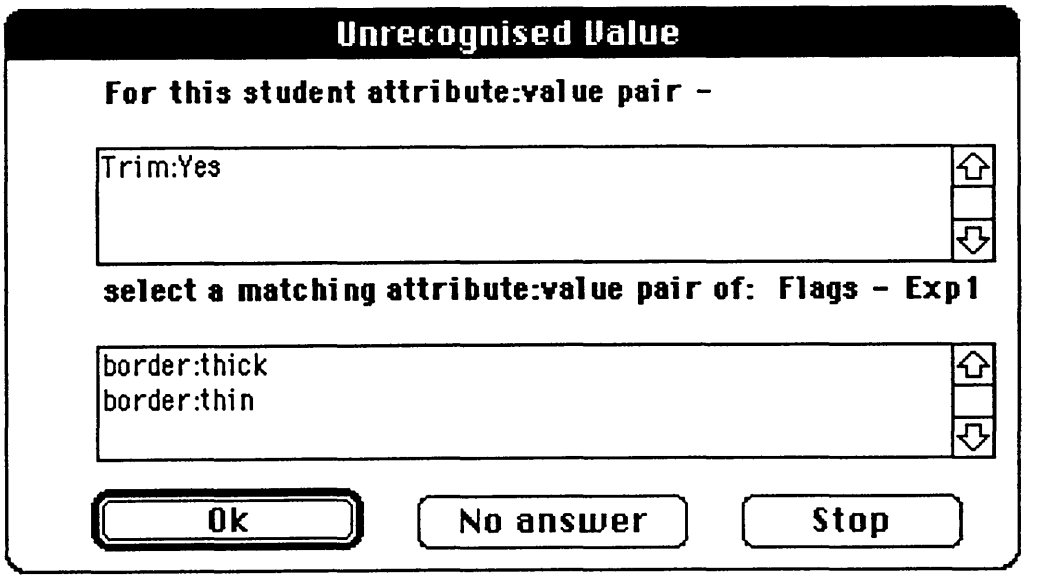
Field	*Contents*	*Type*
1	Analysis reference	atom
2	Model file name	atom
3	File type (PDT,PFT,PCT,KRL give DNTR,DNTB,FRTR,PMXS)	atom
4	Functionality (0=non-runnable, 1=runnable)	integer
5	Domain (prefix of reference model file name)	atom
6	No. of Questions answered in normalisation	integer
7	Model was changed by manual editing (0=no, 1=yes)	integer
8	No. of Gaps in model-matching mappings	integer
9	No. of Defined RM classes	integer
10	No. of Defined SM classes	integer
11	No. of SM unrecognised classes	integer
12	No. of SM fully correct classes	integer

13	No. of SM buggy classes	integer
14	No. of SM missing classes	integer
15	No. of SM rules	integer
16	No. of SM phrases	integer
17	No. of SM distinct symbols	integer
18	No. of SM phrases in correct part of model	integer
19	No. of SM distinct symbols in correct part	integer
20	No. of PM phrases in corresponding part	integer
21	No. of PM symbols in corresponding part	integer
22	No. of Qns for all SM conclusions	integer
23	No. of Qns for SM conclusions for correct part	integer
24	No. of Qns for PM conclusions for corresponding part	integer

(6) Other facilities provided by the PMA include provision to: maintain the Synonyms database ('Synonyms' button); inspect library models and other data ('Show' button); create new files for another student ('New student' check box); and select manually a model to be used as a reference model rather than allowing the system to make its own choice of a library model ('Choose ref' check box).

Sample question asked by the Model Matcher

In this example, the normal form representation of a student model for the Flags domain (see Appendix 12) contains the feature `Trim=Yes`. The PMA's database of synonyms contains an entry which equates `Trim` to `border`, but the reference model for the domain (viz., Flags-Expl) associates the attribute `border` only with the values `thick` and `thin`. The PMA's Model Matcher cannot decide which of these two values gives the appropriate mapping for the student's feature. Hence the following question is generated:



Sample Output from 'All tests'

In this example, a student model for the Flags domain is being analysed as at step (4) above.

```
** CORRECTNESS **

Reference model refers to 10 classes.

Student model refers to 10 classes.
 9 are fully correct.
 1 is buggy.

Fully correct classes: bullfrogs, popeyes, racers,
rambos, redsocks, runners, scouts, tearaways,
wolves

Buggy classes: skins

In the interpretation:
    stars=2
    colour=grey
    border=thin
the ref. model validates skins
but stu. model validates nothing

** CONCISENESS **

Total student model = 10 rules 40 phr 20 sym

Restricting to the student model's 9 fully correct
classes:
   • Student model takes 36 phr  19 sym
   • 'Flags - performance' takes 34 phr 19 sym

** EFFICIENCY **

To generate all student conclusions takes 30 questions.
 To generate the 9 fully correct classes:
   • 'Flags - performance' asks 25 questions.
   • Student model asks 27 questions.
```

Appendix 12: Specifications of model domains

These are the domains used in the large-scale trials. Note that children were not asked to build all types of models in all domains and some of the domains were used only by teachers for expository purposes.

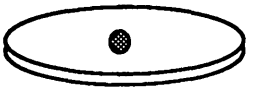
DOMAIN: CAR WASH

Build a model to describe the car wash tokens pictured below. When your model is run the computer should ask you about the features of a token. Depending on the answers, the computer should be able to name the type of car wash that has been requested.

Basic

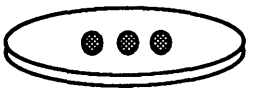
Regular

Regular Plus



Luxury

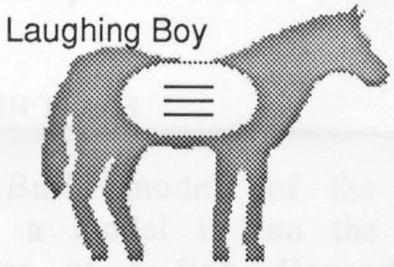
Luxury Double Foam

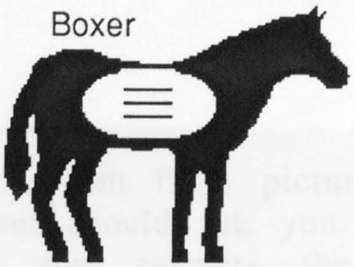


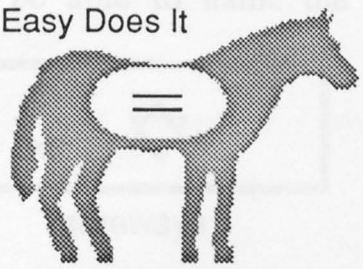
Luxury Deluxe

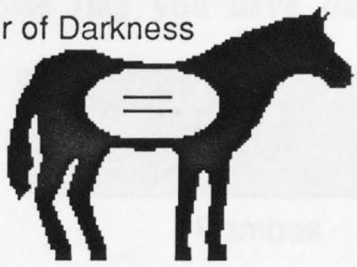
DOMAIN: HORSES

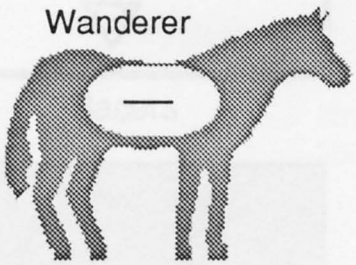
Build models of the race horses pictured below. When a model is run the computer should ask you about the features of a race horse. Depending on your answers, the computer should be able to name the horse that you have described.

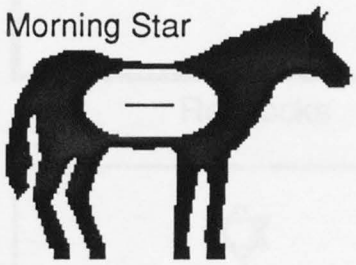












DOMAIN: WIDGETS

Build models of the information below. When a model is run the computer should ask you about the features of a microwidget. Depending on your answers, the computer should be able to name the microwidget's type.

Microwidgets come in three colours: black, red and green.

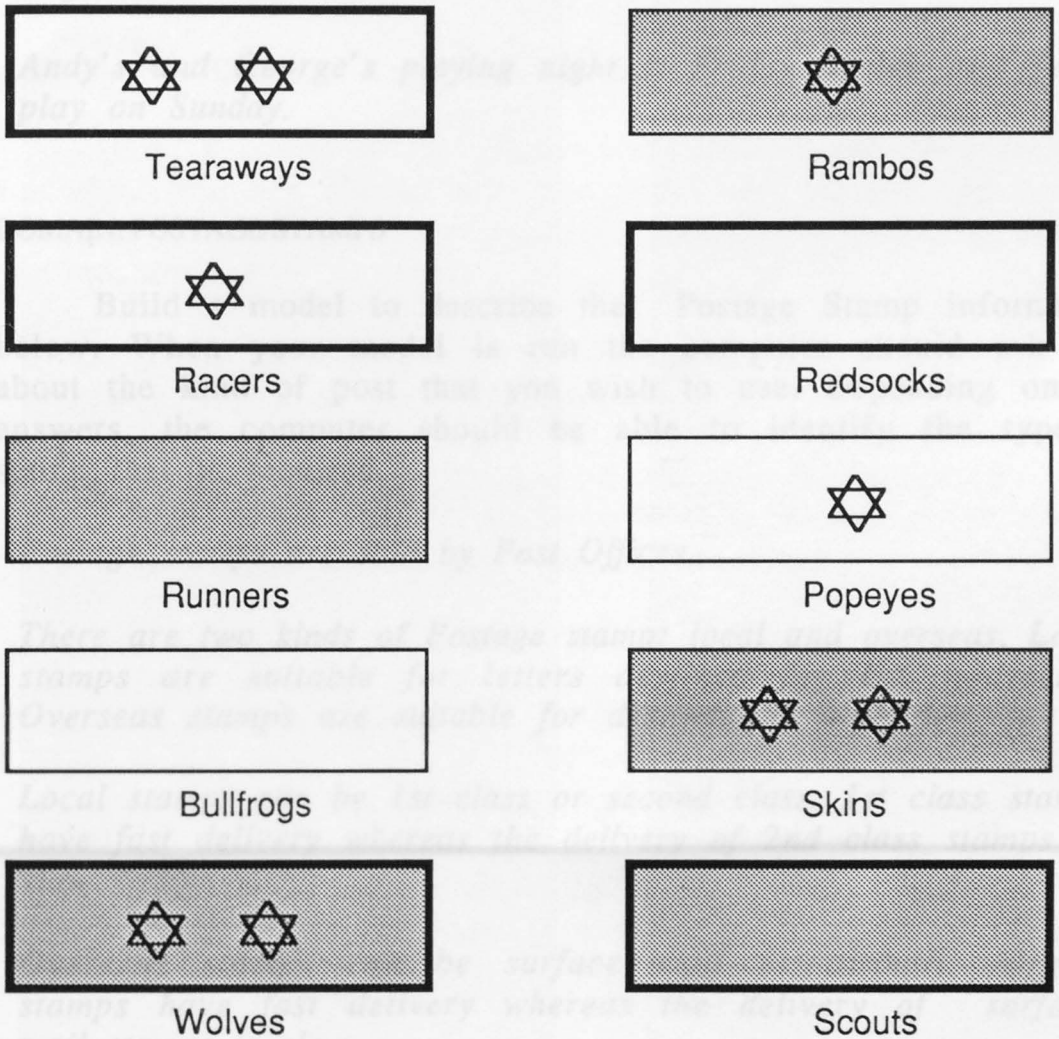
Black microwidgets that are wide in diameter are type 43, those that are narrow in diameter are type 44.

Red microwidgets may be long or short: those that are long are type 102, short ones that have flanges are type 107 and short ones without flanges are type 108.

Green microwidgets have edges that are plain, serrated or sharp. Those with plain edges may be made of steel (type 202), bronze (type 203), or copper (type 204). Green microwidgets with serrated edges are type 300. Sharp-edged green microwidgets sometimes have fluted rims; these are type 500, the non-fluted ones are type 501.

DOMAIN: FLAGS

Build models of the baseball team flags pictured below. When a model is run the computer should ask you about the features of a flag. Depending on your answers, the computer should be able to name the team whose flag you have described.



DOMAIN: DARTS

Build a classification tree model to describe the Darts Club information below. When your model is run the computer should ask you questions about the Club members. Depending on the answers, the computer should be able to name the player.

Darts Club members meet in the Red Lion Hotel.

There are three classes of member: seniors, regulars and beginners. Seniors play with darts that are black in colour, regulars play with red darts and beginners play with darts that are white.

Andy is a senior. George and Helen are regulars. Tony is a beginner.

Andy's and George's playing night is Friday. Helen and Tony play on Sunday.

DOMAIN: POSTAGE STAMPS

Build a model to describe the Postage Stamp information below. When your model is run the computer should ask you about the kind of post that you wish to use. Depending on the answers, the computer should be able to identify the type of stamp that is required.

Postage stamps are sold by Post Offices.

There are two kinds of Postage stamp: local and overseas. Local stamps are suitable for letters destined for UK addresses. Overseas stamps are suitable for destinations abroad.

Local stamps can be 1st class or second class. 1st class stamps have fast delivery whereas the delivery of 2nd class stamps is slow.

Overseas stamps can be surface mail or airmail. Airmail stamps have fast delivery whereas the delivery of surface mail stamps is slow.

DOMAIN: BOINGS

Build models of the information below. When your model is

run the computer should ask you about the features of a space creature. Depending on the answers it should be able to identify the creature's type.

On the planet Blip there lives a species known as the Boings.

There are three kinds of Boing: Ards, Bards and Cards.

Bards are yellow. Ards and Cards are both red in colour but differ in the texture of their skin: Ards have a rough skin texture whereas the skin texture of the Cards is smooth. There are two kinds of Ards and these are known as Acks, which are tall in height, and Adders which are short. There are also two kinds of Bards. Bisons are Bards with a shape that is blobby. Bipods are Bards with a shape that is angular.

DOMAIN: PONDOIDS

Build a model of the information below. When your model is run the computer should ask you about the features of a pond creature. Depending on the answers it should be able to identify the creature's type .

In the ponds of the country of Venusia there lives a family of creatures known as the Pondoids.

There are four kinds of Pondoid. Nittles are the largest, Poggers are smaller than Nittles but bigger than Wombats, Wombats are third largest and Clippers are the smallest.

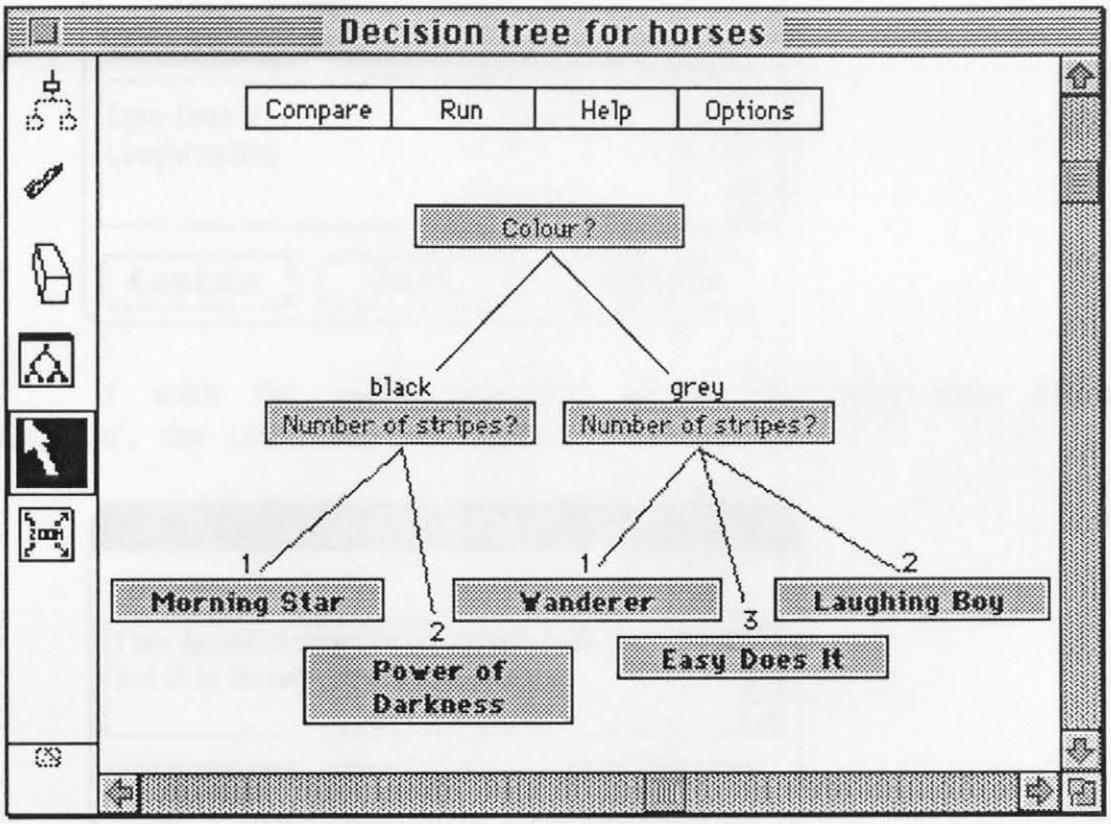
Nottles and Nuttles are kinds of Nittle that are distinguished by the colour of their spots: Nottles have orange spots whereas Nuttles have blue spots.

Wombats have three subspecies with names Woms, Wans and Wins: all Woms have tails and toes, Wans are tailless and have toes, and Wins have neither tails nor toes. A Win is an exceptionally large Wombat, in fact it is the same size as a Nittle.

There is a type of Pondoid known as a Plod which is both a kind of Clipper and a kind of Wan. In size Plods are more like Clippers than they are like Wans. Plods have poisonous blood and they eat Poggers when they are hungry.

Appendix 13: Using the Primex Model Comparator

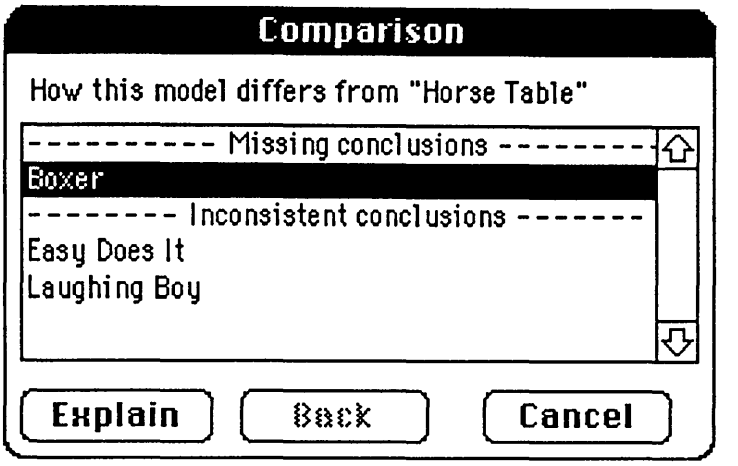
The PMC is implemented as a Primex extension which, when installed, automatically adds a 'Compare' command button to PDT, PFT and PCT model windows. Below is a PDT window with the button present. The model shown is a buggy model of the Horses domain (see Appendix 12).



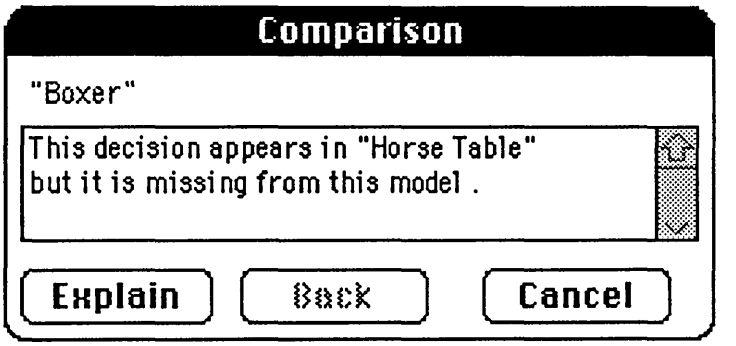
If the model builder now wishes to compare this model to one constructed by a peer, he or she clicks the 'Compare' button and selects the other model via a standard file selection dialogue. Suppose that the file selected for comparison is named 'Horse table' and contains the following PFT model:

Colour?	Number of stripes?	Name of horse
grey	1	Wanderer
grey	2	Easy Does It
grey	3	Laughing Boy
black	1	Morning Star
black	2	Power of Darkness
black	3	Boxer

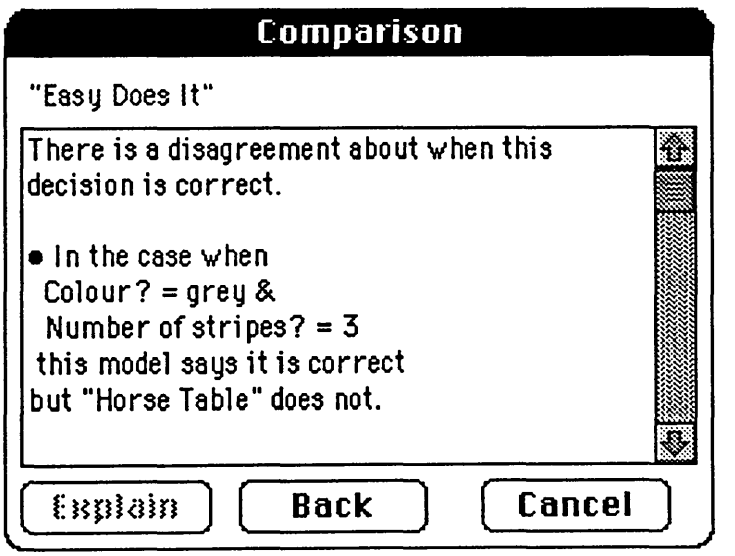
The PMC then analyses the two models for differences. No human intervention is required. In this case, the analysis produces a dialogue like the one below (in which 'this model' refers to the decision tree):



If with the menu selection shown the user now clicks 'Explain', the following explanation is generated:



If however the user selects 'Easy does it', the explanation is:



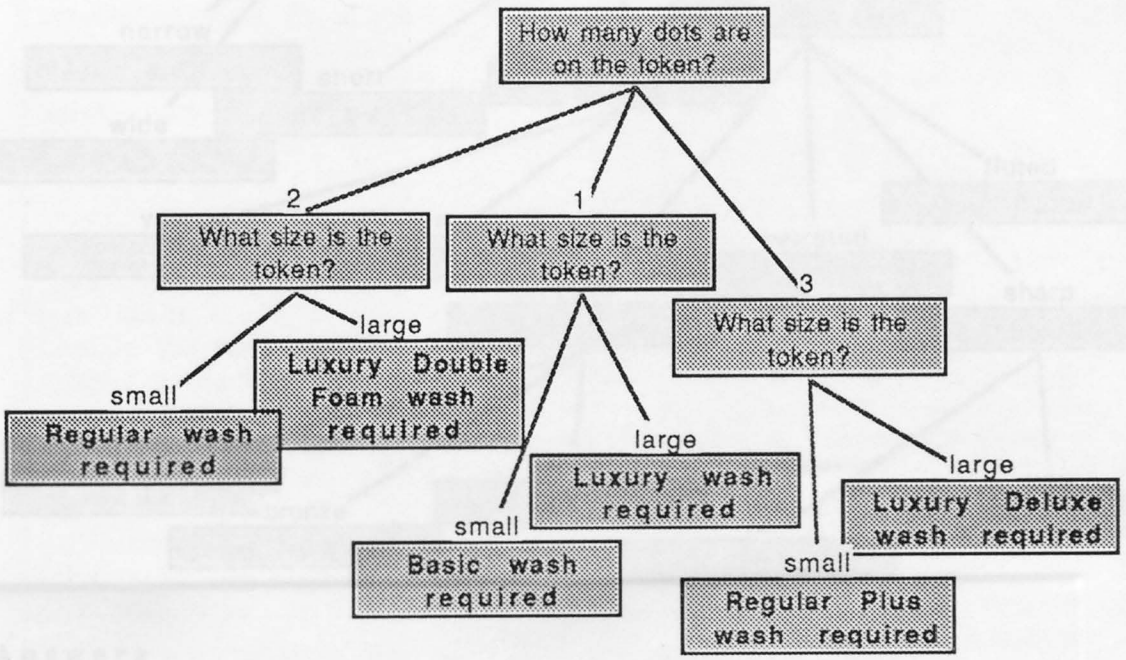
Appendix 14: Sample tasks used in the evaluation of the PMC

Pupils were assessed on four tasks, each requiring a model to be debugged (if necessary — not all models were buggy) to bring it into line with a given specification. Specifications and models from Task 1 and Task 4 are shown below. Tree diagrams have been narrowed to fit within the page. Correct answers are shown at the end of the appendix.

Task 1 — Specification

Size	*Number of dots*	*Type of wash*
small	1	Basic
small	2	Regular
small	3	Regular Plus
large	1	Luxury
large	2	Luxury Double Foam
large	3	Luxury Deluxe

Task 1 — Model

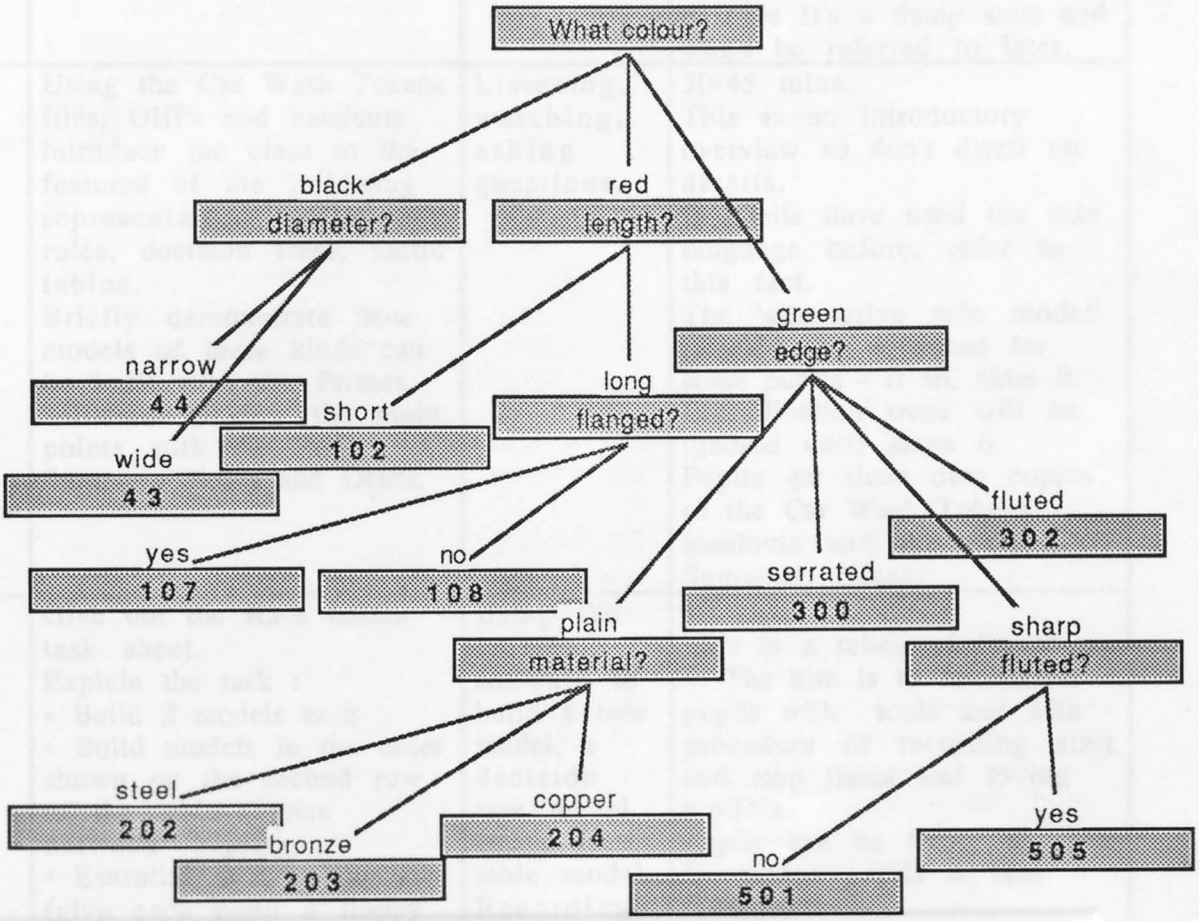


Task 4 — Specification

Microwidgets come in three colours: black, red and green. Black microwidgets that are wide in diameter are type 43, those that are narrow in diameter are type 44.

Red microwidgets may be long or short: those that are long are type 102, short ones that have flanges are type 107 and short ones without flanges are type 108.
Green microwidgets have edges that are plain, serrated or sharp. Those with plain edges may be made of steel (type 202), bronze (type 203), or copper (type 204). Green microwidgets with serrated edges are type 300. Sharp-edged green microwidgets sometimes have fluted rims; these are type 500, the non-fluted ones are type 501.

Task 4 — Model



Answers

In Task 1, the given model is correct. No repair is necessary.

In Task 4, the given model contains 3 bugs which can be repaired as follows. '505' should be edited to '500'. For red widgets, the 'long' and 'short' labels should be reversed. A nonexistent class '302' should be deleted.

Appendix 15: Teacher's briefing notes for large-scale trials

These notes have been reformatted from landscape format.

	Your activities	*Pupils' activities*	*Time (estimate)* *Notes*
1	Administer Initial Task. Explain that this will give us useful information but not to worry if it's hard to understand (it's not a test!).	Attempt to complete task.	10-30 mins. No surprise if pupils struggle with this. Don't help them too much. Be prepared to retrieve fairly quickly if little progress. Maybe best done a few periods in advance of the subsequent material, because it's a damp start and won't be referred to later.
2	Using the Car Wash Tokens files, OHPs and handouts introduce the class to the features of the following representation techniques: rules, decision trees, factor tables. Briefly demonstrate how models of these kinds can be built with the Primex software. Review the main points with the Primex Summary Notes and OHPs.	Listening, watching, asking questions.	30-45 mins. This is an introductory overview so don't dwell on details. If pupils have used the rule language before, refer to this fact. The 'alternative rule model' may be too advanced for some pupils - if so, skim it. Classification trees will be ignored until stage 6. Pupils get their own copies of the Car Wash Tokens handouts and the Primex Summary Notes.
3	Give out the Race horses task sheet. Explain the task : • Build 3 models each • Build models in the order shown on the second row of the table (varies between pupils) • Essential to save models (give each pupil a floppy disk with name on disk label?). • To avoid confusion, use 'File' 'Close' to remove each old model before starting a new one. • Fill in the table correctly • A well-built model is more important than a fast finish ('it's not a race').	Using the Primex software to build a rule model, a decision tree model and a factor table model. Recording build times. Saving models.	60 mins. This is a rehearsal for stage 4. The aim is to familiarise pupils with tools and with procedure of recording start and stop times and saving models. Pupils can be freely assisted. Encourage pupils to test models. When building the rule model, pupils can choose to build a basic or advanced version. Afterwards, pupils get their own copies of the Race horses handout (shows correct solutions) to take away. If convenient, a pupil who completes this stage could proceed directly to stage 4.

4	Give out the Flags task sheet. The tasks are similar to Race horses but it may be helpful to repeat the above instructions. If you wish, consult the Model solutions disk for correct models.	Same as stage 3.	60 mins. This is the first task that will be analysed experimentally. Pupils can refer to previous handout if they wish. Try to ration help to pupils who reach an impasse (not always feasible!). Encourage pupils to test models.
5	Repeat stage 3 for the Widgets task sheet.	Same as stage 3.	60 mins. Again, ration help. However pupils may be reminded the asterisk (wildcard) will be useful in making the factor table model.
6	Using the Postage Stamps file, OHPs and handouts introduce the class to the classification tree representation technique. Briefly demonstrate how the model can be built with the Primex software. Review the classification tool with the Primex Summary Note and OHP.	Listening, watching, asking questions.	40 mins. Pupils get their own copies of the Postage Stamps handout and the Primex Summary Note.
7	Give out the Darts Club task sheet.	Using the Primex software to build a class'n tree model. Recording build time. Saving the model.	20 mins. This is a rehearsal for stage 8. The aim is to familiarise pupils with the classification tool. Pupils can be freely assisted. Afterwards, pupils get their own copies of the Darts Club handout (shows correct solution) to take away.
8	Give out the Planet Blip task sheet. Remind pupils again to build their models in the order shown on the second row of the table (varies between pupils). If you wish, consult the Model solutions disk for correct models.	Using the Primex software to build all four kinds of model Recording build times. Saving models.	60 mins. Ration help. However pupils may be reminded the asterisk (wildcard) will be useful in making the factor table model. Encourage pupils to test models.
9	Repeat stage 8 for the Pond Creatures task sheet.	Same as stage 8.	60 mins. Maybe best to view this as an extension task for high-achievers. Perhaps it to a close when all pupils have finished stage 8? Afterwards, pupils get their own copies of the Pond Creatures handout.

10	Administer Pupil's Questionnaire. Instructions: • Write name in top right. • Put one tick on each row. • Comments welcome — write them on reverse of sheet. Meantime complete the Teacher's questionnaire.	Complete questionn-aire.	10 mins. May be helpful to explain the language, e.g. 'I found the software easy to use' refers to software for that type of model, not the entire Primex package.
11	Administer Final Task. Explain that this will give us useful information but it's not a test.	Attempt to complete task.	20-30 mins. The aim is to assess ability to select and apply a suitable representation of information — hopefully there will be a gain over stage 1. Don't help them too much. Discourage conferring.

Appendix 16: Conciseness and economy of models

These tables show the differences between model types that were identified in the conciseness and economy of models constructed in the large-scale school trials at at least a 5% level of significance.

(1) Conciseness

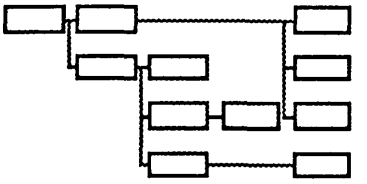
Domain	*Model types compared*	*No. of pairs*	*Z=*	*p =*	*Type with higher mean*
Flags	PDT vs PKRL	41	-2.9578	0.0031	PRKL
Widgets	PFT vs PDT	15	-2.2014	0.0277	PDT
Widgets	PFT vs PKRL	14	-2.2014	0.0277	PKRL
Boings	PFT vs PKRL	27	-3.4320	0.0006	PRKL
Boings	PCT vs PFT	27	-2.0699	0.0385	PCT
Boings	PDT vs PFT	28	-3.5740	0.0004	PDT

(2) Economy

Domain	*Model types compared*	*No. of pairs*	*Z=*	*p =*	*Type with higher mean*
Horses	PFT vs PKRL	43	-2.7231	0.0065	PKRL
Horses	PDT vs PKRL	48	-3.3322	0.0009	PKRL
Horses	PFT vs PDT	48	-2.1917	0.0284	PFT
Flags	PFT vs PDT	39	-2.4318	0.0150	PFT
Widgets	PDT vs PKRL	15	-2.5558	0.0106	PKRL
Widgets	PFT vs PKRL	14	-3.2958	0.0010	PKRL
Boings	PFT vs PDT	28	-3.5011	0.0005	PDT
Boings	PDT vs PCT	28	-2.5799	0.0099	PCT
Boings	PCT vs PFT	27	-3.2857	0.0010	PCT
Boings	PFT vs PKRL	27	-4.5407	<0.00005	PKRL
Boings	PCT vs PKRL	28	-2.7212	0.0065	PKRL
Boings	PDT vs PKRL	28	-4.3724	<0.00005	PKRL

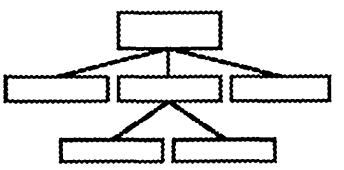
Appendix 17: Pupil questionnaire

For convenience, the questionnaire has been reformatted from landscape to portrait mode. In the original, the model diagrams were arranged vertically down the left edge of the page.

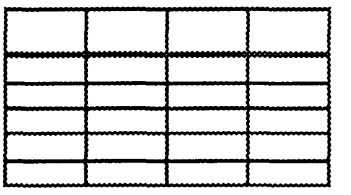


CLASSIFICATION TREE MODELS

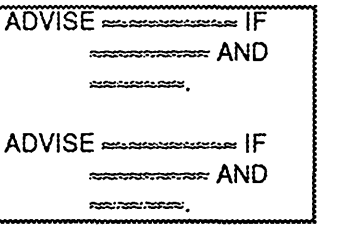
	No, very strongly disagree	No, strongly disagree	No, mildly disagree	Neutral feelings	Yes, mildly agree	Yes, strongly agree	Yes, very strongly agree
CLASSIFICATION TREE MODELS							
I enjoy building these models							
I understand the ideas behind them							
I find the software easy to use							
I would like to use this software again							
DECISION TREE MODELS							
I enjoy building these models							
I understand the ideas behind them							
I find the software easy to use							
I would like to use this software again							
FACTOR TABLE MODELS							
I enjoy building these models							
I understand the ideas behind them							
I find the software easy to use							
I would like to use this software again							
RULE MODELS							
I enjoy building these models							
I understand the ideas behind them							
I find the software easy to use							
I would like to use this software again							



DECISION TREE MODELS



FACTOR TABLE MODELS



RULE MODELS

Appendix 18: Teacher questionnaire with written responses

The questionnaire contained a multiple-choice section and a section in which free-format comment was invited in response to a set of questions. These sections are shown in A) and B) below, with teachers' written comments included in B). For convenience, the questionnaire has been reformatted from landscape to portrait mode. In the original, the questionnaire contained illustrations of the four types of model diagram.

A) The questionnaire

	No, very strongly disagree	No, strongly disagree	No, mildly disagree	Neutral feelings	Yes, mildly agree	Yes, strongly agree	Yes, very strongly agree
CLASSIFICATION TREE MODELS							
Pupils enjoy building these models							
Pupils understood the ideas behind them							
Pupils' models were of high quality							
I would like to use this software again							
DECISION TREE MODELS							
Pupils enjoy building these models							
Pupils understood the ideas behind them							
Pupils' models were of high quality							
I would like to use this software again							
FACTOR TABLE MODELS							
Pupils enjoy building these models							
Pupils understood the ideas behind them							
Pupils' models were of high quality							
I would like to use this software again							
RULE MODELS							
Pupils enjoy building these models							
Pupils understood the ideas behind them							
Pupils' models were of high quality							
I would like to use this software again							

B) Teachers' written responses to set questions

(1) How successful has this trial been in convincing you that children will benefit from using a variety of knowledge-based tools other than rule systems?

JM: I have found the trial very useful and shall continue to use the materials with classes.
LJ: Very, anything not purely text-based generates more enthusiasm.
RG: Not nearly enough time or experience on my part to even formulate a view.
AC: The graphical tools were more intuitively accessible to the pupils.

(2) What advice would you offer to other teachers contemplating using the new set of Primex tools?

JM: Do the tasks first yourself and make sure you have read everything thoroughly. Worksheets/helpsheets will aid pupil understanding.
LJ: Spend more time on teaching theory and use of classification models. The concept is harder for pupils to understand.
RG: Make sure you have plenty of time to try out software before using with class.

(3) What was the biggest problem that you experienced in the course of the trial?

JM: Minor bugs and the frustration caused by factor tables refusing to save (Error 23).
LJ: The long time taken to load on an LCII or III, inability to save an incomplete factor table.
RG: Technical difficulties — software wouldn't run, took ages to load etc.
AC: Trying to complete the tasks in the short time available — because of the end of term.

(4) What was the greatest source of satisfaction?

JM: A real understanding (and entertainment) caused by the rather odd (!) examples.
LJ: Seeing the great satisfaction of some pupils for models other than rule models.
RG: When offered a free choice on last period of term (after trial was complete) some pupils wanted to complete or begin more Primex models.
AC: Pupils working well — concentrating and enjoying the tasks.

Appendix 19: Pre-course and Post-course tests

Formatting has been changed to reduce space requirements, but of course the text of the questions is unchanged.

(A) Pre-course test

For each of 1-3 below, draw a diagram or table to represent the information differently. If you can't think of any suitable diagram or table, just leave the space blank.

(1) Grading of Eggs

Eggs that are white in colour and large in size are Grade 1.
Eggs that are brown in colour and large in size are Grade 2.
Eggs that are white in colour and medium in size are Grade 3.
Eggs that are brown in colour and medium in size are Grade 4.
Eggs that are white in colour and small in size are Grade 5.
Eggs that are brown in colour and small in size are Grade 6.

(2) Light bulb types

A factory makes two kinds of light bulbs: Tungsten and Neon. Tungsten bulbs come in three types, known as Standard, Extra-life, and Sky. There are two kinds of Neon bulb: Industrial and Domestic. Fancy and Radiant are the two kinds of Domestic neon bulbs that are available at present.

(3) Circuit board testing

When testing a circuit board that may be faulty, the first question to ask is: Is there a power signal at the input? If the answer is 'no' then the fault will be a power supply fault. If the answer is 'yes' then the next question becomes: How many volts at the output? If there is less than 20 volts then the fault is an amplifier failure. If there is between 20 and 40 volts then the circuit board is OK. If there is more than 40 volts then it is necessary to ask if there are signs of burnout. If so then the fault is a power overload, otherwise the fault is a short in the wiring.

(B) Post-course test

For each of 1-3 below, draw a diagram or table to represent the information differently. If you can't think of any suitable diagram or table, just leave the space blank.

(1) Types of Cake

A cake shop sells two kinds of cake: Plain and Fancy. Plain cakes come in three types, known as Sponge, Filled, and Extra. There are two kinds of Fancy cake: Cream and Iced. Coconut Ice and Doubled Ice are the two kinds of Iced cakes that are available at present.

(2) Identifying a worzel

When identifying a worzel, the first question to ask is: Does it have a long tail? If the answer is 'no' then the worzel will be a Forest worzel. If the answer is 'yes' then the next question becomes: How many whiskers? If there are fewer than 20 whiskers then the worzel will be a Northern Mink. If there are between 20 and 40 whiskers then the worzel will be a Stumpie. If there are more than 40 whiskers then it is necessary to ask if the worzel has a black nose. If so then the worzel is a Black Nosed Worzel, otherwise it is a Common Worzel.

(3) Placing books onto shelves[1]

Books that have paperback covers and are tall in height are placed onto shelf A.
Books that have hardback covers and are standard in height are placed onto shelf B.
Books that have paperback covers and are tall in height are placed onto shelf C.
Books that have hardback covers and are standard in height are placed onto shelf D.
Books that have paperback covers and are tall in height are placed onto shelf E.
Books that have hardback covers and are standard in height are placed onto shelf F.

---

[1]This question contained an error. By analogy with question (1) of the pre-test, the heights of books in successive sentences should read: *tall, tall, standard, standard, short, short.* The version shown is the one that was answered by pupils.

Appendix 20: M.Ed. Assignment on knowledge-based modelling

Task

Select a curriculum topic relevant to your own teaching and design, implement and evaluate an application of knowledge-based modelling that could support learning in the topic. Write a report to describe the experience and prepare a presentation which could be used to outline your teaching approach for the benefit of colleagues attending an INSET session.

Submission

The submission will consist of a report of approximately 2000 words. Slide masters for the presentation will appear in an appendix. A disk containing relevant examples of your models and where possible also those of children will be attached.

Criteria

A satisfactory submission will be one which:

(a) Presents model files of your own construction that demonstrate your proficiency in the use of knowledge-based modelling software, featuring at least three of the following kinds of model: rule models; decision tree models; classification tree models; factor table models; and-or tree models.

(b) Reports on an application of knowledge-based modelling to a curriculum topic in a way that: describes the context; identifies teaching and learning goals; outlines and justifies the methods used; presents observations and interpretations of children's experiences; and evaluates the outcomes.

(c) Includes comparisons between different kinds of model, identifying benefits and limitations with reference to the classroom experiences.

(d) Concludes with a section that draws on the development to critically appraise the general potential of knowledge-based modelling in the curriculum.

(e) Demonstrates knowledge of recent and relevant research.

(f) Provides 5-10 computer-prepared slides which can support a clear, coherent, and stimulating INSET presentation of approximately 20 minutes. You will be expected to perform a version of this presentation at the class to be held on March 28th 1996.

Structure for the report

A possible (but not mandatory) structure for the main text of the report is as follows.

Section 1: The topic

Describes what the topic is, identifies its place in the curriculum (e.g. with 5-14 or SEB references), and explains why it is plausible to think that knowledge-based modelling (KBM) could contribute to teaching and learning in the topic (e.g. by referring to your own experience of weaknesses in the standard approach, your own experiences with KBM, claims that have been made in the research literature for KBM). *250 words*

Section 2: Preparing a KBM activity

Identifies the teaching and learning aims for your KBM innovation, describes your plan for introducing the activity, selects the kinds of models (rule/dtree/ctree/ftable/aotree) pupils will construct, justifies your approach (e.g. in terms of your knowledge of the children's learning, classroom pragmatic aspects, approaches reported in research literature, etc), illustrates the models which you expect children to make or adapt by providing your own specimen files, and identifies your criteria for assessing the success of the innovation (e.g. successful model building by pupils, observable increases in pupil motivation, high levels of on-task discussion, high test scores, etc). *700 words*

Section 3: Classroom experiences

Describes what you observed during the activity; interprets these observations (e.g. in terms of children's learning, features of the model-building environment, features of the tasks you set, research reports of previous classroom experiences); applies your success criteria to evaluate the quality of the innovation. *800 words*

Section 4: Reflection and conclusion

Drawing on (1), (2), (3), assess the benefits and limitations of the different kinds of model used; summarise the qualities of your own innovation; offer recommendations to others who may contemplate a similar development; broadly appraise the general potential of KBM to other areas and stages of the curriculum. *250 words*

References

Aikins, J. (1983). *Prototypical knowledge for expert systems.* Artificial Intelligence, 20, 163-210.

Alvey, P. (1983). *Problems of designing a medical expert system.* Proceedings of Expert Systems, 83, 20-42.

Anderson, J. (1988). *The Expert Module.* In: Polson, M. & Richardson, J. (Eds) *Foundations of Intelligent Tutoring Systems.* Hillsdale, New Jersey: Lawrence Erlbaum. pp21-54.

Bielaczyc, K. (1993). *The Content and Strategic Features of Collaborative Explanations and Their Impact on Achievement.* In: Terveen, L. (Ed). *Proceedings of the Workshop on Collaborative Problem Solving: Theoretical Frameworks and Innovative Systems.* AIED 93: World Conference on Artificial Intelligence in Education, Edinburgh.

Bird, D., Conlon, T. & Swanson. J. (1996). *Computing and Information Technology in Higher Still: Let's get IT right.* Scottish Educational Review Vol. 28 No. 1 pp3-15.

Birmingham, W. & Klinker, G. (1993). *Knowledge-acquisition tools with explicit problem-solving models.* The Knowledge Engineering Review, Vol 8:1, 5-25.

Birtwistle, G., Dahl, O., Myhrhaug, B. & Nygaard, K. (1979). *SIMULA Begin.* Bromley, UK: Chartwell-Bratt.

Boos-Bavnbeck, B. (1991). *Against Ill-founded, Irresponsible Modelling.* In: Niss, M., Blum, W. and Huntley, I. (Eds) *Teaching of Mathematical Modelling and Applications.* Chichester: Ellis-Horwood, pp. 70-82.

Boose, J. & Bradshaw, J. (1987). *Expertise transfer and complex problems: using Aquinas as a knowledge acquisition workbench for expert systems.* International Journal of Man-Machine Studies, 26, 3-28.

Boose, J. (1988). *Uses of repertory grid-centred knowledge acquisition tools for knowledge-based systems.* International Journal of Man-Machine Studies Vol 29 pp287-310.

Boose, J. (1989). *A survey of knowledge acquisition techniques and tools.* Knowledge Acquisition 1, 3-37.

Brachman, R. (1985). *'I lied about the trees' Or, Defaults and Definitions in Knowledge Representation.* The AI Magazine. Fall issue.

Briggs, J. (1984). *MITSI Rules!* Logic Programming Associates Ltd, Royal Victoria Patriotic Building, Trinity Road, London SW18 3SX.

Briggs, J. (1987). *Learning with Expert Systems: a Starter Pack.* The Advisory Unit for Microtechnology in Education, Hertfordshire.

Bruner, J. (1960). *The Process of Education.* Cambridge, Ma: Harvard University Press.

Bruner, J., Goodnow, J. & Austin, G. (1956). *A Study of Thinking.* New York: John Wiley.

Burton, A., Shadbolt, N., Hedgecock, A., & Rugg, G. (1988a). *A formal evaluation of Knowledge Elicitation Techniques for Expert Systems: Domain 1.* In Moralee, D. (Ed). *Research and Development in Expert Systems IV.* BCS Workshop Series. Cambridge: Cambridge University Press.

Burton, A., Shadbolt, N., Rugg, G. & Hedgecock, A. (1988b). *Knowledge Elicitation Techniques in Classification Domains.* In Kodratoff, Y. (Ed). *Proceedings of the 8th European Conference on AI.* London: Pitman.

Carnegie Group (1985). *Knowledge Craft 3.0 Reference Manual.* Pittsburgh PA: Carnegie Group.

Carr, D. (1994). *5-14: A Philosophical Critique.* In Kirk, G. & Glaister, R., (Eds). *5-14: Scotland's National Curriculum.* Edinburgh: Scottish Academic Press.

Chandrasekaran, B. (1986). *Generic Tasks in knowledge-based Reasoning: High level building blocks in expert system design.* IEEE Expert, Fall.

Cheng, P. (1993). *Multiple interactive structural representation systems for education.* In: Cox, R., Petre, M., Brna, P., & Lee, J. (Eds). *Proceedings of the Workshop on Graphical Representations, Reasoning and Communication.* AIED 93: World Conference on Artificial Intelligence in Education, Edinburgh.

Chipman, S. (1993). *Gazing Once More into the Silicon Chip: Who's Revolutionary Now?* In: Lajoie, S. & Derry, S. (Eds) *Computers as Cognitive Tools.* Hillsdale NJ: Erlbaum.

Clancey, W. (1982). *Tutoring rules for guiding a case method dialogue.* In: Sleeman, D. and Brown, J. S. (Eds) *Intelligent Tutoring Systems.* New York: Academic Press.

Clancey, W. (1983). *The epistemology of a rule-based expert system: a framework for explanation.* Artificial Intelligence, 20, 215-51.

Clancey, W. (1985). *Heuristic Classification.* Artificial Intelligence, 27, 289-350.

Clancey, W. (1987). *From GUIDON to NEOMYCIN to HERACLES in twenty short lessons.* In: van Lamsweerde, A., & Dufour, P. (Eds) *Current Issues in Expert Systems.* London: Academic Press.

Clancey, W. (1988). *The Knowledge Engineer as Student.* In Mandl, H. and Lesgold, A. (Eds) *Learning Issues for Intelligent Tutoring Systems.* Springer-Verlag 1988.

Clancey, W. (1992). *Representations of Knowing: In Defense of Cognitive Apprenticeship.* Journal of Artificial Intelligence in Education, 3(2), 139-168.

Clancey, W. (1993). *Guidon-Manage Revisited: A Socio-Technical Systems Approach.* Journal of Artificial Intelligence in Education, 4(1), 5-34.

Claris Corporation (1994). *Claris Works User Guide.* 5201 Patrick Henry Drive, Santa Clare, California 95052.

Clark, K. & McCabe, F. (1984). *micro-Prolog: Programming in Logic.* New Jersey:Prentice-Hall.

Collins, A., Brown, J. & Newman, S. (1989). *Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing, and Mathematics.* In: Resnick, L. (Ed) (1989). *Knowing, Learning and Instructing: Essays in honour of Robert Glaser.* Lawrence Erlbaum.

Conlon, T. (1985). *Start Problem-solving with Prolog.* Wokingham: Addison-Wesley.

Conlon, T. (1991). *Prolog and Educational Software.* In: *Prolog and Knowledge Bases in Schools.* Commission of the European Communities, ECSC-EEC-EAEC, Brussels.

Conlon, T. (1993). *PathFinder: a programming tool for search-based problem solving.* In: Brna, P., Ohlsson, S., & Pain, A. (Eds). *Proceedings of AIED 93: World Conference on Artificial Intelligence in Education,* Edinburgh.

Conlon, T. (1994). *PhraseMaker Tutorial and Reference Guide.* Parallel Logic Programming Ltd., PO Box 49, Twickenham TW2 5PH, UK.

Conlon, T. (1995). *Automated Analysis for Knowledge-based Modelling.* In: Vanneste, P., Bertels, K., & De Decker, B. (Eds) *Proceedings of the Workshop on Automated (Novice) Program Analysis.* AIED 95: World Conference on Artificial Intelligence in Education, Washington DC. pp31-37. Also available as *DAI Research Paper No. 757,* Department of Artificial Intelligence, University of Edinburgh.

Conlon, T. & Bowman, N. (1995). *Expert Systems, Shells, and Schools: Present Practice, Future Prospects.* Instructional Science Vol. 23 Nos 1-3 pp111-131.

Conlon, T. & Cope, P. (1989). *Computing in Scottish Education: the First Decade and Beyond.* Edinburgh: Edinburgh University Press.

Conlon, T. & Pain, H. (1995). *Persistent Collaboration: marrying the technology push with the learning pull.* In: Greer, J. (Ed). *Proceedings of AI-ED 95: World Conference on Artificial Intelligence in Education,* Washington DC. pp437-444.

Conlon, T. & Pain, H. (1996). *Persistent Collaboration: A Methodology for Applied AIED.* Journal of Artificial Intelligence in Education, Vol 7 No. 3/4 219-252. Also available as *DAI Research Paper No. 790,* Department of Artificial Intelligence, University of Edinburgh.

Cooke, N. (1994). *Varieties of Knowledge Elicitation Techniques.* J. Human-Computer Studies 41, 801-849.

Cornell-Way, E. (1994). *Knowledge Representation and Metaphor.* Oxford: Intellect Books.

Cox, R. & Brna, P. (1993). *Analytical Reasoning with External Representations: The relationship between prior knowledge*

and performance. In: Cox, R., Petre, M., Brna, P., & Lee, J. (Eds). *Proceedings of the Workshop on Graphical Representations, Reasoning and Communication.* AIED 93: World Conference on Artificial Intelligence in Education, Edinburgh.

Cox, R. & Brna, P. (1995). *Supporting the use of external representations in problem solving: the need for flexible learning environments.* Journal of Artificial Intelligence in Education Vol 6 No 2/3 pp 239-302.

CPCS (1991). *KSS0 Manual.* Centre for Person-Computer Studies, 2019 Underhill Drive, Calgary, Alberta, Canada T2N 4E4.

Crabb, D. (1985). *Expert-Ease.* Infoworld, 28th January. pp44-45.

Cumming, G. (1990). *Artificial Intelligence and Images of Natural Learning.* Computers in Education, 319-325.

Cumming, G. (1993). *A perspective on learning for intelligent educational systems.* Journal of Computer Assisted Learning, 9, pp229-238.

Cumming, G. & Abbot, E. (1984). *Making front-ends friendly: designing PROLOG to fit children's minds.* In Nichol, J., Briggs, J. and Dean, J. (Eds) *Prolog, Children and Students.* London: Kogan Page.

Cumming, G. & Abbot, E. (1986). *Exploiting expert systems in education.* In Salvas, A. & Dowling, C. (Eds) *Computers in Education: On the crest of a wave?* Melbourne, Computer Education Group of Victoria. Proceedings of the Fourth Australian Computer Education Conference pp129-134.

Cumming, G. & Self, J. (1990). *Intelligent educational systems: identifying and decoupling the conversational levels.* Instructional Science 19:11-27.

Cypher, A. & Stelzner, M. (1991). *Graphical knowledge-based model editors.* In: Sullivan, J. & Tyler, S. (Eds). *Intelligent User Interfaces.* ACM Press: Addison-Wesley. pp403-420.

Davis, R. (1989). *Expert systems: how far can they go? Part 1.* AI Magazine, Spring, 61-67.

Davies, R. & O'Keefe, R. (1989). *Simulation Modelling with Pascal.* Hemel-Hempstead: Prentice-Hall.

Davison, A. (1991). *Parlog++ : A Parlog Object Oriented Language.* Parallel Logic Programming Ltd, PO Box 49, Twickenham TW2 5PH, United Kingdom.

DES and Welsh Office, (1990). *Technology in the National Curriculum.* London: HMSO.

Drever, E. (1988). *Resource-Based Teaching: the New Pedagogy?* In: Brown, S. & Wake., R. *Education in Transition.* Edinburgh: SCRE (Scottish Council for Research in Education) Publication 102, 86-96.

Durkin, J. (1993). *Expert Systems: Catalog of Applications.* Akron, OH: Intelligent Computer Systems.

Durkin, J. (1994). *Expert Systems: An overview of the field.* PC AI, January/February 37-39.

Dutton, W. (1987). *Decision-making in the Information Age: Computer Models and Public Policy.* In: Finnegan, R., Salaman, G. & Thompson, K. (Eds) *Information Technology: Social Issues.* Sevenoaks: Hodder and Stoughton.

Ehn, P. (1988). *Work-oriented design of computer artifacts.* Stockholm: Arbeslivscentrum.

Eisenstadt, M. & Brayshaw, M. (1988). *The Transparent Prolog Machine (TPM): an execution model and graphical debugger for logic programming.* Journal of Logic Programming 5(4), 277-342.

Emerald Intelligence (1992). *Mahogany Help Desk demonstration disk.* 3915-A1 Research Park Drive, Ann Arbor, MI 48103, USA.

Ennals, R. (1983). *Beginning micro-Prolog.* Chichester: Ellis-Horwood.

Feigenbaum, E. (1977). *The art of artificial intelligence: themes and case studies of knowledge engineering.* Proceedings of the 5th International Joint Conference on Artificial Intelligence, pp 1014-29.

Fraught, W. (1986). *Applications of AI in Engineering.* Computer 19, 17.

Fullan, M. (1991). *The New Meaning of Educational Change.* London: Cassell.

Futo, I. & Szeredi, J. (1982). *A discrete simulation system based on artificial intelligence techniques.* In: Javor (Ed). *Discrete Simulation and Related Fields.* Amsterdam: North-Holland pp135-150.

Galpin, B. (1989). *Expert Systems in Primary Schools.* British Library Research Paper 73, British Library Research and Development Department.

Gammage, P. (1996). *Cloning the Key Stage Kids.* Times Educational Supplement (Primary Update), September 6 1996, p2.

Genesereth, M. (1982). *The role of plans in intelligent teaching systems.* In: Sleeman, D. and Brown, J. S. (Eds) *Intelligent Tutoring Systems.* New York: Academic Press.

Greenbaum, J. & Kyng, M. (1991). *Design at Work: Cooperative design of computer systems.* Hillsdale, NJ: Lawrence Erlbaum.

Greene, T. (1989). *Children's Understanding of Class Inclusion Hierarchies: The relationship between External Representation and Task Performance.* Journal of Experimental Child Psychology 48, 62-89.

Guttag, J. (1991). *Why Programming is Too Hard and What to Do About it.* In: Mayer, A., Guttag, J., Rivest, R., Szolovits, P. (Eds) *Research Directions in Computer Science: an MIT Perspective.* MIT Press.

Haggith, M. (1995). *A Meta-level framework for exploring conflicts in multiple knowledge bases.* Proceedings of AISB-95.

Hensgens, J., van Rosmalen, P., & van der Baaren, J. (1993). *Simulations in Courseware.* In: *Proceedings of PEG93: AI Tools and the Classroom.* Moray House Institute of Education, Edinburgh. pp210-223.

Hinkle, D. (1965). *The Change of Personal Constructs from the Viewpoint of a Theory of Construct Implications.* Unpublished PhD thesis, Ohio State University.

Homa, D. (1984). *On the nature of categories.* In: Bower, G. (Ed). *The Psychology of Learning and Motivation.* New York: Academic Press.

Hoyles, C., Healy, L ., & Pozzi, S. (1994). *Groupwork with Computers: an overview of findings.* Journal of Computer Assisted Learning 10, 202-215.

Intellicorp (1984). *The Knowledge Engineering Environment.* Mountain View CA: Intellicorp.

Jackson, P. (1990). *Introduction to Expert Systems.* Second Edition. Addison-Wesley.

Johns, N. (1991). *MacProlog 4.0 Reference Manual.* Logic Programming Associates Ltd, Studio 4, Royal Victoria Patriotic Building, London.

Johnson-Laird, P. (1983). *Mental Models.* Cambridge University Press.

Kahn, G., Nowlan, S. & McDermott, J. (1985). *MORE: an intelligent knowledge acquisition tool.* Proceedings of the 9th Joint Conference on Artificial Intelligence. Los Angeles, CA, August. pp581-584.

Karel, G. & Kenner, M. (1989). *Klue: A diagnostic expert system tool for manufacturing.* Intellinews 5(1).

Kingston, J. (1992). *User Interfaces for Knowledge Based System Tools.* AIAI-TR-101, Artificial Intelligence Applications Institute, University of Edinburgh.

Kingston, J. (1995). *Linking Knowledge Acquisition with CommonKADS Knowledge Representation.* AIAI TR-156, Artificial Intelligence Applications Institute, University of Edinburgh.

Kingston, J., Doheny, J. & Filby, I. (1995). *Evaluation of workbenches which support the CommonKADS methodology.* The Knowledge Engineering Review, Vol 10:3, 269-300.

Klausmeier, H., Ghatala, E. & Frayer, D. (1974). *Conceptual Learning and Development: a Cognitive View.* New York: Academic Press.

Kolodner, J. (1995). *From case-based reasoning to scaffolded electronic notebooks: a journey.* In: Greer, J. (Ed). *Proceedings of AI-ED 95: World Conference on Artificial Intelligence in Education,* Washington DC. pp25-35.

Kowalski, R. (1984). *Logic as a Computer Language for Children.* In: Yazdani, M. (Ed) *New Horizons in Educational Computing.* Chichester: Ellis-Horwood.

Kowalski, R. (1985). *The relation between logic programming and logic specification.* In: Hoare, C. & Shepherdson, J. (Eds). *Mathematical Logic and Programming Languages.* New Jersey: Prentice Hall. pp11-27.

Lave, J. & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation.* Cambridge University Press.

Lochead, J. & Clement, J. (1979) (Eds). *Cognitive Process Instruction: Research on Teaching Thinking Skills.* Philadelphia: Franklin Institute.

Lutz, R. (1995). *Deep Knowledge for Intelligent Program Debugging.* In: Vanneste, P., Bertels, K., & De Decker, B. (Eds) *Proceedings of the Workshop on Automated (Novice) Program Analysis.* AIED 95: World Conference on Artificial Intelligence in Education, Washinton DC. pp4-10.

Major, N. & Reichgelt, H. (1990). *ALTO: An Automated Laddering Tool.* In: Wielinga, B. et al (Eds): *Current Trends in Knowledge Acquisition.* pp 222-236. Amsterdam: IOS Press.

Malone, T., & Lepper, M. (1987). *Making Learning Fun.* In Snow, R. & Farr, M. (Eds). *Aptitude, Learning and Instruction: Cognitive and Affective Process Analyses.* Hillsdale, N.J.: Lawrence Erlbaum.

Marcus, S. (1987). *Taking backtracking with a grain of SALT.* In Boose, J. & Gaines, B. (Eds). *Knowledge-based systems: Knowledge acquisition tools for expert systems.* Vol 2 211-226. New York: Academic Press.

Marcus, S. (1988). *SALT: a knowledge acquisition tool for purpose-and-revise systems.* In: Marcus, S. (Ed). *Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic Press.

Marques, D., Latto, A., & McDermott, J. (1988). *A comparison of case-based reasoning with few large vs many small features.* In: *Proceedings of the Workshop on Case-based Reasoning,* St Paul.

McArthur, D., Klahr, P., & Narain, S. (1986). *ROSS: An object-oriented language for constructing simulations.* In: Klahr, P.

& Waterman, S. (Eds) *Expert Systems Techniques, Tools, and Applications.* Reading, MA: Addison-Wesley. pp70-94.

McCabe, F. (1992). *Logic and Objects.* Hemel Hempstead: Prentice-Hall International.

Mellar, H. (1990). *Creating alternative realities: Computers, modelling and curriculum change.* In: Noss, R., & Dowling, P. (Eds). *Mathematics versus the National Curriculum.* London: Falmer Press.

Mellar, H., Bliss, J., Boohan, R., Ogborn, J., Tompsett, C. (Eds) (1994). *Learning with Artificial Worlds: Computer Based Modelling in the Curriculum.* Falmer Press.

Michie, D. (1990). *The superarticulacy phenomenon in the context of software manufacture.* In: Partridge, D. & Wilks, Y. (Eds). *The Foundations of Artificial Intelligence.* Cambridge University Press.

Michie, D., Paterson, A. & Hayes-Michie, J. (1989). *Learning by Teaching.* In: *Proceedings of the 2nd Scandinavian Conference on Artificial Intelligence.* Tempere, Finland. pp307-330.

Miller, R.S., Brough, D.R, and Briggs, J.H. (1991). *Declarative Software Tools for Learners.* In: *Proceedings of PEG 91.* Genoa pp 177-184.

Moss, C. (1994). *Prolog++: The Power of Object-Oriented and Logic Programming.* Wokingham: Addison-Wesley.

Murray, T. (1993). *Formative Qualitative Evaluation for "Exploratory" ITS Research.* Journal of Artificial Intelligence in Education Vol. 4 No. 2/3.

Musen, M., Fagan, L., Combs, D. & Shortliffe, E. (1988). *Use of a domain model to drive an interactive knowledge-editing tool.* In Boose, J. & Gaines, B. (Eds) *Knowledge-based systems: Knowledge acquisition tools for expert systems.* Vol 2 257-274. New York: Academic Press.

Myers, C., Fox, J., Pegram, S. & Greaves, M. (1983). *Knowledge acquisition for expert systems: experience using EMYCIN for leukaemia diagnosis.* In: *Proceedings of Expert Systems '83,* 277-83.

Nichol, J., Briggs, J., Dean, J., Nichol, R., O'Connell, K., Raffan, J. & Wild, M. (1988). *PROLOG tools in education.* In Nichol, J., Briggs, J. and Dean, J. (Eds) *Prolog, Children and Students.* London: Kogan Page.

Nichols, D. (1994). *Issues in Designing Learning by Teaching Systems.* AAI/AI-ED Technical Report No 107. Computing Department, Lancaster University, Lancaster, UK.

Nielsen, J. (1993). *Usability Engineering.* Boston: Academic Press.

Norman, D. (1986). *Cognitive Engineering.* In: Norman, D. & Draper, S. (Eds). *User-centred system design.* Hillsdale, NJ: Erlbaum, pp31-61.

Nydahl, G. (1991). *Expert Systems in Education: New Opportunities or Threats to the Learning Environment.* In: *Proceedings of PEG91,* pp188-193, Genoa.

O'Keefe, R., & O'Leary, D. (1989). *Expert system verification and validation: a survey and tutorial.* Artificial Intelligence Review 7, 3-42. Kluwer Academic.

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas.* Hemel Hempstead: Harvester Wheatsheaf.

Papert, S. (1993). Introduction to Second Edition of *Mindstorms: Children, Computers and Powerful Ideas.* Hemel Hempstead: Harvester Wheatsheaf.

Papert, S. (1994). *The Children's Machine.* Hemel Hempstead: Harvester Wheatsheaf.

Paterson, A. & Niblett, T. (1982). *ACLS User Manual.* Glasgow: Intelligent Terminals Ltd.

Petre, M. (1993). *Using graphical representations requires skill, and graphical readership is an* acquired *skill.* In: Cox, R., Petre, M., Brna, P., & Lee, J. (Eds). *Proceedings of the Workshop on Graphical Representations, Reasoning and Communication.* AIED 93: World Conference on Artificial Intelligence in Education, Edinburgh.

Piaget, J. (1941). *The Child's Conception of Number.* New York: Norton.

Piaget, J. (1974). *The Grasp of Consciousness: Action and Concept in the Young Child.* Cambridge, Mass: Harvard University Press.

Polson, M. & Richardson, J. (Eds) (1988). *Foundations of Intelligent Tutoring Systems.* Hillsdale, New Jersey: Lawrence Erlbaum.

Price, C. J. (1990). *Knowledge Engineering Toolkits.* Chichester: Ellis-Horwood.

Quinlan, J. (1979). *Discovering Rules from Large Collections of Examples: A Case Study.* In Michie, D. (Ed), *Expert Systems in the Microelectronic Age*, Edinburgh University Press, pp168-201.

Reddy, Y., Fox, M. & Husain, N. (1986). *The knowledge-based simulation system.* IEEE Software, 3, 26.

Reiter, R. (1978). *On Closed World Data Bases.* In: Gallaire, H. & Minker, J. (Eds). *Logic and Data Bases.* New York:Plenum Press. pp55-76.

Repman, J. (1993). *Collaborative Computer-Based Learning: Cognitive and Affective Outcomes.* Journal of Educational Computing Research 9(2), 149-163.

Resnick, L. (Ed) (1989). *Knowing, Learning and Instructing: Essays in honour of Robert Glaser.* Lawrence Erlbaum.

Richmond, B., Peterson, S., & Vescuso, P. (1987). *STELLA.* High Performance Systems, New Hampshire USA.

Robertson, D. (1992). *A Toolkit for Assessing Resource Management Options.* Department of Artificial Intelligence, University of Edinburgh (unpublished).

Robertson, D., Bundy, A., Uschold, M. and Muetelfeld, R. (1989). *The ECO program construction system: ways of increasing its representational power and their effects on the user interface.* International Journal of Man-Machine Studies, Vol. 31 pp 1-26.

Rosch, E. (1975). *Cognitive Representations of Semantic Categories.* Journal of Experimental Psychology: General, 104, 192-233.

Rugg, G. & McGeorge, P. (1995). *Laddering.* Expert Systems, Vol 12 No 4. pp339-346.

Salomon, G. (1990). *Cognitive effects with and of computer technology.* Communications Research Vol 17 (1) 26-44.

Salomon, G. (1993). *On the Nature of Pedagogic Computer Tools: The Case of the Writing Partner.* In: Lajoie, S. & Derry, S. (Eds) *Computers as Cognitive Tools.* Hillsdale NJ: Erlbaum.

Salomon, G. , Perkins, D. & Globerson, T. (1991). *Partners in Cognition: Extending Human Intelligence with Intelligent Technologies.* Educational Researcher, April, pp2-9.

Schriber, T. (1974). *Simulation using GPSS.* New York: Wiley.

Schweickert, R., Burton, A., Taylor, N., Corlett, E., Shadbolt, N., & Hedgecock, A. (1987). *Comparing Knowledge Elicitation Techniques: a Case Study.* Artificial Intelligence Review Vol 1 No 4 254-254.

Scottish Office Education Department, (1993). *Curriculum and Assessment in Scotland National Guidelines: Environmental Studies 5-14.* Scottish Office Education Department.

Self, J. (1990). *Bypassing the Intractable Problem of Student Modelling.* AAI/AI-ED Technical Report No. 41, Computing Department, Lancaster University.

Shaw, M. & Gaines, B. (1987). *Techniques for knowledge acquisition and transfer.* International Journal of Man-Machine Studies, 27, 251-280.

Shneidermann, B. (1987). *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Reading, Massachusetts: Addison-Wesley.

Simons, G. (1985). *Expert Systems and Micros.* Manchester: NCC Publications.

Sleeman, D. & Brown, J.S. (1982). *Intelligent Tutoring Systems.* London: Academic Press.

Snow, R. & Yallow, E. (1982). *Education and Intelligence.* In Sternberg, R. (Ed), *Handbook of Human Intelligence.* Cambridge: Cambridge University Press.

Sommerville, I. (1982). *Software Engineering.* London:Addison-Wesley.

Stelzner, M. & Williams, M. (1988). *The Evolution of Interface Requirements for Expert Systems.* In: Hendler, J. (Ed) *Expert Systems: The User Interface.* New Jersey: Ablex. pp285-306.

Stenhouse, L. (1975). *An Introduction to Curriculum Research and Development.* London: Heinmann.

Stenning, K., McKendree, J., Conlon, T., Lee, J., Cox, R., & Oberlander, J. (1996). *Integrating Social and representational aspects of problem solving across the high school curriculum.* Proposal to the Cognitive Studies for Educational Practice (CSEP) Program, McDonnell Foundation. University of Edinburgh.

Swan, K. & Black, J. (1993). *Knowledge-based Instruction: teaching problem-solving in a Logo Learning Environment.* Interactive Learning Environments Vol 3 Issue (1), 17-53.

Szolovits, P. (1991). *Knowledge-based Systems.* In: Mayer, A., Guttag, J., Rivest, R., Szolovits, P. (Eds). *Research Directions in Computer Science: an MIT Perspective.* MIT Press.

Tansley, D., & Hayball, C. (1993). *Knowledge-Based Systems Analysis and Design: A KADS Developer's Handbook.* Hemel Hempstead:Prentice-Hall.

Taylor, J. & du Boulay, B. (1987). *Studying Novice Programmers: Why They May Find Learning Prolog Hard.* In: Rutkowska, J. & Crook, C. (Eds) *Computers, Cognition and Development.* John Wiley.

Touretzky, D. (1986). *The Mathematics of Inheritance Systems.* Morgan-Kaufmann.

Twidale, M. (1993). *Redressing the Balance: The advantages of informal evaluation techniques for Intelligent Learning Environments.* Journal of Artificial Intelligence in Education Vol. 4 No. 2/3.

Underwood, J., & Underwood, G. (1990). *Computers and Learning.* Oxford: Basil Blackwell.

Valley, K. (1990). *The Use of Expert Systems in Education: an Explanation-Based Approach.* Ph.D Thesis, Department of Artificial Intelligence, University of Edinburgh.

Valley, K. (1992). *Explanation, Exploration and Learning: The Use of Expert System Shells in Education.* Journal of Artificial Intelligence in Education 3(3), 255-273.

van Melle, W. (1979). *A Domain-Independent Production-Rule System for Consultation Programs.* Proceedings IJCAI-79, pp. 923-925.

VanLehn, K. (1988). *Student Modelling.* In: Polson, M. & Richardson, J. (Eds) *Foundations of Intelligent Tutoring Systems.* Hillsdale, New Jersey: Lawrence Erlbaum. pp55-78.

Vanneste, P., Bertels, K., & De Decker, B. (1993). *The Use of Semantic Augmentation within a Student Program Analyser.* Proceedings of PEG93, Moray House Institute of Education, Edinburgh pp250-260.

Vanneste, P., Bertels, K., & De Decker, B. (1995). *Reverse Engineering and Novice Program Analysis.* In: Vanneste, P., Bertels, K., & De Decker, B. (Eds) *Proceedings of the Workshop on Automated (Novice) Program Analysis.* AIED 95: World Conference on Artificial Intelligence in Education, Washinton DC. pp11-16.

Vargas, J. & Raj, S. (1993). *Developing maintainable expert systems using case-based reasoning.* Expert Systems, November, Vol 10 No 4, pp219-225.

Vasey, P., Spenser, C., Westwood, D., Westwood, A. (1990). *Prolog++ Programming Reference Manual.* Logic Programming Associates Ltd, Studio 4, Royal Victoria Patriotic Building, London.

Vygotsky, L. (1978). *Mind in Society: the Development of Higher Psychological Processes.* Cambridge, Mass: Harvard University Press.

Ward, R. & Sleeman, D. (1987). *Learning to use the S1 knowledge engineering tool.* Knowledge Engineering Review, 2(4), 265-76.

Webb, M. (1992). *Learning by Building Rule-based Models.* Computers in Education Vol 18 No 1-3 pp. 89-100.

Webb, M. (1993). *Children building models: the design and evaluation of a computer-based modelling environment for schools.* PhD thesis (1st full draft), Open University.

Webb, M. (1994). *Learning by Building Expert System Models.* In Mellar, H., Bliss, J., Boohan, R., Ogborn, J., Tompsett, C. (Eds) *Learning with Artificial Worlds: Computer Based Modelling in the Curriculum.* Falmer Press.

Wedekind, J. (1993). *Computer-aided Model Building.* In: Lajoie, S. & Derry, S. (Eds) *Computers as Cognitive Tools.* Hillsdale NJ: Erlbaum.

Wideman, H. & Owston, R. (1988). *Student Development of an Expert System: a Case Study.* Journal of Computer-Based Instruction, 15:3, pp 88-94.

Wideman, H. & Owston, R. (1993). *Knowledge Base Construction as a Pedagogical Activity.* Journal of Educational Computing Research, Vol 9(2) 165-196.

Wielinga, B., Van de Velde, W., Schreiber, G. & Akkermans, H. (1992). *The CommonKADS framework for knowledge modelling.* Technical Report KADS-II/T1.1/PP/UvA/35/1.0, University of Amsterdam.

Winograd, T. & Flores, F. (1986). *Understanding computers and cognition: A new foundation for design.* Norwood, NJ: Ablex.