Developing An Authoring Environment For Procedural Task Tutoring Systems

A dissertation presented in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science at Massey University, Palmerston North, New Zealand.

Shamus Paul Smith

1997

Abstract

The use of computers in education is becoming more and more common as the price of technology drops and its general availability is increased. Unfortunately, building computer based tutoring systems is a difficult process which is fraught with many problems. A significant problem in this area is the lack of reuse of system components between computer tutor developments. This means that each new system must be started from scratch and mistakes from earlier projects can easily be repeated. A complementary difficulty is the variety of specialist skills that are required to build these systems. Typical developers do not usually possess the combination of domain, cognitive science and programming knowledge that is needed to build computer tutors. One solution to these problems is the use of an authoring environment for facilitating the building of computer based tutoring systems.

This thesis presents an authoring tool for the construction of computer based tutoring systems teaching procedural tasks in a discovery learning environment. TANDEM (Task ANd Domain Environment Model) provides tools for domain and task definition, sub-domain definition and a domain independent tutoring engine.

It is argued that such an environment can provide a non-expert user with access to advanced techniques from artificial intelligence research for knowledge acquisition and representation. Several tasks from the construction process have been automated, thus simplifying this activity. The use of sub-domain partitioning has been considered and techniques for the integration of custom built domain interfaces are described. Also, it is proposed that by providing a domain independent tutoring engine, reuse can be encouraged over numerous domains which can reduce the development time required to build these systems.

Acknowledgements

Firstly, I would like to thank my main supervisor, Ray Kemp, for his help, support and encouragement over the course of this research. He provided the motivation for this topic, co-authored several publications and proof-read multiple versions of this thesis.

Also, I have appreciated the support provided by my two second supervisors, Daniela Mehandjiska-Stavreva and Mark Apperley, especially Daniela's observations on the first draft of this thesis.

Thanks is also necessary to the FIMS Computer Support Staff for keeping all the equipment working and their flexibility and consideration, i.e. network disk space, during the write-up period of this thesis.

I am also grateful for the financial assistance provided by Massey University in the form of a MURF (Massey University Research Fund) grant, a Massey University Doctoral Scholarship and a graduate assistant position in the department of Computer Science.

Four years is a long time to keep focused and motivated on one topic and the emotional pressures of post-graduate research can be intense. I have been fortunate to have been able to share this time with four years of Honours, Masterate and Doctoral students and Computer Support Staff from the Computer Science, Information Systems and Mathematics departments. This has made life bearable with all its ups and downs.

Finally, I wish to thank my parents for their continuing encouragement and optimism.

Publications

The following publications are associated with the research presented in this thesis.

Journal Article

Kemp, R. H. and Smith, S. P. (1994). Domain and task representation for tutorial process models. *International Journal of Human Computer Studies*, 41(3), 363-383.

Conference Proceedings

Kemp, R. H. and Smith, S. P. (1994). Teaching procedural skills by computer simulation. In *Second Singapore International Conference on Intelligent Systems (SPICIS '94)*, (pp. B73-B78), Singapore.

Kemp, R. H. and Smith, S. P. (1994). Using Planning Techniques to Provide Feedback in Interactive Learning Environments. In P. T. Metaxas (Ed.), *Sixth International Conference on Tools with Artificial Intelligence*, (pp. 700-703). New Orleans, USA: IEEE Computer Society Press.

Smith, S. P. and Kemp, R. H. (1995). Development of a Discovery Learning Tutoring System Construction Environment. In J. Greer (Ed.), AI-ED 95 : *World Conference on Artificial Intelligence in Education*, (pp. 595). Washington DC, USA: AACE : Association for the Advancement of Computing in Education.

Smith, S. P. and Kemp, R. H. (1995). Efficient Modelling of Domains for Computer Tutoring Systems. In R. Kotagiri (Ed.), *ACSC 95 : Proceedings of the 18th Australasian Computer Science Conference*, Vol. 17, Number 1 (pp. 491-498). Adelaide, South Australia: Australian Computer Science Communications.

Kemp, R. H. and Smith, S. P. (1996). A Visual Approach to Procedural Tutor Specification. In J. Grundy and M. Apperley (Eds.), *Sixth Australian Conference on Computer-Human Interaction*, (pp. 190-196). Hamilton, New Zealand: IEEE Computer Society Press.

Internal Report

Kemp, R. H. and Smith, S. P. (1994). Facilitating feedback in discovery learning systems. *Information and Mathematical Sciences Report No. 94/19*. Massey University.

Table of Contents

Figures

Chapter 1

Introduction

1.1 Background

Since the first attempts at building Computer Assisted Instruction (CAI) systems to the more recent Intelligent Tutoring Systems (ITSs), the availability and/or cost of the technology needed to realise these systems has been a major constraint on their design and construction (Hall, Thorogood, Sprunt, Carr and Hutchings, 1990; Merrill, Li and Jones, 1990; Syllabus, 1992).

The present availability of computer technology has reached a point where this is no longer as critical a development constraint. So, where are all the intelligent tutoring systems? There are a number of factors that are impeding the progress of constructing custom designed ITSs. The time consuming nature of computer based tutor development, lack of specified production standards, scarcity of reusable components, limited use of sharable knowledge bases and non-general tutoring engines have all contributed to the difficulties of building ITSs.

In addition, new ITS developments are typically started from scratch, without a foundation on which they should be built. This has led to some undesirable consequences in their construction, namely, high development costs, the lack of reuse of existing general-purpose software,

prohibitive delivery platform requirements, little standardisation between developed systems, difficulty in system evaluation and high maintenance costs.

High development costs are encountered as researchers in ITSs are forced to design their own system architectures, implement all system components, develop their own knowledge representation techniques and reasoning mechanisms and encode all the domain and instructional knowledge in order to conduct research. Typically, some research idea is investigated and much of the rest of the system is just a skeleton infrastructure of components used only to support the research.

Current instructional design and development practices are extremely labour intensive. Even though the hardware is affordable, the courseware frequently is not (Merrill, *et al.*, 1990). Merrill, *et al.* estimate that the design and development time for ITSs is approximately 300-500 hours per hour of instruction. This view is also shared by Beverley Woolf (Syllabus, 1992) and Orey, Trent and Young (1993).

The time-consuming nature of ITS development implies a massive commitment. Much of this time commitment is spent in the elicitation of domain knowledge. This knowledge is needed for the development of the various models that are needed by ITSs, especially the domain model.

Similar knowledge elicitation problems have been noted in the areas of knowledge-based systems and the development of user models. One of the most troublesome and time-consuming activities in constructing a knowledge-based system is the elicitation and modelling of expert knowledge about the problem domain (Garg-Janardan and Salvendy, 1990; Kitto and Boose, 1989). This view is also held by Cerri, Cheli and McIntyre (1992) in regard to the development of user models. This activity is considered both time-consuming and labour-intensive and represents a significant part of the cost and difficulty of developing a system.

Reuse of existing software is not common in ITS developments. General-purpose software is available, both in the commercial area (eg. text editors, spreadsheets, drawing tools and databases) and in research laboratories (eg. planners, natural language processing (NLP) tools and parsers). Unfortunately, existing general software and tools are not easily

incorporated into ITSs. This can lead to an increase in the total development time of a system. However, Bordier, Paquette and Carrier (1990) observe that using general-purpose software makes prototyping faster and can allow interative development, with end-user participation, of systems possible. Bordier, Paquette and Carriers' systems are built using general-purpose software such as Microsoft Excel (Microsoft, 1997), SuperCalc (Computer Associates, 1996) and HyperCard (Apple, 1997).

Platform demands can be prohibitive as ITSs are typically implemented as large programs that require significant resources. Also, ITSs may need special hardware or may be implemented in a language that may restrict the delivery platform. Utilisation of tools allowing the development of hypertext, hypermedia, multimedia and CD-ROM applications, the current 'buzzwords' of technology, can result in what has been called *techno-romanticism* (Beveridge, 1989). These applications may pose undesirable platform demands without adding any significant benefits. This technocentric approach to building systems leads to an overestimation of the capabilities of the new technology (Hutching, Hall, Briggs, Hammond, Kibby, McKnight and Riley, 1992).

Little standardisation between systems exists. 'Ad hoc' development of ITSs encourages the utilisation of new knowledge representation techniques, new media, new advances in architectures and control flow schemes. Little reuse of software components across developments is common, especially with the sharing of teaching strategies and instructional material. Design methodologies for ITS construction have been defined over the years, for examples see Anderson, Boyle, Farrel and Reiser (1987), Clancey (1987b), Bordier *et al.* (1990). However, many are not easily applied when building systems.

One result of the lack of standardisation is the complexity of system evaluation and the difficulty of comparing developed systems. Therefore, the process of determining the strengths and weaknesses of implemented systems is non-trivial (Or-Bach and Bar-On, 1993). This makes internal and external evaluation of ITSs very difficult.

Maintenance costs for ITSs can arise as many systems are developed as laboratory prototypes with little attention being paid to maintenance. Thus, system modifications and new functional requirements can force

large portions of a system to be removed or be re-implemented from scratch. For domain models, student models, teaching strategies and interface development there are no set standards for building ITSs. As a result, they are commonly domain dependent. This can make systems difficult, if not impossible, to modify for a new domain or when adding new features to an existing domain.

All these factors have contributed to make building ITSs difficult. Another problem involves the skills needed and personnel required to build such systems. Firstly, if a development team is building an ITS, either the domain experts must become integral members of the development team (which is not effective use of expert time) or the developer must become an expert in the domain (which is not possible or desirable in many domains).

Secondly, a teacher may wish to construct his/her own custom tutoring system within some domain (Kearsley, Hunter and Furlong, 1992). Again, a majority of domain experts would not have the required skills for this development process. Creating a computer based tutor requires not only knowledge of the domain but also knowledge of cognitive science and of programming (Blessing, 1995). There are three main knowledge areas that are needed for ITS development :

(i) Programming knowledge :
to encode knowledge representation.
to implement the tutoring system and its various models.
to implement user interfaces.

(ii) Cognitive science knowledge :
to define knowledge representation techniques.
to specify teaching strategies.
to develop student models and their usage.

(iii) Domain knowledge :
to define domain and task models.

Unfortunately, domain experts do not necessarily have a programming or cognitive science background, programmers do not necessarily know

cognitive science or have domain knowledge and cognitive scientists do not necessarily have domain knowledge or programming experience.

Another problem related to the high maintenance cost of ITSs is the issue of domain dependence. Most ITSs can be grouped into one of two categories : *domain dependent* and *domain independent*.

Domain dependent systems combine the domain knowledge, teaching strategies and interface code. This makes it difficult to re-use, modify and maintain such systems. Domain dependent systems are useful for small, well defined domains and/or prototype systems where the domain knowledge is the most important component.

Domain independent systems keep some components of the system independent from the domain knowledge. This allows the system to be used for a variety of domains. However, many domain independent systems are still restricted to certain environments, for example simulation, mathematics or language learning domains.

Figure 1.1 shows the problems identified and the relations between these problems.

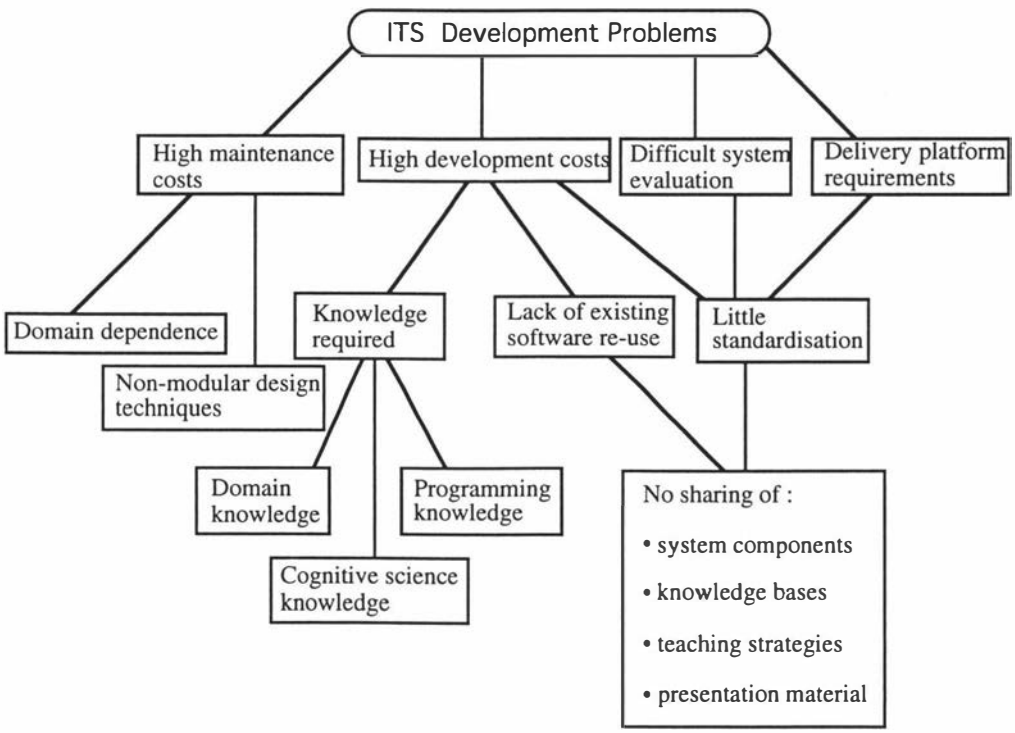


Figure 1.1 : ITS development problems.

One possible solution to these problems is to provide tools and methodologies that allow the development of ITSs in a way that reduces the high development cost overhead and curbs the associated high maintenance costs.

Authoring tools and ITS shells have been developed to aid the development of ITSs. Examples from the recent literature can be seen in Bessiere and Vacherand-Revel (1990), Bell and Jackson (1993), Munro, Johnson, Surmon and Wogulis (1993), Blessing (1995) and Major, Murray and Bloom (1995). Unfortunately, as Bell and Jackson observe, such systems require a detailed level of understanding of the knowledge representation techniques and processes involved or require the developer to program using textual code.

The use of authoring tools and other aids helping the construction of components of an ITS is not enough. The success of an ITS development requires the combination of domain knowledge elicitation, tutoring system construction and user interface development activities. Building appropriate domain models and teaching strategies is only part of the solution. What is needed is a development environment that facilitates the construction of ITSs while reducing the effects of the problems identified.

The goal of the research initiated here is to provide a prototype system for a construction environment for a limited range of ITSs : those involving the teaching of procedural tasks using guided discovery learning.

1.2 Context of the Research

1.2.1 Motivation

The main motivation for the research undertaken has been the perceived need for a general tutoring system construction environment. The specific purpose of the research project has been to investigate whether it is possible to provide such a tool and, if so, to produce a piece of software that demonstrates its feasibility. General systems allow component reuse which is a large step towards reducing the development time and cost of computer tutoring systems. Several problems are associated with reuse,

namely domain and system platform considerations and the management of domain dependent components, eg. domain and task descriptions and user interfaces. Also, the separation of system elements from the domain context can improve their potential for reuse.

1.2.2 Thesis

A development environment for tutoring system construction should allow developers to build an ITS in a way that reduces the usual development costs of ITS development while basing itself around a sound teaching methodology.

An ideal construction environment would :

- ease the model (domain, task or student) definition process.
- provide alternative teaching strategies within the environment.
- not restrict the type of interface that is required and would allow easy integration of interface links to the environment.
- automate as much as possible the construction and validation procedures to remove time consuming and difficult tasks from the system developer.

The implementation of a system to make progress towards these ideals has been the goal of the current work. The tangible product of the research presented is a package running on Macintosh computers called TANDEM (Task ANd Domain Environment Model) (Kemp and Smith, 1996). There are two main components to this system, namely, a domain knowledge representation tool and a general tutoring engine. The software is written in the high level programming language Prolog and contains 6000 lines of source code. Features of the TANDEM environment are described and examples of its use are demonstrated.

1.3 Thesis Outline

In this thesis, the research presented has been divided into discrete sections. Although there is a large amount of overlap between many of these sections, this has been partially ignored, for the purpose of achieving a coherent description of the research.

Chapter 2 presents an overview of CAI and ITS which are the general areas for this research. The components of a typical ITS model are described, with reference to relevant systems from the current literature where appropriate.

The focus of the research is in the area of ITS authoring tools. The issues concerning ITS authoring tools, their implementation and the use of restrictions to limit their scope are discussed in Chapter 3.

Chapter 4 is the first of two theory chapters which provide a theoretical foundation for the implementational work in the current research. Firstly, knowledge representation and the development of domain and task models will be examined. Secondly, in Chapter 5, the use of feedback and how appropriate feedback can be provided by utilising teaching strategies is examined.

Although the theory presented has been implemented, the current prototypes have been developed as functional, rather than polished. Nevertheless, the system has been developed to a stage that allows the verification of the complete construction process. Chapter 6 describes TANDEM, the practical component of this research. The features considered beneficial from the knowledge representation techniques and feedback considerations are discussed in terms of their integration into the TANDEM environment.

Chapter 7 presents some examples of domains defined within the TANDEM system and their use in an educational environment.

Finally, Chapter 8 presents the conclusions of the current research. Areas for future work that were identified during the project will be described.

Chapter 2

Intelligent Tutoring Systems

2.1 Introduction

The use of computers as an educational tool was considered early in the development of computers. As computers have become more powerful and accessible, more interest and research has been focused in using them in an educational environment. The development of the artificial teacher has been one of the ideals in this area. A total automation of the teaching process would provide many advantages including the benefits of one-to-one teaching, reuse and standardisation of resource material and a more personalised learning environment. Unfortunately, the complexities of learning and the teaching process have left the idea of the artificial teacher in the realms of science fiction.

However, much work has been carried out in the research of teaching and how computers can best be incorporated into its structure. This chapter is an overview of the use of computers in education from initial computer assisted instruction to the current development of Intelligent Tutoring Systems (ITSs). The use of planning techniques and knowledge representation methods and their application to ITSs are investigated. The structure of a typical ITS is examined. The utilisation of knowledge representation techniques and discovery learning environments during the development of ITSs are also addressed.

2.2 CAI and Education

The use of Computer Assisted Instruction (CAI) systems in an educational environment provides several advantages over more traditional approaches to teaching. CAI can provide high quality individual training where training effectiveness can be increased on a large scale as strategies used by the best instructors can be replicated in a CAI lesson (Suppes, 1980). Educational psychologists have observed that private tutoring is an advantage when teaching with many different types of material (Anderson and Reiser, 1985; Bloom, 1984). The student can get immediate feedback as the grading is automatic. In addition, with computer based training, the instructional time is driven by the individual students' own capabilities and motivations in comparison to a classic classroom situation where instruction time for the whole group is typically driven by the slowest learners.

For some students, a computer environment is less threatening than a traditional classroom environment. Students can succeed or fail in private and this encourages experimentation (Bork, 1980). Anderson and Reiser (1985) raise the point that a student should be able to work on a problem in a 'friendly' environment. The impersonal aspect of computers can help children, who are nervous about exposing their mistakes to teachers, to engage in problem-solving encounters. Therefore, they can learn from their mistakes without fear of being thought ignorant (Beveridge, 1989; Reinhardt and Schewe, 1995). Also, Eberts and Brock (1988) observe that students like using computers because of their novelty value, although for some systems, the actual learning increases can be small.

Computer environments provide hands-on instruction and in many cases, computer machinery can be less costly than expensive or dangerous specialised equipment. Many existing systems have been developed due to the desire to provide a teaching simulation without the cost and danger of using real systems; for examples see Hollan, Hutchins and Weitzman (1984), Newman, Grignetti, Gross and Massey (1992) and Woolf (1992). Also high quality training becomes available at remote sites and instructions can be sent to students so instructors do not need to travel and equipment does not need to be transported or purchased.

Unfortunately, CAI environments are not without their disadvantages. As mentioned in Chapter 1, the length of time required to author one hour of instruction is too long. This is a major problem for the production of instructional material and consequently for its wide utilisation. Also, the human-computer interface is inflexible in comparison to the human teaching environment. The modes of communication between the student and the computer are limited. The responses from students must be within a range anticipated by the CAI author. Unexpected or unique student questions often receive inadequate feedback. This is especially the case in ill-defined problem domains. Eberts and Brock (1988) note that CAI systems can be less applicable in areas where the knowledge to be learned is less explicit.

2.3 CAI and ITS

Traditional approaches to CAI involved three broad areas of use (Carbonell, 1970, Bryan, 1969). The first area identified is a CAI system where the student is given full control to explore some computer simulated environment. The computer is used to present information and the student has total control of the system and the interaction with it. This approach is similar to the card-based navigation environment as provided by tools like HyperCard.

The second area identified is the use of games and simulations which constrain the students learning to a particular area of interest. As long as the rules of the simulation/game are adhered to, the student is free to explore. Unfortunately, the operations that the user can perform are limited and have to be anticipated at some level by the computer. However, users can get a sense of being involved in a realistic situation and of making decisions that have plausible consequences (Kemp, 1992).

The third area identified is the use of controlled learning. Drill-and-practice systems present questions with anticipated correct and incorrect answers. Sequencing through these systems is traditionally deterministic.

CAI systems can be classified as *generative* or *adaptive*. Generative CAI systems were designed for situations requiring repetitive practice of a skill. The second type of systems are termed adaptive CAI systems. They make

use of information obtained from the student to guide the selection of problems or the course of the tutoring sessions (Kass, 1989).

The major problem with both these approaches is the lack support of student initiative and the inflexibility of the systems responses. Tutors developing such systems face difficulties when modifying these responses as this information is embedded into the system. Consequently, modification and initial development is quite difficult and time consuming, if possible at all.

The lack of deep knowledge and reasoning capabilities are the most obvious shortcomings of the conventional CAI systems. Intelligent Computer Aided Instruction Systems (ICAISs) or Intelligent Tutoring Systems (ITSs) attempt to overcome these deficiencies by use of Artificial Intelligence (AI) techniques (Carbonell, 1970; Clancey, 1987b; Specht, 1989; Yazdani, 1987). Peachey and McCalla (1986) suggest that the goals of CAI are ultimately achievable only through the application of AI techniques.

The use of AI techniques in the development of ITSs has been widespread. However, three AI techniques seem to be prominent in the literature in regard to the construction to educational systems, namely planning, natural language processing and knowledge representation techniques. It is noted by Specht (1989) that AI techniques can be used to improve the diagnosis of students' mistakes, the detection of students' incorrect concepts and the structuring and planning of teaching strategies. Examples of NLP use can be seen in SCHOLAR's case grammar system (Carbonell, 1970) and the performance/semantic grammars in the SOPHIE systems (Brown, Burton and Kleer, 1982).

The development of knowledge representation techniques becomes important for ITSs as many systems require the use of student and domain models. These models need some underlying representation that is efficient, in terms of storage and manipulation, and appropriate for the system that is to be developed (Smith and Kemp, 1995). Typical examples include BUGGY (Brown and Burton, 1978; Burton, 1982), the LISP tutor (Anderson and Reiser, 1985), GUIDON (Clancey, 1987b) and the VCR tutor (Mark and Greer, 1995). While the use of NLP techniques is outside the scope of this thesis, the use of planning techniques and knowledge

representation methods and their application to ITSs are central to this work and will be covered in more detail in later sections.

In most ITSs, four components are considered common, namely, a representation of the subject matter (the domain knowledge), the instructional strategy used to teach the subject matter (the tutoring engine), a representation of the current state of the student's progress through the subject matter (the student model) and an interface for user/system interaction (the system interface). Figure 2.1 shows the common ITS model.

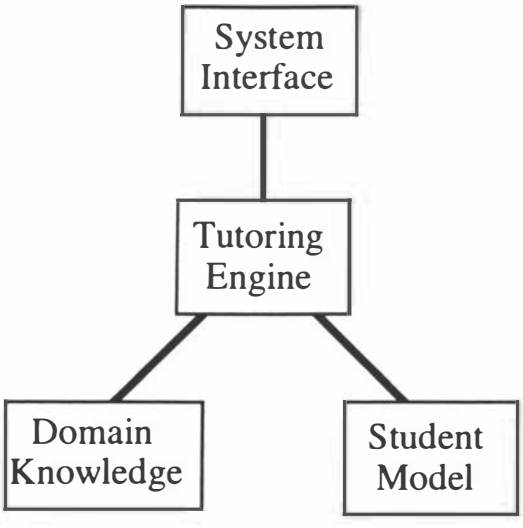


Figure 2.1 : Common ITS model.

Variations on this model can be seen in many publications; for examples see Anderson and Reiser (1985), Direne (1993), Fairweather, Gibbons, Rogers, Waki and O'Neal (1992), Kaplan and Rock (1995), Kemp (1995), Liddle, Brown, Slater and MacDonnchadha (1995), Murray (1989) and Reinhardt and Schewe (1995). However, at a core level, most have the same common architecture.

Although not all systems contain all the components in Figure 2.1, this model provides a basis of system functionality for researchers to focus on. Kemp (1995) observes that many systems that have been developed have concentrated on one or two of these components of an ITS, allowing the rest of the system to be left as skeleton infrastructure. This is due to the high cost of ITS development. Unfortunately, this means that there are many partially completed ITS that are only useful for research purposes.

Thus instead of trying to describe the perfect ITS based on the model described in Figure 2.1, each component will be described in detail with references to the relevant systems from the ITS literature.

2.3.1 Domain Knowledge

The domain knowledge is the domain dependent information that is used during a tutorial session. If the knowledge can be separated out from the rest of the system, then components of an ITS can be reused in a variety of different domains. As noted in Chapter 1, the reuse of components can significantly affect the cost and development time for a system.

Many attempts to define appropriate knowledge representation techniques have been done in the development of ITSs over the years. Techniques have been developed which allow domain knowledge to be effectively stored and accessed. SCHOLAR's semantic networks (Carbonell, 1970) (Figure 2.2), WHY's scripts (Stevens, Collins and Goldin, 1982) and SOPHIE's procedural specialists (Brown, *et al.*, 1982) are examples of knowledge structures that have been used for representing expertise within domain models and have provided a basis for knowledge representation in other projects (Kemp, 1995).

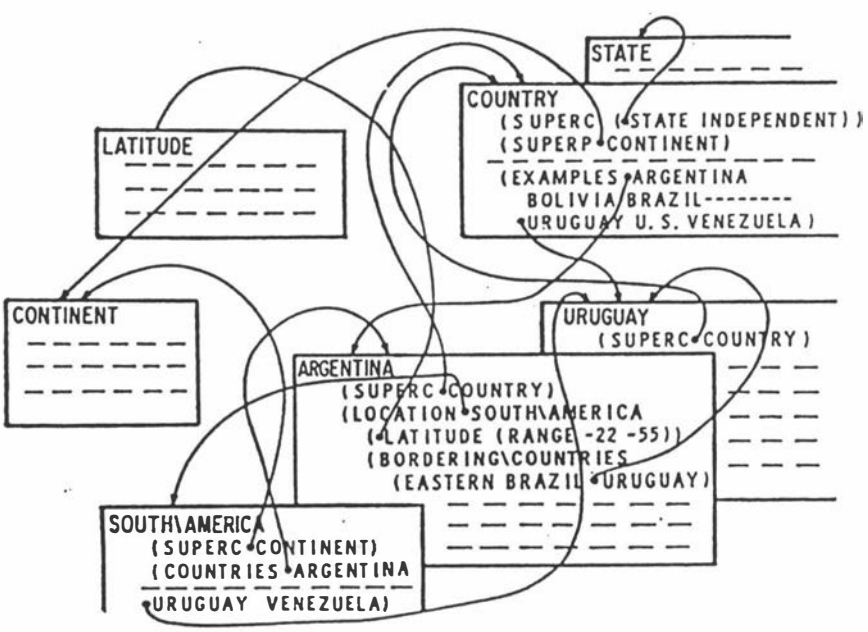


Figure 2.2 : Semantic network example, a knowledge representation method; from (Carbonell, 1970).

The definition and use of domain knowledge is central to the research presented in this thesis and will be discussed in detail in section 2.4 and Chapter 4.

2.3.2 Student Model

The student model in an ideal ITS is an accurate view of the knowledge of a student using the system. The student model contains the system's belief about the student's knowledge. It does not interact with the student, it represents him/her (Ragnemalm, 1993). From this model, the system can identify areas of knowledge that are incorrect or missing and can then target these areas in presenting teaching material. The student model must be able to infer the abilities of the student with respect to the domain being taught and consequently adapt to the teaching strategy preferred by the student. Unfortunately, the development, maintenance and use of an accurate student model is a non-trivial task (O'Shea and Self, 1983).

Kass (1989) identifies three issues that are relevant to research into student models and which need consideration:

(i) What information about a student must be modelled?
(ii) How is the information represented?
(ii) How is the student model built?

Typically, the information stored in a student model is the knowledge the student has learnt about the current domain. This may be stored as either, what has already been learnt or as what is left to be learnt. Other information that might be incorporated into student model includes:

- the student's preferred modes for interacting with the system.
- a rough characterisation of his/her level of ability.
- a consideration of what he/she seems to forget over time.
- an indication of what his/her goals and plans seem to be for learning the subject matter (Barr and Feigenbaum, 1982).

Three techniques for information representation have proven to be effective in the area of student modelling. These are the *overlay model*, the *differential model* and the *perturbation model* (Kass, 1989).

One of the simplest student modelling techniques used in ITSs is overlay modelling. This techniques is proposed by Carr and Goldstein (1977). It considers the student model as a subset of the domain model. The student model can be used to identify areas of deficiency when compared to the domain model. Therefore, the teaching process can concentrate on these areas until the student model and domain model are equivalent. Systems that use an overlay model can be seen in Carbonell (1970), Carr and Goldstein (1977) and Clancey (1987b).

However, overlay modelling has some serious deficiencies. The model is limited to the knowledge represented in the domain model. Thus, if a student tries an alternative but potentially valid strategy, firstly, the system may not recognise it. Secondly, it may be outside the scope of the system's knowledge. Therefore the system is incapable of even explaining why the answer is wrong. As the domain model is only a model, it is incomplete. Thus, the student model is based on incomplete knowledge which can severely restrict the explanation capabilities of the ITS system (Bertels, 1994; Ohlsson, 1987).

A modification to the overlay model is the differential model. Here, the student model is compared to a model describing what an expert would do in the same situation. The domain knowledge is split into two areas: the knowledge the student should have and the knowledge the student is not expected to have. The student can roam outside the confines of the expected model and try alternative strategies. The differential model again assumes that the student model is a subset of the domain model and consequently is also constrained by the same incompleteness limitations as the overlay model. Burton and Browns' system WEST (1982) uses a differential model for student modelling.

A third student modelling technique is perturbation modelling. Similar to the previous models, a close link is kept between the student and an expert model in the domain knowledge. In addition, this technique allows the student model to be extended beyond the range of the expert model. Thus the student and expert models are basically the same with small differences in some areas of the domain knowledge. Kass (1989) observes that a common approach to using perturbation models is to encode the expert knowledge and to augment that knowledge with likely

misconceptions the student might have. Examples of systems adopting perturbation modelling can be seen in Anderson, *et al.* (1987), Brown and Burton (1978), Brown, *et al.* (1982), Johnson and Soloway (1987) and Sleeman (1987).

The use of bug libraries is a common technique for using perturbation modelling. Holt, Dubs, Jones and Greer (1994) note three typical approaches to the development and representation of bug libraries, namely, *enumerative*, *generative* and *reconstructive* approaches. The first approach involves the enumeration of all the bugs in some domain, or in some cases all the bugs based on some targeted misconceptions. The generative approach involves generating and explaining bugs based on some cognitive model. However, by using this method many implausible bugs can be generated and adding a context sensitive monitor for checking the relevance of the bugs would be excessive. The reconstructive approach involves reconstructing errors from observed behaviour and the process of trying to identify the misconception which led to the error. Unfortunately, these perturbation models may be able to identify and classify bugs, but they do not explain why mistakes have occurred (Holt, *et al.*, 1994).

An alternative approach is to make the student model the focus of the system. Bull, Pain and Brna (1995) describe "Mr Collins", a student model for intelligent computer assisted language learning. The student model is not restricted to the student's current state but also maintains a record of his/her past and anticipated future learning history. Also, the user is an active participant in the construction and repair of the student model.

Although student models are used as the teaching basis for many systems, some researchers believe that their usefulness is limited and that complex student models may be impossible to construct. For instructional domains which are more complicated than trivial problems in simple arithmetic, the definition of the student model and the subsequent interpretation of student actions become daunting problems (Roberts, 1993).

Muhlhauser (1990) observes that more and more scientists believe that the construction of student and instructional models is largely impossible and should not be considered. Students should navigate their own way

through the information space and should be provided as much freedom and as little 'determinism' as possible. This will help to make the 'free journey' through the information space as easy as possible. Also, system developers can avoid excessive student modelling by focusing on the observable student actions and their effects on the current environment. One system that uses this method is Hill Jr. and Johnson's REACT (1994).

System developers trying to exclude the student modelling component must be aware of the effect this may have on the final environment. The absence of an overriding student model affects the personality of the computer tutor. The tutoring system must assume a more passive demeanour. Since the system cannot presume to know the student's intentions, it must include in any response the context within which to interpret the response (Roberts, 1993).

2.3.3 Tutoring Engine

The tutoring engine is the 'heart' of an ITS. At a basic level, it may be used to gather and format information from the domain knowledge component and display it to the student. However, the tutoring engine may be more complex, providing multiple teaching strategies that adapt to the current student and organise the updating and maintenance of a student model. The tutoring engine is usually an attempt to automate the human teacher's role. This is to prod, hint, ask leading questions or ask for justifications and guide problem solving with just the right amount of intervention and feedback (Kaplan and Rock, 1995).

The computer coach, WUMPUS (Goldstein, 1979; Yob, 1975), supplies help for a player that must hunt and kill the vicious Wumpus. This coaching environment leaves the initiative largely up to the student as the subject matter is not taught explicitly but is put to practical use in the course of playing the game. Four models are used within the system: the *expert* checking for non-optimal moves, the *psychologist* formulating hypothesis for domain skills known to the student, an *overlay model* used as a student model and the *tutor* which uses the student model to guide the interaction with the student.

Another coaching system, WEST, was developed by Burton and Brown; see Barr and Feigenbaum (1982) and Burton and Brown (1982). This

research focused on identifying diagnostic strategies required to determine a student's misunderstanding from their observed behaviour and the use of various tutoring strategies for directing the tutor to supply the appropriate feedback at the correct time.

Anderson and Reiser's LISP Tutor (1985) contains a tutoring module that determines from the students behaviour what they know. It decides when to interrupt the student and what feedback to give. It also decides what problem the student should attempt next and when to advance to a new topic. The LISP tutor is designed to provide as much guidance as necessary. One of its main teaching strategies is the switching between student *coding mode* and tutor *planning mode*. In the later mode, the tutor works with the student through different algorithms using examples. The student can then return to coding mode with an improved understanding of the problem.

The DOMINIE project (Spensley, Elsom-Cook, Byerley, Brooks, Federici and Scaroni, 1990) focuses on the development of a domain-independent tutoring system. The emphasise is on constructing a tool to support users learning to operate computer-based interfaces. Initially, the system starts teaching at the level of a new user. This is based on the assumption that the user knows nothing about the area which is being taught. DOMINIE uses multiple teaching strategies in order to be adequately responsive to the student's needs. A switch can be made to an alternative strategy when the system determines that the current strategy is not proving effective.

SIEEL (Wigetman, Patacchini and Evrard, 1992) has a domain-independent instructor module (an expert system teaching module) that matches student actions against hypothetical actions proposed by an air traffic control expert. The system tries to *situate* the student with respect to the execution of a correct plan. This approach is typical for many coaching ITSs where the tutoring engine diagnoses the student actions by comparing them to what an expert would do.

A less intrusive tutoring engine can be seen in computer based simulations. The use of simulation in an ITS is particularly useful when a device is to be modelled. Examples of this can be seen in Hollan, *et al.* (1984), Mark and Greer (1995) and Woolf (1987). The tutoring engine simulates the workings of some device and processes student interaction

with the device model. Students can gain instant feedback from actions performed in the simulated environment. Suggested actions can be easily demonstrated and the effects of inappropriate actions highlighted. However, the benefits that the ITS can give to the teaching process is dependent on the ease of student interaction with the system, which is dependent on the system interface.

2.3.4 System Interface

The system interface can be the pivotal component in the success of an ITS. Without an appropriate interface, the user cannot gain the full benefits of a system. With computer systems in general, the user interface plays a large factor in the success of the system (Pressman, 1997; Sommerville, 1996). If the users are unsatisfied with the interface, then the effectiveness of the system will suffer and it may be mothballed. User interfaces are also an important consideration with computer tutoring systems as the user interface should not impede the learning process. This can be the case when systems are fixed to one type of interface which can also constrain its reuse in alternative domains.

Kemp (1995) notes that the interface module is, perhaps, the most neglected part of ITSs. Many early systems relied on natural language processing techniques to make text interfaces as user-friendly as possible; for examples see Brown, *et al.* (1982), Carbonell (1970) and Yob (1975). However, there has recently been an active move in the ITS research community to more multimedia based ITSs. This is partly due to the current reduction in costs and availability of hardware to drive these resource-demanding multimedia applications.

Multimedia techniques have been used to enhance the user interface in modern ITSs. Woolf and Hall (1995) remark that audio-visual material can provide valuable aids for teaching systems. However, a system is only useful if the learner remains active and motivated. Beverley Woolf and Chris Eliot, from the Computer Science Department at the University of Massachusetts, have built a knowledge-based simulation for teaching cardiac resuscitation techniques. The Cardiac Tutor (Figure 2.3) presents a graphical view of a simulated emergency room patient. Students experiment by interacting with the simulated person applying proper procedures visualised on the screen. The tutor provides extensive

(colour) graphical displays as well as spoken advice, emergency room sounds and graphical indications of ECG (electrocardiogram) trace, blood gases and vital signs.

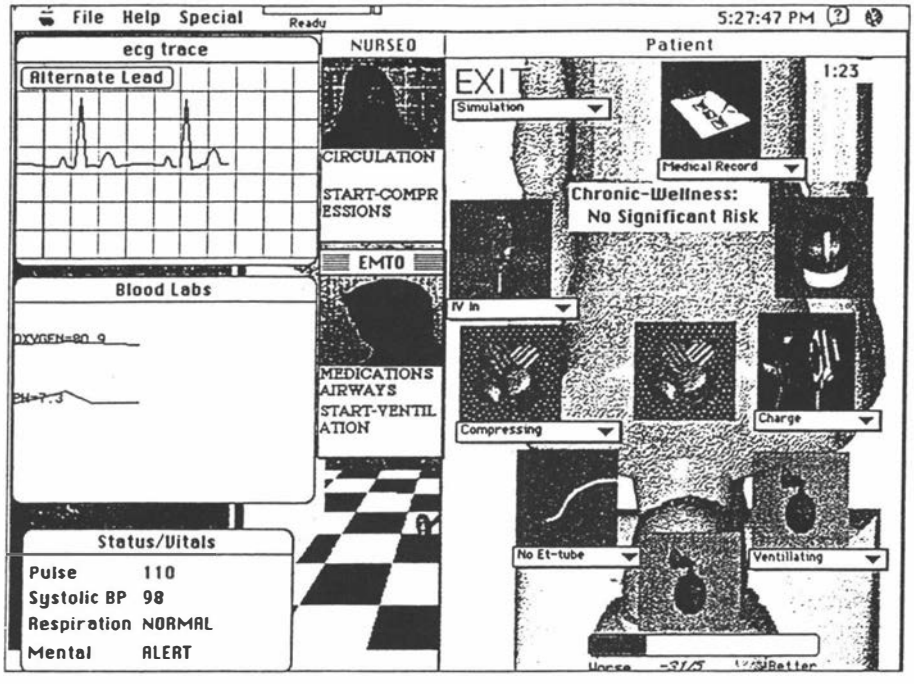


Figure 2.3 : The Cardiac Tutor; from (Woolf and Hall, 1995)

The interface component of an ITS must provide an appropriate link between the student and the tutoring engine. This issue is addressed in more detail in Chapters 3 and 6.

2.4 Knowledge Representation

The fundamental strategy behind ITSs is the separation of knowledge that is to be taught (domain expertise) from the way that it is to be taught (instructional methods) (Liddle, *et al.*, 1995). Knowledge representation and feedback mechanisms are two important features of any ITS. The first determines how domain knowledge is structured and organised. The second is the ITS's *voice* to the student. It is generated from a combination of manipulating the domain knowledge and applying an ITS's teaching methods. In this section different techniques for structuring the knowledge are considered.

The representation of knowledge is the foundation on which many ITSs are built. Boy (1996) observes that an essential aspect of knowledge is that

it is contextualised. This is the reason why knowledge is so difficult to acquire and represent. Knowledge representation techniques allow knowledge to be acquired and organised so that it is accessible for use within an ITS.

Before accurate knowledge representation can be attempted, consideration of the type and nature of the knowledge required must be defined. At one level of abstraction, knowledge can be separated into *general* and *specific* knowledge. General knowledge is the domain-independent knowledge for a system. A typical example of general knowledge is the instructional knowledge component or teaching strategies used in many ITS; for examples see van Joolingen and de Jong (1992) and Spensley, *et al.* (1990).

Specific knowledge refers to domain or implementation dependent knowledge that is required by the system. The domain knowledge or domain model for a system is a typical type of specific knowledge in many early CAI systems. Recently there has been a move towards constructing system-independent knowledge bases. This allows the reuse of knowledge bases thus greatly reducing the construction time for ITSs in common domains.

The type of knowledge that is to be used must be defined before knowledge representation techniques can be applied. *Declarative* and *procedural* knowledge are two types of knowledge which are relevant to this process (Giarratano and Riley, 1989). Declarative knowledge is a collection of stored facts about a domain. When used in technical domains, it is sometimes called *system knowledge* or *device knowledge*. Procedural knowledge can be regarded as a collection of actions or procedures that an intelligent system can carry out (Schaafstal and Schraage, 1993).

The selection of a knowledge representation technique for modelling domain information is an important decision in the development of an ITS system. Kemp and Smith (1994b) note that the way that the domain is represented may be critical to the ease with which appropriate feedback can be given. Several knowledge representation techniques have been 'borrowed' from artificial intelligence and expert systems research and applied in the area of ITS development. Frames, semantic nets, predicate calculus and production rules are all formalisms that are used for

representing world concepts, their features and their classification (Direne, 1995).

Semantic networks are used in Carbonell's SCHOLAR (1970), where questions and answers in domains are generated by interrogating the network. More recently semantic networks have been utilised in KONGZI (Lu, Cao, Chen and Han, 1995) to represent the main portion of the domain knowledge. Semantic-net-like structures are used in the DISCOURSE project (Van Marcke, 1993) to describe a domain perspective in terms of domain objects and links. Semantic grammars are applied in the electronic trouble-shooter and tutoring system SOPHIE (Brown, *et al.*, 1982).

Rule-based structures, typically found in expert systems, are used in the 'Hunt the Wumpus' tutor (Yob, 1975) and in Clancey's GUIDON (1987b). Procedural networks have been developed by Sacerdoti (1977) for use in his planner NOAH and are used in BUGGY (Burton, 1982) to build diagnostic models. These models are used to provide a mechanism for explaining why a student is making an arithmetic mistake. Sowa's conceptual graphs (1984) are used to provide Arienti and Cazzaniga's UNIX_TUTOR (1990) with a deep and general model of the UNIX operating system domain, where its teaching is centred.

Feyock (1977) describes the use of transition diagrams in the development of CAI/HELP systems. He notes that transition diagrams have been found capable of modelling a wide enough class of concepts to be useful in many applications. Mark's work (1991) on the VCR Tutor also involved the use of transition-based state diagrams.

The use and application of knowledge representation techniques can simplify the development process by the use of 'tried-and-tested' techniques. However, the choice of technique can have a critical effect on the final system. The use of knowledge representation techniques to aid the development of ITSs is one of the issues addressed in this thesis and will be discussed in more detail in Chapter 4.

2.5 Discovery Learning Systems

Many researchers have found that simulation of a system coupled with a controlled exploratory approach is an effective means of teaching. Discovery learning and exploration based learning have been successful in the past in the development of computer tutoring systems; for examples see STEAMER (Hollan, *et al.*, 1984), SOPHIE (Brown, *et al.*, 1982), ICCARUS (Powell, 1992), SIEEL (Wigetman, *et al.*, 1992) and REACT (Hill Jr. and Johnson, 1994).

A simulation environment using discovery learning can provide students with an exploratory environment. This helps the student to identify basic concepts in an area of knowledge, to understand its general trends and, eventually, to feel the need for a formalisation of his or her discoveries (Bordier, *et al.*, 1990). This approach has been advocated by many educators for use in both traditional and computer-based instructional settings. Postulated benefits include greater 'intellectual potency', superior motivational effects, propensity to develop metacognitive skills, improved recall, transfer and analogical reasoning (Cox and Cumming, 1990). These systems allow the user to navigate around environments with varied degrees of system control. This follows a definition of a discovery learning system as a system where the student investigates the domain and is guided towards discovering important concepts.

Use of device models in discovery learning systems can provide high quality simulation without the high cost and danger of interacting with a real system. Experiments by Kireas and Bovair (1990) show that having a device model does improve performance on learning and remembering the operating procedures for a device. Kireas and Bovair also suggest that knowledge of how a system works helps by enabling the user to infer how to operate the device. Similar conclusions were found by Mark and Greer (1995). They show the benefits of using device models in a simulated VCR (Video Cassette Recorder) environment.

Unfortunately, discovery learning systems are not without their faults or critics. Much of the criticism of discovery learning systems has involved the lack of focus that the student may exhibit. Students can forget what they 'discover' and can even repeatedly discover the same thing (Peters,

1974). Aimless wandering may be undesirable in some domains and may lead to the user building misconceptions about the environment. Learners may be easily distracted from their main task and may end up 'lost in space' (Hulst, 1993).

Although computer simulations are a promising application of computer assisted teaching because they enable exploratory learning, research has shown that a simulation environment needs additional support for the student. Control and/or support are needed of the specific exploratory learning processes that are required for successful learning (van Joolingen and de Jong, 1992). This is also the case with discovery learning systems which, by definition, do not structure the material to be learned, consequently requiring the provision of additional guidance feedback. Cox and Cumming (1990) assert that for an effective and attractive environment for learning, there must be provision for advice and guidance.

One compromise to the free-exploration vs. guided learning problem is the use of *learning by doing* in a discovery learning system. Learning by doing is an advisory approach where the user can freely initiate actions. Each user move is compared with an expert move, as generated by the system. Feedback is provided to shape the user's responses towards the expert prototype. This approach has the advantage of placing more initiative in the hands of the user. This can be desirable as it allows the student to gain control of his/her learning speed. As there is a limited degree of structure imposed on the user, they can experiment and find their own solutions to tasks that may be set in a domain. Also, initial freedom in an environment can allow users to identify their own mistakes and then evaluate and implement appropriate counter-measures.

However, it is well known that if users are left to themselves, they automatically adjust their pace to suit their current knowledge level (Suppes, 1980). Also, learning by doing environments may entail more demanding knowledge requirements of the system, which must know a substantial number of ways that a users action can depart from those of an expert and the particular significance of each potential departure (Carroll and McKendree, 1987).

Discovery learning systems which utilise abstraction allow users to grasp higher-order concepts before having to deal with more detailed concepts. Abstraction is useful when complex mental models are to be constructed and the domain can be easily abstracted/decomposed. Also, the issue of comparing the simulation to the real world is relevant to this approach. Any model, by definition, is incorrect in some way. If it was not it would not be a model, it would be the real thing. If a model is going to be used for teaching purposes, consideration of the features of the original included in the model and how they are to be represented is needed (Kemp, 1995).

2.6 Environments for Authoring ITSs

As noted previously, the construction of ITSs is expensive, both in the time to build the systems and the cost of their development. As Kaplan and Rock (1995) observe, there are very few tools that exist to help create ITSs. This leaves ITS developers in a predicament when choosing tools with which to build one. Few commercial authoring tools encode domain knowledge, inferences about student knowledge or tutoring strategies for responding to a student idiosyncratically (Woolf and Hall, 1995).

Tools for creating graphical user interfaces typically lack a means for knowledge representation. Since the knowledge-based component must communicate with the interface, an ITS authoring tool must efficiently provide this capability (Kaplan and Rock, 1995). Specht (1989) maintains that an important objective in ITS research may therefore be perceived in the development of appropriate tools which also make possible the input of knowledge by the expert without the assistance of the knowledge engineer. To combat this problem, researchers have been developing systems to help reduce the difficulties of authoring ITSs. Two notable methods for building ITSs in the current literature are building ITSs out of existing components and the use of *authoring shells*.

The use of existing general software to build ITSs is an attractive option as much of the development work has already been done in the general software. Bordier, Paquette and Carrier (1990) report on the LOUPE project which focused on the building of discovery environments using generic

software such as Excel, SuperCalc and HyperCard. They found that using generic tools makes prototyping faster and facilitates interactive improvement through successive experiments with students. However there are certain flaws that limit the final application's educational possibilities. Firstly, generic software favours specific knowledge representation and access techniques which may be unsuitable for some domains. Secondly, much of the software proposed for reuse, eg. spreadsheets, databases and some hypermedia packages, only allow the presentation and storage of factual knowledge. Students can be tested on this knowledge but the result of the testing cannot be integrated back into the knowledge base.

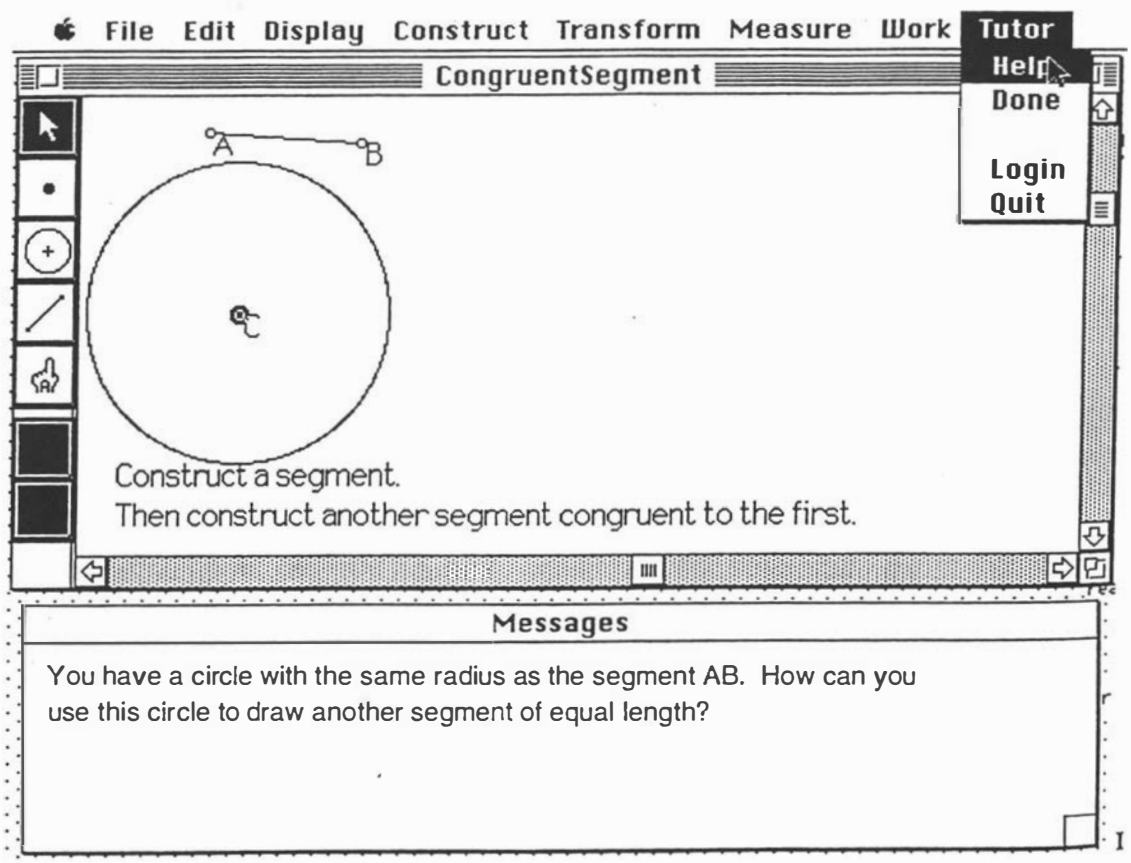


Figure 2.4 : Geometer's Sketchpad and a tutoring agent for geometric construction; from (Ritter and Koedinger, 1995).

Ritter and Koedinger (1995) also tried to incorporate tutoring elements into pre-existing software packages. Instead of building an ITS with generic software, they embedded *lightweight tutoring agents* within the existing software. They described their work on adding these agents into two environments: Geometer's Sketchpad and Microsoft Excel. Figure 2.4 shows a screen shot from the Geometer's Sketchpad with tutoring agent.

From the student's point of view the screen is identical to the normal Geometer's Sketchpad with the addition of the *Tutor* menu and a *Messages* window to provide feedback.

Ritter and Koedinger believe that technology has advanced to a point where it is practical to use lightweight tutoring agents to add tutoring elements to existing packages and they feel that the resultant systems combine the best elements of existing workplace tools and educational environments with guided instruction that has proven to be effective.

This type of ITS construction has potential for the applications involved but does not really solve the difficulties of reuse of components in future systems. The system above may indeed be very helpful in tutoring geometric construction in Geometer's Sketchpad but it is still limited to that domain. The need for domain flexibility in the construction of ITSs has led to the development of ITS authoring shells and environments.

Liddle, Brown, Slater and MacDonnchadha (1995) observe that in current industrial computer based training systems, there are two typical approaches. The first is based on providing high quality simulation with a human supervisor and the second is the use of computers to represent a particular task and to test the user on this task. Their system, MOBIT, has a toolkit approach to system development. Pre-developed modules representing the various expertise and instruction options are selected and combined for a particular training application. MOBIT is a general system that allows training in many domains and for different users. It provides multiple training interactions which can be used at run-time. To aid this process, Liddle, Brown, Slater and MacDonnchadha are trying to identify what possible generic criteria exists as a guide in determining which independent strategy should be used during a training session. Their current research is concentrating on identifying the domain independent strategies required for industrial training and on developing rules for their usage in ITSs. Domain independent strategies are provided and are linked with author created domain simulation and domain expert models to provide training; see Figure 2.5.

Multiple teaching strategies are embedded in pre-developed modules that can be used during the course of a training session. If the current strategy is proving ineffective then a new one is selected. This allows adaptive

training and ensures non-restrictive delivery of the required knowledge. The pre-developed modules represent the various expertise and instruction options available and can be selected and combined for a particular training session.

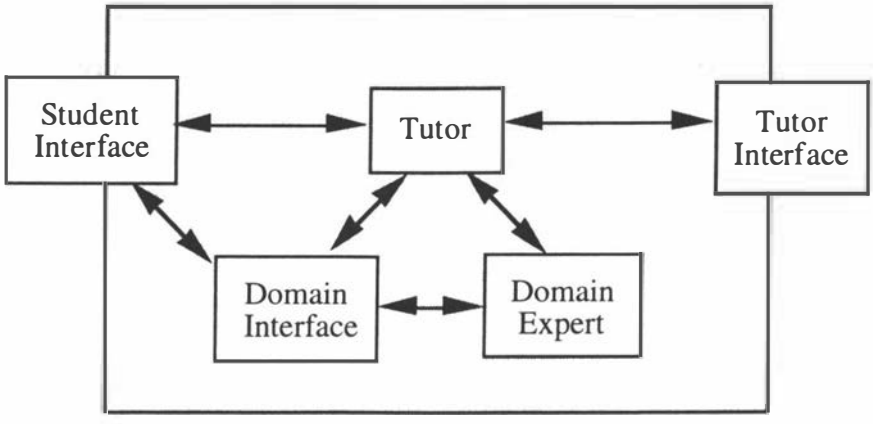


Figure 2.5 : MOBIT Architecture (Modified from Liddle, *et al.* (1995)).

RAPITS (Woods and Warren, 1995) is a rapid prototyping environment to build electronic book based ITSs. Following the work of COCA-1 (Major, 1993), RAPITS provides adaptable tutoring strategies to be applied on independent domain information. The constructed ITS adapts its tutoring to the current student and to the topic being taught. A system has been generated in the domain of teaching Pascal loop structures. Student assessment is done through the use of multiple choice, fill in the blanks or test by categorisation questions. ITS authors edit an electronic book template and link teaching strategies to domain topics. Also a high strategy, or *meta-strategy*, needs to be developed to determine what teaching strategy should be used in regard to certain student histories. Lesson material is stored as Microsoft Word for Windows files. This simplifies the authoring process as most teachers are familiar with word processors.

Rides (Munro, *et al.*, 1993) is an authoring system and computer based teaching environment that is oriented towards the development and presentation of simulations. These simulations are interactive graphical models. A graphical environment is supplied to allow authors to draw the system they want to simulate and to attach behaviours to the items in the system (Figure 2.6).

The simulation behaviour is centred around attributes and their values. A self-directed learning facility is also provided to add textual descriptions to the model. As noted by its authors, the most significant advance is that simulations do not have to be programmed separately. Units in the simulation can be drawn in an authoring environment and do not have to rely on limited libraries of pre-defined components.

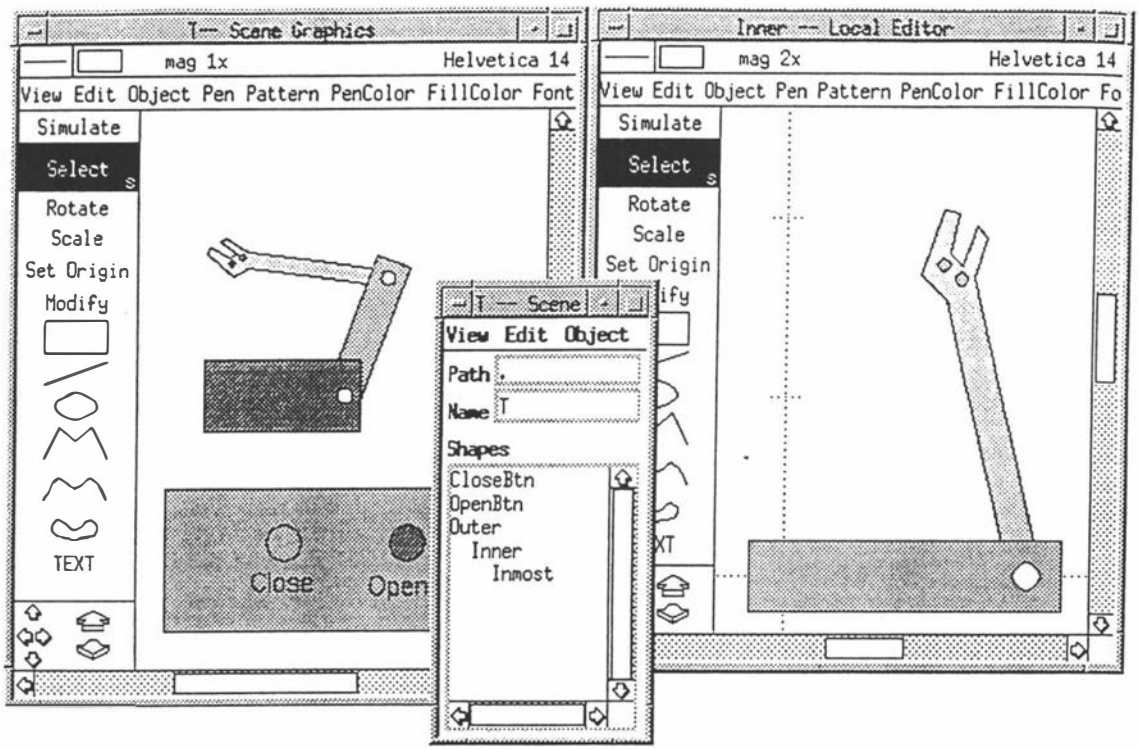


Figure 2.6 : Defining the behaviour of simulated objects in Rides; from Munro, *et al.* (1993).

An interesting approach to the authoring of courseware can be seen in Bell and Jackson's work on CALVIN (1993). CALVIN is a purely visual programming language that includes a CAI development environment. It functions in two modes, *student mode* and *author mode*. In the student mode, students can follow an interactive lesson where they navigate through windows, see diagrams, answer multiple-choice questions and view multimedia effects.

In the author mode, a human tutor builds a lesson structure using iconic representations of functions. These can be joined together using control-flow constructors such as loops. The author can then manipulate the lesson environment, eg. windows and menus, and set up testable

variables for arithmetic and conditional expressions. A screen shot of the CALVIN environment is shown in Figure 2.7.

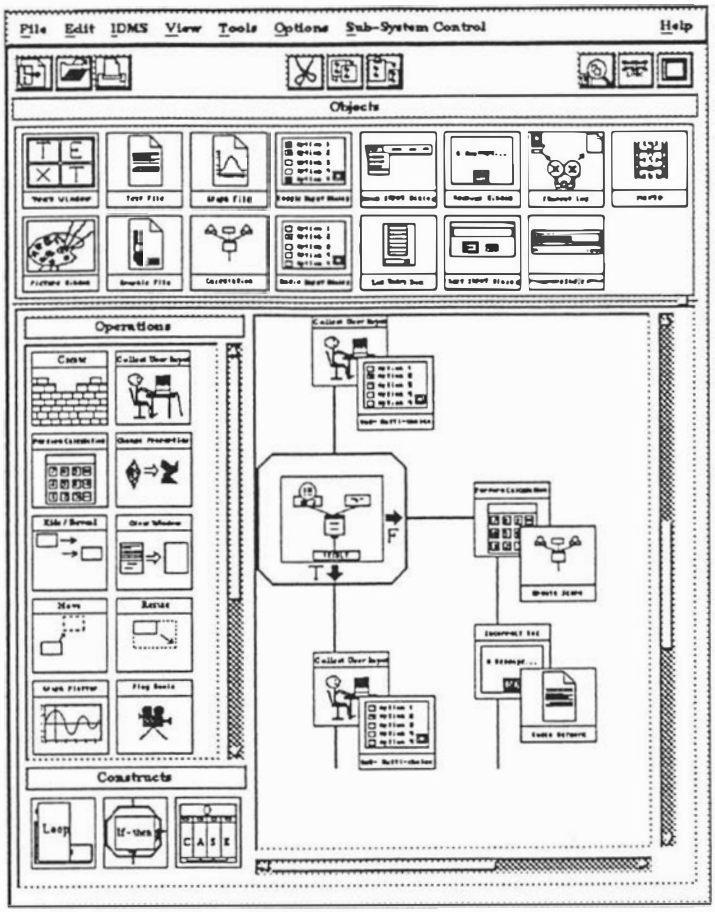


Figure 2.7 : CALVIN screen display; from Bell and Jackson (1993).

An alternative approach can be seen in the work carried out by Craske (1991) and Richards, Webb and Craske (1988). They are part of the EXCALIBUR Project which investigated what is involved in providing a general-purpose ICAL system, namely, one which provides an authoring and knowledge engineering shell.

Richards, Webb and Craske (1988) use an object-oriented representation of task/goal hierarchies and consider ways in which domain expertise, student information and teacher expertise can be made to interact with such hierarchies. Authoring consists of constructing a Goal/Task network for a tutorial domain and a component they refer to as a Domain Information System, which provides the domain data and processing that a student may need at each node of the Goal/Task network.

ITSAT requirements will be discussed, in terms of the current work, and the main area of specialisation will be introduced, namely, the teaching of procedural skills. How this is handled in the scope of a dynamic real world environment is the final concern to be discussed in this chapter.

3.2 ITS Authoring Tools

In recent years there has been an increased interest in the use of specialised software tools to aid in the process of ITS development. This interest has been encouraged by both the fact that constructing an ITS is a very time consuming and expensive exercise and that there has been little standardisation between developed systems or even components of systems. It would be desirable if some of the tasks of ITS construction could be automated. Two obvious candidates are for the automation of user interface construction and knowledge acquisition. One example from the literature is KONGZI (Lu, *et al.*, 1995), which uses knowledge bases that are automatically stripped from textbooks. Also ITSs could be more efficiently built if components were reusable between developments, for example, the knowledge bases and teaching strategies.

One aim of authoring environments is to provide a tool that a non-programming/knowledge representation expert can easily use to produce the required ITS. To this end, high level representations have been used to hide the underlying techniques so that the user does not need to be a proficient knowledge engineer, interface developer and also the application domain expert.

There are three distinct agents in the area of using authoring tools to aid the development of ITS: the authoring tool developers; the ITS authors and the ITS users. The authoring tool developers are concerned with building the ITS authoring environment. The users of this environment are the ITS authors which use this environment to construct their customised system. This in turn is used by students who are learning by interaction with the ITS. Figure 3.1 shows the ITS development environment and the agents which interact with it.

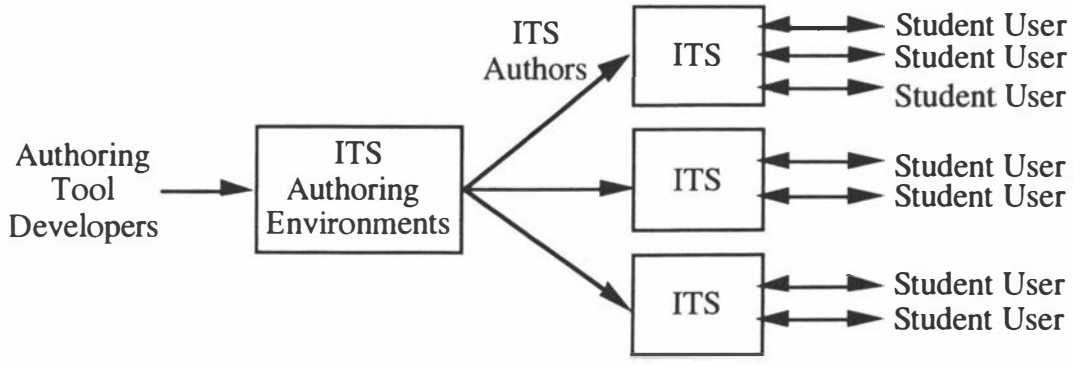


Figure 3.1 : ITS development environment overview.

ITS authoring is a complex task. The development of an authoring tool is even more difficult as it should address the identified ITS problems as well as its own issues. Thus the problems of developing an ITSAT are really a superset of the issues involved when building an ITS. Not only must they be considered but also these problems need to be presented to the ITS authors. Typical issues are author requirements, the level of system control, the use of cognitive models and how can knowledge base acquisition and reuse be encouraged and presented to the ITS authors.

Two major problems identified by Bloom (1995) need to be addressed in ITS authoring. Firstly, ITSs are complex and require the development of specialised knowledge representations for domain, instructional and student knowledge and specialised user interfaces for each domain. Secondly, there is little reuse of current ITS architectures across applications. Sarti and Van Marcke (1995) define reusability as the sum of retrievably and adaptability. Therefore, reuse can be seen, in general terms, as the ease with which ITS components can be recovered and modified for new environments. (NB. ITS component reuse is discussed in section 3.4.1.)

For several years, there have been a variety of commercial systems available to aid in courseware authoring. It would be helpful if such products could be used in the development of ITSs as they have many useful features. Unfortunately, many of these commercial systems have no intelligent components and need adaptive behaviour to be programmed into them (Major, 1995). This requires not only specialised programming knowledge but also detailed understanding of the AI

techniques that are involved. Therefore, most systems end up as Computer Based Training (CBT) systems or limited CAI systems.

Direne (1995) notes that to bridge the gap between CBT and ITS there is a requirement for an intelligent development environment, consisting of initiative methods supported by software tools that facilitate both the design of external-to-internal knowledge representation and the interpretation of such descriptions during tutorial sessions. This is one of the major difficulties of ITS development. The knowledge for an application must be acquired and stored in a form that can be used effectively by teaching methods. The main problem is primarily one of allowing authors to access formal methods and strategies without having to become proficient computer programmers or knowledge engineers (Direne, 1995).

In the current literature, there is no clear definition of what components should be in an ITSAT. Part of the problem of providing a generic definition for an ITSAT is similar to the problem of trying to define a generic ITS. The domain and range of application of the target ITS needs to be considered in the definition its ITSAT. As Dooley, Meiskey, Blumenthal and Sparks (1995) note, it is not well specified what components of an ITS system should be authorable. They suggest that the knowledge/domain base are basically compulsory with the possibility of authoring tutoring strategies and student-tutor dialogue rules.

Even if certain components are considered as prime candidates for being part of an ITSAT, how many of them will be default components and how many should be customisable (Major, 1995)? Should a domain independent tutoring engine, similar to a Expert System Shell inference engine, be provided or should teaching methods be able to be totally re-defined by the author. Also, how can custom user interfaces be defined for generic systems?

For a generic ITSAT, the most sensible approach would be to take the standard ITS model, comprising the domain knowledge, a tutoring engine, the teaching methods and a user interface, as a basis for components to be authored. Then, within the scope of the desired target domain environment, it can be determined which components should be customisable.

The types of tutoring system construction systems available are extremely varied. In recent years there has been an increase in the number of commercial available tools for building tutoring systems. These system are primarily CAI construction tools that aid authors in the definition of courseware. Major (1995) considers two different classes of authoring systems. Firstly, the commercial systems, Authorware (Macromedia, 1995), Tool Book (Asymetrix, 1997) and SmarTText (Thibeault, 1994) and secondly, the research tools IDE (Russell, Moran and Jordan, 1988), KAFITS (Murray, 1996; Murray and Woolf, 1992), COCA (Major and Reichgelt, 1992) and GTE (Van Marcke, 1992). This classification is also used by Dillenbourg, Schneider, Mendelsohn and Borcic (1995).

The commercial systems allow the design of declarative instruction. These tools allow an easy entry and display of information, in the form of text, pictures, movies and sound (Ritter and Blessing, 1995). Help with lesson planning and the provision of pre-defined interface components ease the task of computer tutor construction. These tools allow an author, such as an instructional designer, to create simple courseware relatively quickly (Dooley, *et al.*, 1995). Direne (1995) observes that two of the prime advantages of these tools are that they include extensive resources for the definition and direct manipulation of interface objects and the ease of use.

However, commercial tools do not provide any AI to the systems they build. All the adaptive behaviour is pre-programmed by the system designer. This leads to mainly system-passive tutorial interactions. This is due, in some part, to the lack of an expressive scripting language to encode intelligent behaviour inside these tools.

3.3 Expert System Shells and ITS Authoring Tools

In some respects, ITS authoring tools can be considered comparable to Expert System Shells (ESSs) used to aid the development of expert systems. Similar to the development of ESSs, ITS authoring tool development has been driven by the need to reduce system development costs (Blumenthal, Meiskey, Dooley and Sparks, 1996). ITSATs and ESSs are intended to allow non-programmers to benefit from the effort of others who have solved a problem similar to their own. A basic

framework is provided in both systems, an inference engine for an expert system and a combination of tutoring engine and/or teaching strategies for an ITS. Relevant domain knowledge for the current domain can then be entered into the system by an author.

Jackson (1990) and Turban (1992) note that not all ESSs are suited to all tasks. This is also true for ITS authoring tools. With both types of development tools there is a trade-off between the generality of the tool and the power of the constructed system. Waterman and Hayes-Roth (1983) share this view and maintain that tool builders naturally want to develop general-purpose tools applicable to a wide range of problems.

Reye (1996) maintains that, similar to ESSs, a certain level of ITS domain-independence can be achieved by keeping all the domain-dependent knowledge in the knowledge base and using domain-independent methods for representing the knowledge in the knowledge base (Figure 3.2).

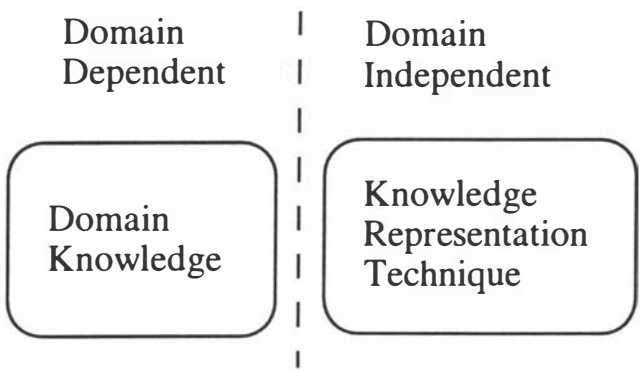


Figure 3.2 : Domain knowledge and representation technique separation.

The separation of the knowledge base, or domain knowledge, is only part of the problem of domain independence. There are also the issues of domain independence for the tutoring engine and the system interface. In ESSs a general inference engine is provided that functions with a defined knowledge base. A similar concept can be applied to ITSATs. If the tutoring engine can be based on the knowledge representation technique involved, it too can be kept domain independent. This can similarly be applied to teaching strategies over the domain knowledge (Figure 3.3).

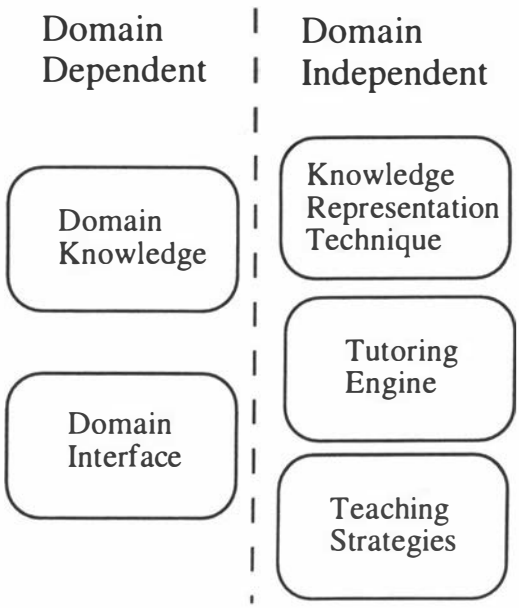


Figure 3.3 : Components with domain dependence and independence.

Unfortunately, a truly domain independent ITS development tool may be impossible to achieve as it is difficult to design a tool that can accommodate both knowledge from a variety of domains and also provide deep knowledge about any given domain (Hsieh and Redfield, 1995). Dooley, Meiskey, Blumenthal and Sparks (1995) also suggest that one compromise is to build an ITS shell within a class of domains so that several similar domain-specific tutoring systems can be developed. Thus, the scope of the development shell can be limited to some extent and be more focused on the area involved.

Another advantage of ESSs is that many are available on personal computers e.g. IBM PCs and Apple Macintoshes. Therefore, they are inexpensive when compared to purchasing a complete ES development environment (Turban, 1992). The same trend is also emerging in the ITS field as it provides the opportunity for systems to be used by a wider audience for a reduced cost. Also, moving the development of ITSs to common platforms allows for easier use of the finished systems in real world environments. Another related trend is that of keeping components of the finished systems platform-independent. This is especially relevant if a commercial tool is to be constructed as it provides flexibility in the implementation platform of the final system.

3.4 ITSAT Requirements

The construction of ITS authoring tools is an extremely complex task. To try and build one tool that is usable in any domain with any number of teaching strategies would be impossible. The variety of possible ITSs that may be required is too great. To counter this, many ITS authoring tools concentrate on a specific area, domain type or teaching style. Even when this is done, there are several issues of ITSAT development that require special consideration. These are the reuse of system components, the degree of system control, the use of visual notations, the construction of domain interfaces and the generality of the systems being developed.

3.4.1 Reuse

One of the fundamental problems in the development of an ITS is that there is little or no reuse of components between ITS developments. Therefore, there is no reduction in the cost of ITS construction between systems as each one is treated as an one-off development. Many components of ITSs lend themselves to reuse. For example, the knowledge base components and the teaching strategies.

Sommerville (1996) notes several advantages of software development with reuse, including:

- increasing system reliability as reused components have already been tested.
- making effective use of specialists as they can encapsulate their knowledge in reusable units.
- reduction of software development time as both development and validation time should be reduced.
- support of organisational standards as reused components can be implemented to a set of standards.

This view is also shared by Pressman (1997), Schach (1993) and van Vliet (1993).

Reuse of ITS components is a fundamental feature of most ITS authoring tools. As many components of ITSs are time consuming and labour intensive to originally construct, it is sensible to reuse as much as possible.

Sarti and Van Marcke (1995) note that the production of courseware can be greatly improved by the reuse of the results of previous projects. Typical components to be reused are the domain knowledge bases, teaching strategies and presentation media.

As noted in section 3.3, the reuse of domain knowledge bases is something that has been encouraged in the development of expert system shells. As with ITS knowledge bases, the acquisition and development of expert system knowledge bases is extremely costly, especially in terms of utilising the domain experts time. An important step in the possible reuse of the knowledge bases is the process of separating the knowledge base from the rest of the ITS.

One way that this can be done is by the separation of the domain knowledge from its presentation. Vassileva (1995a) suggests that this separation supports authoring by re-using existing materials. If the domain knowledge can be kept independent from the presentation technique, for one ITS, a variety of different presentation methods can be built over the domain knowledge, and for multiple ITSs, the knowledge base can be easily used for future systems based in the same domain (Figure 3.4). This flexibility can add to the power of an ITSAT if it is combined with the reuse of presentation material.

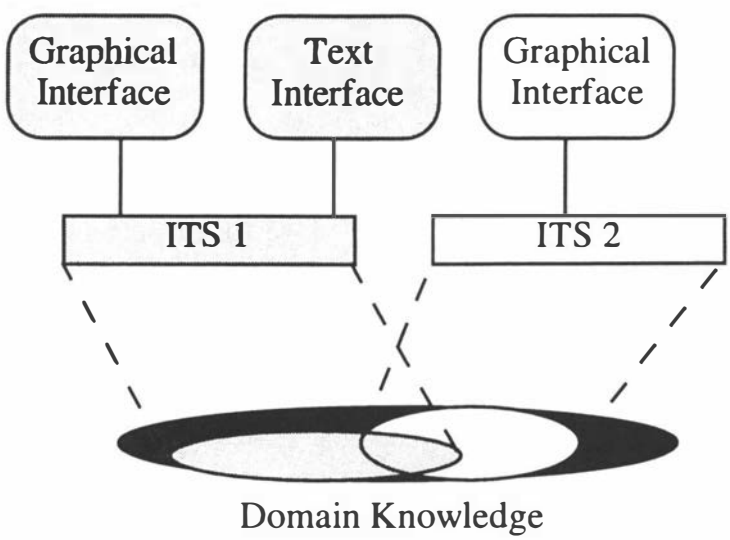


Figure 3.4 : Sharing of independent domain knowledge.

Recently, with the increased usage of multimedia based material, to be economically viable and extensible, authoring shells for ITS must be able

to reuse instructional media (Suthers and Lesgold, 1995). To enable this there needs to be a separation between the multimedia, the interactive characterisation of the learning material and its instructional semantics (Sarti and Van Marcke, 1995).

Many multimedia products are already available in standardised formats. For example, video clips in QuickTime, MPEG and AVI formats, still graphics in GIF, JPEG and PICT formats and sounds in WAVE, AU and MOD formats. Current interest is in building on the current standards provided on the World Wide Web (WWW) by HTML (HyperText Markup Language) and the use of WWW navigators. Material placed on the WWW is already mostly platform independent with viewing navigators like Mosaic (Mosaic, 1994), Netscape Navigator (Netscape, 1995) and Internet Explorer (Microsoft, 1995) being freely available across platforms. Bloom (1995) notes that most ITSs require specialised delivery platforms and that WWW and HTML use can provide a platform independent delivery source.

Several researchers are investigating the use of the WWW as a medium for ITS presentation (Bloom, 1995; Brusilovsky, Schwarz and Weber, 1996; Suthers and Lesgold, 1995) and if it is deemed feasible, reuse of HTML documents could greatly reduce the development of presentation material that is needed for new ITSs.

Another typical way that presentation media can be reused is in interface construction tools which can be used to build user interfaces. Libraries of interface components can be presented as a toolkit resource for tutors to assemble. Providing tools to ease the construction of the interface component of an ITS is essential, as without appropriate interfaces, learning can be reduced, especially if users can easily lose interest in a tutoring session.

A third concern is teaching strategy reuse. Teaching knowledge or teaching strategies have often been hard coded into educational applications. This complicates the reuse of that knowledge in subsequent systems. This problem has been caused by a lack of proven methodologies for ITS construction. Combining the teaching strategies and the control of the system together in an *ad hoc* manner has made it difficult to separate the components for possible reuse. Jona (1995) maintains that by

identifying and codifying general teaching strategies we may be able to address the methodological problem that prevents reuse of teaching knowledge in building educational software. The use and application of ITS independent teaching strategies will be investigated further in Chapter 5.

3.4.2 System Control

One problem that is encountered in the development of an ITS authoring tool is how much control should the system have over the development of ITS components. One important consideration is which of the critical instructional decisions are going to be default behaviours of the ITS authoring tool and how many will be in the hands of the authors (Major, 1995).

As one of the aims of developing ITS authoring tools is to relieve the authors of tasks for which they have no expertise, the problem of generality arises again. If a tool is suitably general, how can it cope with the individual needs of different ITS authors? Also, there is the problem for a general ITS of having appropriate interactions with a variety of different users. Dooley, Meiskey, Blumenthal and Sparks (1995) note that the success of a tutoring system tool will depend on its general applicability, the ease of building new applications as well as its ability to provide the learner with an engaging task and an authentic context in which to perform skills.

One approach to increasing usability for instructional developers is to put instructional, interface, design and programming under the control of the development tool (Hsieh and Redfield, 1995). In Hsieh and Redfield's system, AIDA, the instructor builds a database describing a piece of equipment or system and develops and inserts graphics to illustrate the equipment. AIDA then uses the equipment knowledge in the database to generate tutorial-like instruction with embedded practice questions and feedback. This process is facilitated by the authoring tool by specialising the terminology and concepts presented by the system to those from the current domain.

Direne (1995) observes that one solution to the complexity of building ITSs is to limit the authoring scope. In this approach *common knowledge*

in a category is encoded as domain independent descriptions. These descriptions are then used by an authoring tool in a transparent form for the ITS authors. Examples of common knowledge from existing ITS systems can be found in the area of teaching computer programming (Anderson and Reiser, 1985; Lee, 1990) and the domain of electric circuit troubleshooting (Brown, Burton and Kleer, 1982; White and Frederiksen, 1987).

Another level of control that an authoring system can have over the construction is the monitoring of the authoring process. If the scope of the domain is limited to some domain area, the authoring system can monitor and provide feedback to the author in the form of identifying possible errors and inconsistencies in the current descriptions. One goal of the work by Blumenthal, Meiskey, Dooley and Sparks (1996) is to provide tools that 'watch' the authoring process. These tools detect and diagnosis the work of the human author, keeping track of possible problems, for example, completeness of courseware and whether a grammar contains disconnected conversational pieces. A similar idea is proposed by Paquette, Aubin and Crevier (1994). Their system contains an advisory component that is used to help a designer using a set of system development tools. Rules for coherent and efficient use of the available tools are used to help the author. For example, if the skill level of the ITS is not consistent with the training needs identified in an analysis phase, a rule will fire to display an appropriate warning message.

3.4.3 Visual Notations for Knowledge Acquisition

Harel (1988) observes that visualising information, especially information of a complex and intricate nature, has for many years been the subject of considerable work by many people. The use of visual notations can be especially useful for knowledge acquisition. As many potential users of ITSATs have little or no detailed knowledge of knowledge representation techniques, the use of a consistent graphical notation can ease their learning requirements for the knowledge acquisition phase of an ITS's development.

From the viewpoint of an ITSAT, providing a visual notation is dependent on two things. Firstly, the knowledge representation technique that is to be used and secondly, the type of interface that is to be

utilised by the ITSAT system. The choice of knowledge representation technique is the focus of Chapter 4. The selection of an interface type is complicated by the issues of the systems delivery platform and system resources. Again, these issues are more appropriately investigated in the next section and in the context of the TANDEM system, described in Chapter 6.

3.4.4 Developing Domain Interfaces

Due to falling cost in graphical oriented hardware (monitors, CD ROM and laser disc players, and direct input devices) and the availability of these tools, there is a limitless variety of user interfaces that can be produced. Thus, the only real limitation on interface construction is the time available to construct the interface and level of detail that is required for the teaching that is proposed. The marriage of multimedia to ITSs offers the potential to create intelligent systems that can instruct and demonstrate, using sound, animation and video (Kaplan and Rock, 1995).

Unfortunately, the construction of user interfaces is usually outside the expertise of many potential ITS authors. Unless a large development team is working on an ITS project, an expert in the domain is usually the key developer. Providing high-level toolkits for interface design and development can readily reduce the need for extensive interface construction knowledge. This is useful as a major process in designing and constructing an ITS is building the interface that the students use. Two approaches to the development of interfaces can be seen in Figure 3.5. In Figure 3.5 (a), the domain content is directly linked to user interface and in Figure 3.5 (b), the domain content is kept separate from the interface development.

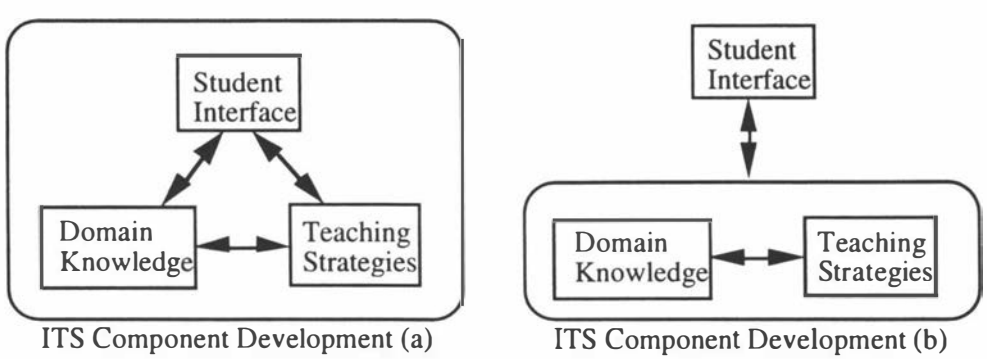


Figure 3.5 : Two approaches to interface development.

Ritter and Blessing (1995) maintain that in many cases the design of the student interface has been coupled with the design of the curriculum and lesson plans. This can result in enormous development tools which are sometimes difficult to use and can be inflexible in their application.

One of the main advantages of keeping the interface development separate is that there is no set restriction on the type of interface that can be used. If the aim of the ITSAT is to be as general as possible, providing flexibility in interface selection is very important. Ideally, provision so that any interface development tool could be used to construct user interfaces would be desirable.

However, the move to separate the interface from the domain is not without its own disadvantages. An independent interface tool has no context to base its development in. Each interface must be completed from scratch. If an interfaces development is linked to a domain, then domain presentation components can be collected into a resource library to aid the interface development. This encourages the reuse of interface components for ITSs in similar domains. While this reuse is desirable, the cost is the flexibility of interface design. Also limiting interface construction to current technology can severely compromise the useful life time of an ITSAT.

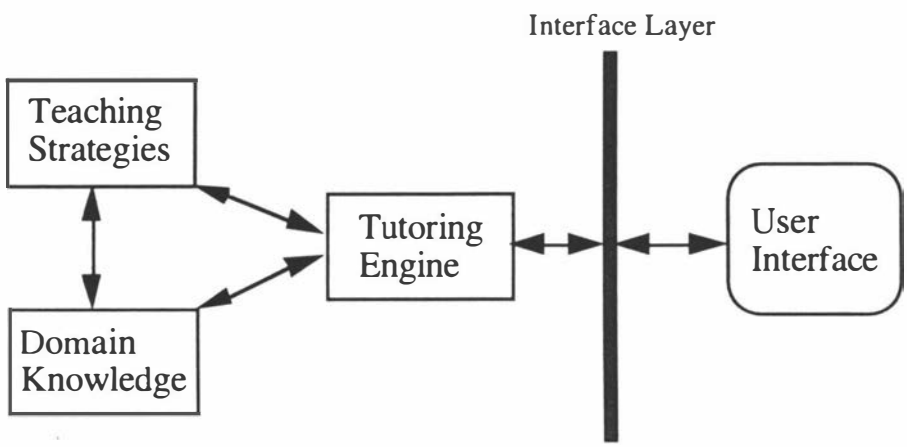


Figure 3.6 : ITS interface separation.

In the current work, the second approach has been taken. The potential benefits in the flexibility and freedom in interface selection and advantages of keeping components of an ITS independent seem justified.

To enable a domain independent interface to be added to an ITS there needs to be some standardised layer between the interface and the ITS system (Figure 3.6). Further discussion on the interface layer between the user interface and the other ITS components developed in the current work will be described in Chapter 6.

3.4.5 General vs. Special Purpose Tools

The general applicability of an ITS authoring tool can become a major concern in its development. It would be desirable if a totally general ITSAT could be developed. Unfortunately, there are multiple levels of generality issues when constructing ITSs which need consideration. Firstly, there is the generality of the completed ITS. Domain-independence is desirable for an ITS but is hard to achieve without limiting the teaching power of the system. Secondly, the types of teaching strategies that can be supported and defined suffer if a general approach is taken. Thirdly, a general tool must provide a generic interface to apply to a wide range of tasks. Due to its generality, this interface must be domain-independent and thus limited. Bell (1995) shares these views and notes that tools which aim for generality have the potential to support the design of a wide range of applications but often impose limitations on the modes of interaction in the target application, offer little design support and rely on general models of instruction.

Although a high degree of generality may seem desirable, Blumenthal, Meiskey, Dooley and Sparks (1996) believe that a good implementation of such authoring tools will be extremely difficult to achieve. Attempting to provide a tool with a high-degree of generality leads either to ITSs that are pedagogically weak or that require development efforts not significantly less than those for individually crafted ITSs.

It may not be advisable to construct general purpose ITSATs, but rather to concentrate on tools with general features for a specific domain or type of task. Bell (1995) maintains that this approach can provide tools with special-purpose rather than general models and which are built upon an explicit model of construction. As the environment is focused in one area, these tools are more likely to be able to provide a rich interaction development environment and will be better able to provide design guidance. The main disadvantages to this approach is that there is a

narrow range of designs that can be developed and there is dependence on the model that is promoted by the tool.

Also by specialising tools, each supports the design of a specific kind of educational software. This has the advantage that design knowledge is easier to code, due to the narrower scope and bonded range of potential users and interfaces may now be more specific to the tasks involved. Unfortunately, such tools can lose their utility.

One way that specialisation can be achieved is by limiting the learning environment, i.e. the types of teaching strategies employed, or the representation techniques could be defined based on some set cognitive model. Examples of this specialist approach can be seen in Bell (1995), a tool for constructing environments for learning by doing in Goal Based Scenarios, and LEAP (Bloom, 1995) and its authoring tools built in the domain of customer interaction skills.

A third approach is the combination of general and specialised approaches. To do this, an identification of the components of an ITS that would benefit most from being specialised or general is needed. Although the trade-offs between teaching power and flexibility are still present, a compromise would greatly improve the usefulness of the finished tool.

Again the issues of what information is domain independent and what is dependent must be taken into account. As stated in previous sections, the two main domain dependent components of an ITS are the domain knowledge and the domain interface. This leaves the tutoring/control component and the teaching strategies that are to be used. If these can be kept domain independent, then half the standard ITS model can fulfil the domain independence requirement that promotes reuse of components and the generality of the final system (Figure 3.7).

This approach is encouraged if separate tools are used for knowledge acquisition and user interface development, thus simplifying the development of future ITSs based in the same environment by the reuse of the domain independent components.

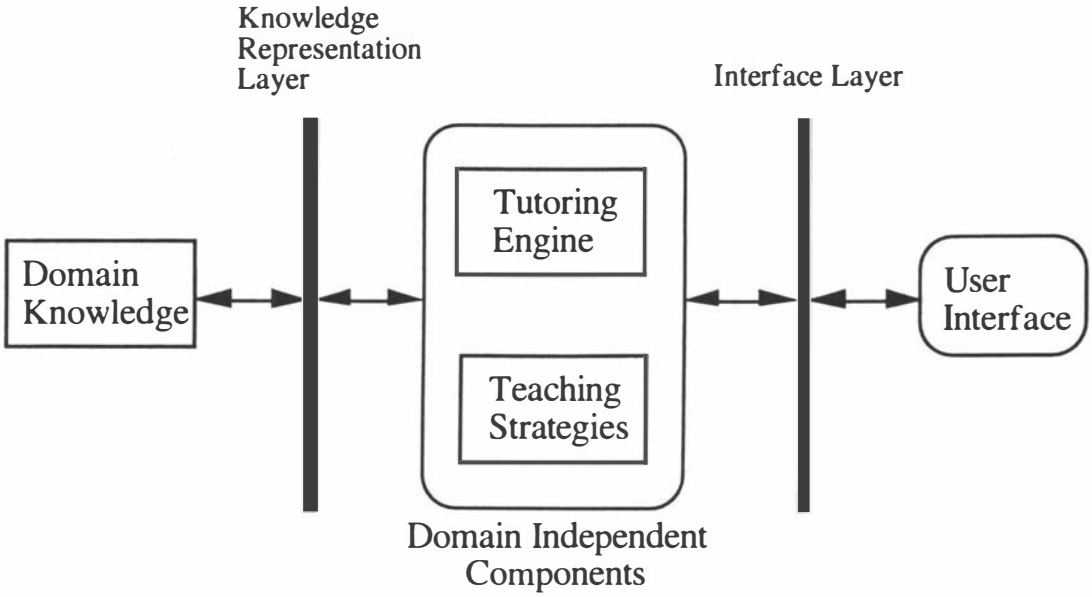


Figure 3.7 : Separating ITS components.

To limit the scope of the current work in line with the issues discussed in the third approach, three elements of the ITSAT have been specialised: the knowledge representation technique, the environment for tuition and the type of problems that can be tutored, namely, the teaching of procedural skills.

3.5 Teaching Procedural Skills

Teaching users to operate equipment or to learn complex procedures can be a time-consuming and expensive experience. This is especially the case when equipment is costly or fragile. Also, if the equipment is to operated in a dangerous environment, on-the-job training may be impossible. A solution to these problems is the use of computer based simulated environments.

There are several advantages to using computer simulations as the basis for a teaching environment. Firstly, equipment behaviour can be accurately defined. The real world usually provides many unpredictable effects and random elements which may be detrimental if learning in one focused situation is required. Secondly, many environments are either too dangerous or can lead to a potentially disastrous conclusion. For example, when training a technician in the operation of a nuclear reactor, hands-on training may not be acceptable. Also, if equipment or materials

to be used are very expensive, training on-the-job can be prohibitively expensive.

As with other computer based training, the use of computer simulations has the advantage of providing tutoring to multiple users at the same time and once packages have been developed they can be reused when necessary. Thus training that requires one-to-one tuition can be more readily provided. Unfortunately, the construction of an adequate simulation environment is not an easy task.

Liddle, Brown, Slater and MacDonnchadha (1995) observe that computer based industrial training has two approaches. The first is the development of a full-replica simulation using experiment-based training with a human supervisor. This type of simulator is very expensive, both in development and for its human supervision, but has the advantage of providing very realistic training environments and allows experimentation where it would otherwise be difficult due to safety reasons. Their other approach is for the computer to represent a task and to test trainees in the correct execution of that task. Although this environment is more restrictive than the first approach, it can be very effective for well defined procedurally based training objectives.

This second method of learning has been found particularly effective when a process is simulated and the student either interacts with the modelling system or has to attempt to rectify problems when a process is not operating correctly (Kemp and Smith, 1994c). Anderson, Boyle, Farrell and Reiser (1987) maintain that when teaching procedural skills in a problem solving context, students appear to learn information more effectively if they are presented with that information during problem solving rather than being taught general principles.

Kemp and Smith (1994a) note that the teaching of procedural skills has been an important aspect of research into intelligent tutoring systems right from the influential work carried out by Goldstein (1979) and Burton (1982). Such methods have been used to create discovery learning type environments in a variety of different domains including M16 rifle maintenance (Miller and Lucado, 1992), boiler operation (Hollan, *et al.*, 1984), VCR operation (Mark and Greer, 1995) and fire fighting command (Newman, *et al.*, 1992).

The current work is focused on using techniques to teach procedural skills in a computer based simulated environment. The main emphasis is on tasks that have sequential steps with some possible partial ordering, especially in the area of equipment maintenance and manipulation.

However, when providing a simulated environment for a domain, there needs to be some restrictions on the simulation to allow the scope of the model to be easily defined. Without any restriction on the simulated models, they would become huge and their production and manipulation would be unmanageable. The major restriction on the current work is the use of a *static world assumption* to keep it to a manageable size and function.

3.5.1 Static World Assumption

In the real world, environments are inherently dynamic. There are scores of independent and dependent free agents that can dynamically change the current state of the world. Simulation of this type of environment on a computer can be extremely computationally expensive, if possible at all. A solution to this problem is to attempt to reduce the operational overheads of the environment without sacrificing the fidelity of the simulations. Thus some constraints need to be added to an environments definition so that a restricted view can be defined.

When teaching procedural skills, it is important to be able to identify when tasks have been completed. Also, if a student is to be taught a certain skill by some defined method, it may be confusing to have the environment dynamically changing as the focus of the lesson may be lost. There are three observations that affect simulated environments:

(i) the user should be able to apply a series of operations on the environment.
(ii) only the user can interact with the environment.
(iii) there should be no time constraints.

How do these observations affect the nature of the physical domain? As the environment consists of applying sequences of operations in the simulation with no explicit time considerations, the domain is termed

static between operations. Also, since the environment simulates the effect of only those actions to be performed by the user, there is an assumption that no other user/agent acts on the domain. Also this approach assumes that the physical world is effectively and completely representable by a database of facts, so the environment has access to all information necessary for the simulation.

Sanborn and Hendler (1988) refer to these assumptions collectively as the static world assumptions (SWAs). They note there is no need to model action in progress as the simulation models the environment as a set of conditions (the current situation) that holds before and after user actions are executed.

When teaching procedural skills, the use of the SWAs as the basis for domain modelling simplifies the definition of domains, while maintaining a realistic environment. In domains related to equipment maintenance or device operation, this approach is helpful in focusing the domain modelling process and reduces the computational overheads for their manipulation.

The SWAs are the basis for the definition of domains that are to be modelled and manipulated in the current work. They allow the domains to be modelled in a way that allows users to easily directly observe the results of their actions and provide well-defined parameters for planning through these domains.

As described in Chapters 4 and 5, planning techniques may be used to build local plans for allowable paths through the domain space. As Sanborn and Hendler (1988) note, a planner in this environment is omniscient - it controls and co-ordinates all the actions and is completely aware of the consequences of the actions involved. Also, they maintain that plans generated under the SWAs may be thought of as state transition diagrams, with arcs representing actions between states.

3.6 Summary

The construction and use of tools to aid in ITS development is fraught with many pitfalls. Unfortunate choices in the ITSAT design process can

limit the flexibility and usefulness of the final ITSs. The ITSAT developer must be aware of ITS development issues as well as ITSAT issues.

As described in this chapter, there are many parallels that can be drawn between ITSATs and the work being done in the area of Expert System Shells. Both have similar goals and suffer from many of the same problems. The issues of component reuse, system control and the development of domain interfaces are especially relevant in these areas and consideration of these issues is very important.

Also, when trying to build something of a general nature, the problem of generality versus specialisation occurs. In the current work, a mid-point between these two approaches has been adopted where the tutoring engine and teaching strategies of an ITS have been kept general while specialisation has been encouraged in the areas of domain definition and the development of user interfaces. This can provide a flexible environment that still has the power to adequately model a variety of domains.

The teaching of procedural skills is a valuable area of study for the development of ITSs as there are many real world environments where such tutoring is particularly relevant. Simulated environments for dangerous or material expensive domains can be defined and specific skills tested. Using restrictions like the static world assumptions allow limits to be placed on system development but do not necessarily adversely affect the learning environment. In the current work, the SWAs and the teaching of procedural skills are two of the foundations that the research is based on. This enables the domain dependent, independent and general components of the proposed ITSAT and target tutoring systems to be finalised.

In Chapter 4, harnessing the power of the general components will be discussed with an investigation into the use and generation of domain knowledge.

Chapter 4

Knowledge Representation

4.1 Introduction

Although the tutoring component of an ITS can be viewed as the 'heart' of a system, it cannot function without accurately represented knowledge about a teaching domain. The domain knowledge provides the context and domain information that is required for the teaching process to be applied. Somehow this domain knowledge needs to be represented so that it can be economically used by an ITS.

This chapter describes some knowledge representation techniques that have been borrowed from artificial intelligence research and applied in the area of computer based tutoring systems. The following sections provide a firm theoretical foundation for the current work and describe the definition and use of two fundamental models for ITSs: the domain model and the task model.

4.2 Domain Model

The main contribution of a domain expert to the development of an ITS is the definition of the domain knowledge. Especially when a simulation environment is to be constructed, accurate knowledge acquisition is required so that the final model is a realistic representation of the chosen environment. As noted in expert systems literature, the acquisition of such information is a non-trivial procedure. Jackson (1990) notes that

knowledge elicitation interviews generate between two and five 'production rule equivalents' per day. He identifies three reasons why productivity is so poor. Firstly, the technical nature of specialist fields requires the non-specialist knowledge engineer to learn something about the domain before communication can be productive. Secondly, experts tend to think less in terms of general principles and more in terms of typical objects and commonly occurring events. Finally, the search for a good notation for expressing domain knowledge and a good framework for fitting it all together, is itself a hard problem. This is before one gets down to the business of representing the knowledge in a computer.

The final point above deals with the use of knowledge representation techniques to allow effective storage, accessing and maintenance of knowledge. This will be discussed in a later section with regard to the techniques used in the current work.

Training an existing knowledge engineer for each domain to be modelled is a costly exercise and may require extensive access to the domain expert's time which is in addition to the rest of their commitment to the development project. The knowledge difference between the domain expert and the knowledge engineer, especially early in the project, can easily slow down the initial development.

Also, when a model has been developed, domain experts will be required to validate the model (Gaines, 1990). If the model is constructed by a domain expert in an incremental fashion, modelling and validation testing can be interspersed. This conforms with the second point above, where an expert can observe or generate events in the model and see how they are processed.

One way to reduce the cost-per-hour of domain model construction is to dispense with a specialist knowledge engineer. If the domain expert is able to directly input information into the domain model, there is a minimal requirement for the knowledge engineer. To facilitate this, tools to help the domain expert use specialist knowledge representation techniques will have to be defined to allow tutors to construct domain models.

4.2.1 Defining the Domain Model

The knowledge representation technique used becomes important for computer assisted tutoring systems as many systems require the use of student and domain models. Typical examples include BUGGY (Brown and Burton, 1978), the LISP tutor (Anderson and Reiser, 1985), GUIDON (Clancey, 1987a), DOMINIE (Spensley, *et al.*, 1990) and the VCR tutor (Mark and Greer, 1995). These models need some underlying representation that is efficient, in terms of storage and manipulation, and appropriate for the system that is to be developed. Even with current technological advances and access to large volumes of memory, the need to store and manipulate information makes the choice of knowledge representation technique a critical factor in the use and expandability of a system.

When looking for a representation technique to model some domain, a natural choice is a state based transition network. This representation technique has been used successfully in a variety of domains (Feyock, 1977; Mark and Greer, 1995).

Transition networks are directed graphs where nodes are associated with some state of a system and arcs represent the transitions that enable movement from state to state. A simple example from a photography domain can be seen in Figure 4.1. Unfortunately, transition networks can become inefficient when used to describe large domain spaces due to the combinatorial explosion of representing all the transitions between states (Smith and Kemp, 1995).

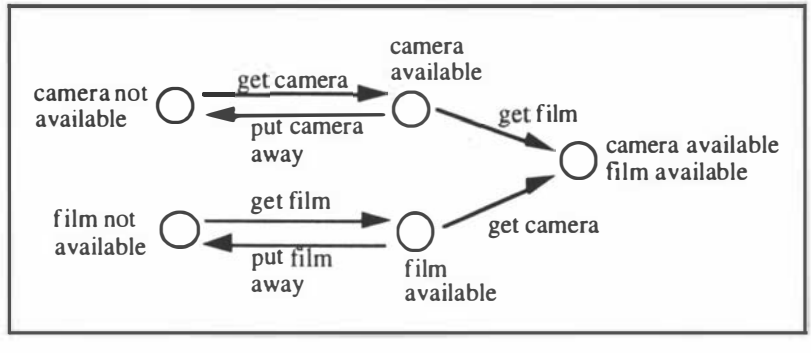


Figure 4.1 : Transition network example.

Another approach to states and their transitions was developed in the AI planner known as STRIPS (STanford Research Institute Problem Solver) (Fikes and Nilsson, 1971). Although it was originally developed as a problem solver to be used to search a space of 'world models' to find one in which a given goal is achieved, it has been used as a basis for educational software. Peachey and McCalla (1986) used STRIPS-type operators in their CAI system when teaching concepts to students.

Kemp and Smith (1994a) detail the benefits of a STRIPS-type notation modified to facilitate the generation of feedback in a variety of domains. This notation has been modified into a simplified version based on the TWEAK planning system developed by Chapman (1987).

Corresponding to states in the transition networks are *situations*. Each situation comprises a set of conditions that describe some current aspect of the system or domain. Events in the modelled environment have an effect on the current situation. Events are typically user actions and are enabled by preconditions and cause postconditions in the current situation. This processing of events and the manipulation of conditions can be summarised in a table of preconditions, operators and postconditions or a POP table (Smith and Kemp, 1995).

4.2.2 POP Tables

POP tables are a tabular representation of a STRIPS-type notation. Events in an environment are described by the use of operators. These are listed in a table format with their relevant preconditions and postconditions. The current state of an environment is maintained as a list of conditions called the *current situation*. This list has all the conditions that are currently enabled at one moment in a domain environment and is used to validate the preconditions of operators when they are attempted. The preconditions of an operator determine which conditions must be in a domains current situation before that operator is enabled. The postconditions are the conditions that are added to the current situation after the application of an operator.

POP tables are used to provide feedback about the domain behaviour and can generate explanations about this behaviour. Explanations into why an action cannot be taken can be obtained by referring to preconditions

that are not currently satisfied. Also, the addition of a postcondition to a state space description can detail how the current environment has changed. An example of a POP table can be seen in Figure 4.2. If the action *open door* was attempted while the door was already open, the action would fail as the precondition *door closed* is not satisfied and that precondition could then be used in error feedback to the user.

Precondition	Operator	Postcondition
door closed	open door	door open
door open	close door	door closed

Figure 4.2 : Example POP table.

Although POP tables provide a basis for domain definition, they are not expressive enough to completely define a domain. For example, in Figure 4.2, there is no obvious connection between *door closed* and *door open*. They are just syntactic labels and have no contextual relation. This is undesirable because it allows the possibility for contradictory conditions to be present in the same instance of some domain, eg. *door open and door closed*. What is needed is some way of structuring context between conditions and representing this in the domain definition.

Tenenberg (1991) provides a formulation where the preconditions and postconditions are simple atoms and additional static axioms are included which define relations that are assumed to be always satisfied. The use of static axioms adds greatly to the power of the domain description since there are often a large number of conditions that remain the same throughout the execution of a process. Also, it enables the description of what does change to be formulated much more succinctly (Kemp and Smith, 1994a).

In regard to augmenting the POP tables with the static axioms, two distinct types of static axioms have been identified. Firstly, *maintenance static axioms* are used to provide consistency in a current situation of a domain between conditions after operators have been executed. When a condition is added to an instance of the current situation, the maintenance static axioms maintain the current situation by removing

any related conditions that may need to be deleted. An example for the POP table above can be seen in Figure 4.3. In this example if *door closed* is added to a situation, *door open* will be removed if it is present, as both these conditions cannot be present in the same situation.

'door closed' THEN not 'door open'
'door open' THEN not 'door closed'

Figure 4.3 : Example maintenance static axiom.

The second type of static axioms are referred to as *standard static axioms*. A standard static axiom consists of a rule for deriving a condition from the current situation. Tenenberg separates conditions in a situation into those that are derivable and those that are not, called *inessential* and *essential* respectively.

Essential conditions are independent, non-derivable conditions whose enablement does not depend on other conditions in an environment. These conditions are entered into the current situation through the postconditions of a completed operator and can be used as preconditions for operator enablement.

The inessential conditions are dependent on other conditions in the current situation. It is not necessary to store these conditions in the current situation as they can be derived if they are applicable. As inessential conditions are not stored in the current situation, they are not valid as postconditions because all postconditions get added to the current situation. However, this lack of storage does not affect their ability to be used as preconditions as their enablement can be derived from the current situation.

A standard static axiom is a rule that describes the conditions that are needed to derive an inessential condition. These derivable conditions may be a mixture of essential conditions or other derivable conditions. Figure 4.4 shows the current POP example defined using a standard static axiom. In this example, the condition *door closed* is the derivable condition as it is only valid if the door is not open.

Precondition	Operator	Postcondition
door closed	open door	door open
door open	close door	

'door closed' IF not 'door open'

Figure 4.4 : Modified POP table and standard static axiom.

The standard static axioms are used to provide another mechanism to describe relations between conditions in a domain. Also, as inessential conditions are not directly stored in the current situation, the size of this structure can be significantly reduced. However, this space reduction is offset by the extra processing that is required for deriving the inessential conditions.

The examples of static axioms shown have had only one condition on the right hand side (RHS) of the rule. This is a simplification of the rules for these examples. Both standard and maintenance static axioms can contain complex boolean expressions. Some complex examples from a VCR domain can be seen in Figure 4.5.

```
tape activated IF vcr playing OR
                  vcr recording OR
                  vcr fast forwarding OR
                  vcr rewinding

vcr playing THEN NOT (vcr recording) AND
                   NOT (vcr fast fowarding) AND
                   NOT (vcr rewinding)
```

Figure 4.5 : Further static axiom examples.

The use of static axioms allows a richer domain description to be defined and provides techniques for linking the relations between conditions. The two types of static axioms can be seen as being directly related to either operator postconditions or preconditions. The maintenance axioms

provide domain consistency when updating a current situation with postconditions after the application of an operator while the standard axioms list the derivable conditions that can be used as preconditions in operator validation.

4.2.3 Graphical POP

Although the POP tables provide an adequate representation technique for the definition of domains, they are still a textual representation and it can be awkward to work at this low level when modelling large domains. From a human perspective, large POP tables are difficult to comprehend and the consequences of additions and deletions of operators from a table may not be initially apparent. When POP tables are defined, each operator entry is independent of the other entries in the table and their only relations are the condition relations defined by the static axioms.

Editing the textual versions of POP and static axiom information for domains with more that twenty or so operators can become tedious and is very error-prone. Adding and deleting operators and the resultant change in the associated conditions, can have effects on the representation that are not obvious. Also, keeping static axiom definitions consistent with the current model is a non-trivial task.

A graphical notation that provides the same power as the POP tables but is easier to manipulate and can provide a more conceptual view of the domain as a whole would greatly improve the usability of POP tables as a representation technique. Also, graphical notations can be easier to learn and can promote better understanding by novice users. Both of these are desirable if non-expert tutors are to use this representation for domain knowledge definition. Therefore, it would be useful to have some notation that was more transparent for development purposes and could be readily used by non-computer specialists.

Many different graphical techniques were examined to see what features would be needed if a new graphical representation technique was to be developed. There are several graphical notations that have been used in Expert System and ITS development. These include: transition nets (Feyock, 1977), petri nets (Leiden, O'Donnell, Peterson and Shenk, 1987),

higraphs (Harel, 1988), statecharts (Harel, 1988), procedural nets (Sacerdoti, 1977) and conceptual graphs (Sowa, 1984).

Initially, Sacerdoti's *procedural nets* seemed a likely candidate as the representation technique as the focus of this research is on procedural domains. The procedural net is a graph structure whose nodes represent actions at varying levels of detail, organised into a hierarchy of partially ordered time sequences. An example procedural can be seen in Figure 4.6.

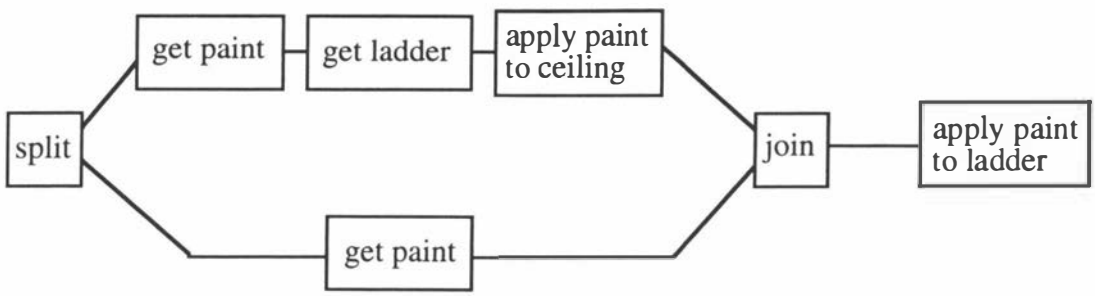


Figure 4.6 : Procedural net for painting; from Sacerdoti (1977).

The time sequence of a procedural net is read from left to right, where the arcs on the net are taken to mean 'before' in the sequence of actions. Drummond (1985) observes that due to the strict temporal ordering of rules in procedural nets, they have an inability to describe iterative behaviour as one cannot simply direct an arc from an action back into the net to a node that has been previously completed. He also observes that since true iterative behaviour is difficult to model in a natural way using procedural nets, all procedural net state-spaces will be loop-free. This means that using procedural nets would limit the types of domains that could be modelled and even extremely simple domains could not easily be defined.

An extension to Sacerdoti's procedural nets, called *plan nets*, was developed by Drummond (1985). These have been modified to correspond to the POP table representations (Kemp, 1995; Kemp and Smith, 1994d). Plan nets are action driven and can be used to maintain the current valid conditions in an instance of an environment.

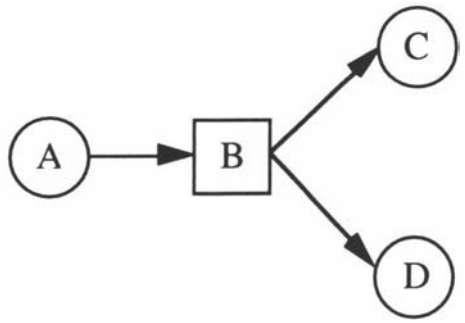


Figure 4.7 : Basic plan net structure.

Figure 4.7 shows an example of a plan net. Circles are used to represent conditions and squares used to represent actions. In the example above, *A* is a precondition of action *B* and action *B* causes postconditions *C* and *D*. This process of action enablement and condition causation matches the definition of the POP tables; see Figure 4.8.

Precondition	Operator	Postcondition
door closed	open door	door open
door open	close door	door closed

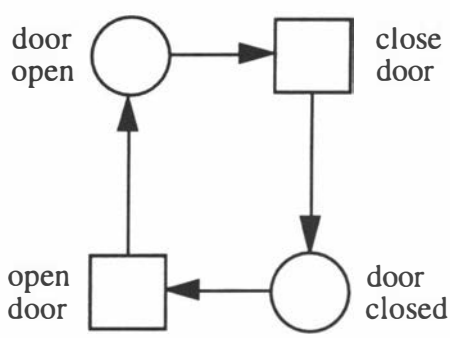


Figure 4.8 : Plan net and POP table.

There is a direct correspondence between the basic POP table and the plan net representation which makes the plan net an acceptable choice for the basis of a graphical POP notation. The static axiom information still needs to be represented and a graphical notation that is consistent with the plan net notation has been developed to allow graphical definition of both the standard and maintenance static axioms. As the static axiom expressions may contain the boolean operators *AND*, *OR* and *NOT*, an appropriate graphical notation has been developed to accommodate them. This

notation has been based on the pictorial view of AND/OR graphs (Slagle, 1963). More recently, Vassileva (1995b) has used an AND/OR graph structure for modelling domain knowledge in the TOBIE system.

This notation is based on horizontal directed graphs with nodes representing conditions, essential (clear) and inessential (grey) (Figure 4.9), and arcs used to indicate the boolean operation AND in an axiom rule. These graphs are read from left to right with the RHS of the graph representing the RHS of the axiom rule. Also, maintenance static axioms are distinguished from standard static axioms by the use of dashed lines connecting the conditions. General examples of standard and maintenance static axioms can be seen in Figures 4.10 and 4.11 respectively, where A and E are inessential conditions and B, C and D are essential conditions.

Figure 4.9 : Condition types.

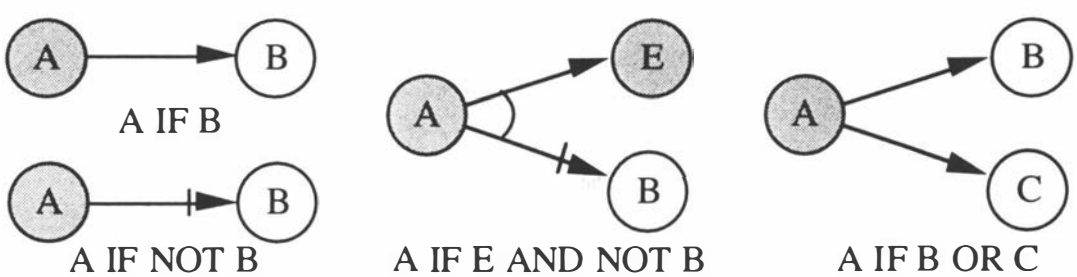


Figure 4.10 : Graphical standard static axioms.

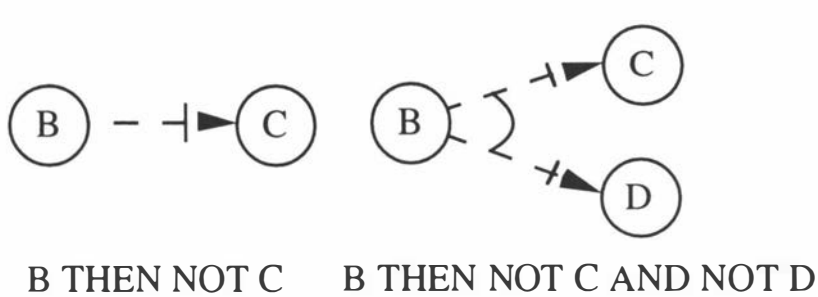


Figure 4.11 : Graphical maintenance static axioms.

It should be observed that although boolean operators are used to define the static axioms, only simplified use of these operators is supported. Providing complete representation of complex boolean expressions would unduly complicate the graphical notation without providing any substantial increase in the notations power. Also, by limiting the graphical notation to the general examples, provides a notation that does not allow contradictions to the basic static axioms ideas to be constructed. Several static axiom definitions that are enforced by the graphical notation are summarised in Figure 4.12.

Standard Static Axioms	Maintenance Static Axioms
Only one condition on LHS	Only one condition on LHS
Only inessential conditions on LHS	Only essential conditions on LHS and RHS
Inessential or essential on RHS	No explicit OR constructs (separate rules are required)

Figure 4.12 : Static axiom rules.

Applying this notation to the current example can produce two equivalent graphical POP representations (Figures 4.13 and 4.14).

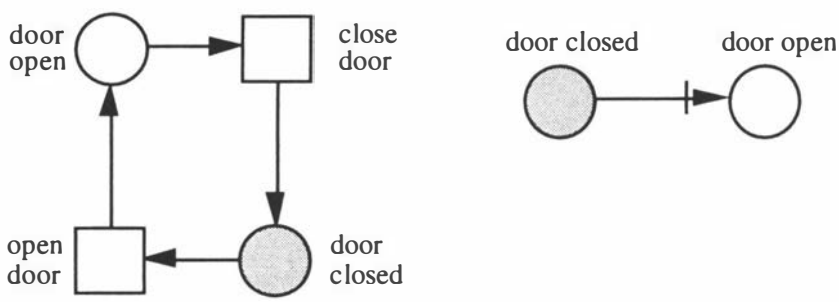


Figure 4.13 : Graphical POP with inessential conditions.

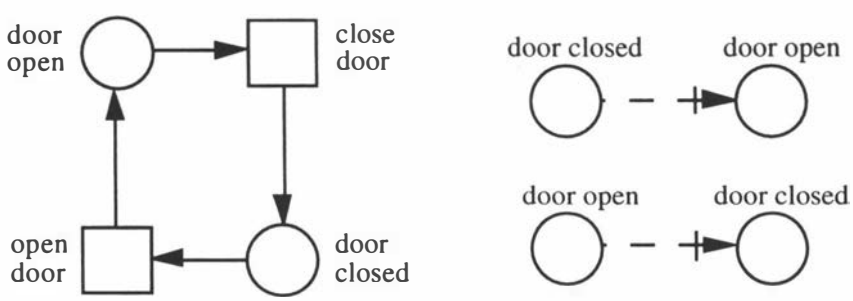


Figure 4.14 : Graphical POP using maintenance rules.

Both the examples above are valid representations of the same domain description. The use of either standard or maintenance static axioms can be arbitrary in many cases. One thing to note is that in the first of the two examples only one condition needs to be stored in the current situation while for the example in Figure 4.14, two conditions need to be stored. Such savings may be considerable if defining an extremely large domain.

The use of POP and static axioms allow the base level domain model to be defined. This is the minimum that needs to be developed to allow users to navigate through an exploratory environment. When teaching procedural skills, some notion of task is needed to allow guidance to be given to users. For this, a task model must be defined by the tutor.

4.3 Task Model

What has been considered so far is the representation of a domain. A model that has all the appropriate functionality of a corresponding domain can be constructed and users can be shown the results of their explorations in the environment. Unfortunately, only feedback on what *is* happening, not on what *should* be happening can be given. To differentiate between what is required and what is possible, the development of a task structure is needed.

Instead of directly considering tasks within a domain, it can be helpful to develop a suitable domain representation and then build the task representation on top of this. Kemp and Smith (1994a) identify several issues that need to be addressed before a suitable task representation can be formulated.

Firstly, the term *task* needs to be defined. The term has been used in a variety of ways throughout the ITS literature; see Hill Jr. and Johnson (1993), Kemp (1995) and Mark and Greer (1995). In the current work, task is an activity or sequence of activities that may be carried out to achieve some goal. A *goal* is a collection of conditions that are satisfied in the current situation of an environment. There may be many ways to complete a task by reaching a goal. Thus, task is a general description of a procedure where there may be many instances of different procedures to

achieve a goal. The separation between task and procedure is important for the second task consideration.

If a student is to complete some goal, is the main focus of the teaching the task or the procedure to complete the task? If the object of the training exercise is to program a VCR (Kemp and Smith, 1994a) or disassemble a M16 rifle (Miller and Lucado, 1992), it might be more appropriate to concentrate on an optimal procedure for achieving the goal than to encourage alternative solutions. However, if the notion of completing the task and developing multiple solutions to the task is important then limiting the user to one procedure would be undesirable. In the STEAMER environment (Hollan, *et al.*, 1984), an interactive inspectable simulation-based training system for a steam propulsion system is provided. In such an environment, there may not be one set way to complete a task. The type of responses needed are determined by a combination of different factors present in the current environment and having one strict rule for a task would force an operator to memorise, by rote, hundreds of special circumstance procedures.

Thirdly, there is the question of how feedback is related to a task. In a discovery environment, basic domain feedback is supplied by the domain model. When tasks are overlaid over the domain, there needs to be another level of task-oriented feedback. In a tutoring environment, when is an appropriate time for feedback to be presented? Should it be after a user makes an undesirable move, as a warning before the move, when the user requests help or after the user can no longer satisfy the current goal? In the ITS literature there are advocates for various positions on task related feedback. One of the simplest models of the VCR tutor (Mark and Greer, 1995) does not allow the user to make moves from an optimal path. Other options on a graphical interface are greyed out. The LISP tutor (Anderson and Reiser, 1985) allows user mistakes but immediately points out the mistake while STEAMER (Hollan, *et al.*, 1984) provides an environment with unlimited freedom for the user. In REACT (Hill Jr. and Johnson, 1994), the system recognises when the student has reached an *impasse*, a tutorial interaction point where the student might benefit from intervention by the tutoring system, because the student's current action has failed or cannot achieve its intended purpose in the device's current state. Then the system provides feedback to coach the student through the impasse.

A further consideration is the amount and type of feedback that is given when a task related error is encountered. Just indicating that an error has been made may be suitable in some situations, but in others, more comprehensive feedback may be required. Kemp and Smith (1994a) suggest some alternatives include telling the user exactly what to do next, listing all acceptable possibilities or indicating some higher level sub-goal that the user should attempt to satisfy.

4.3.1 Defining the Task Model

Kemp and Smith (1994a) contend that increased insight can be given to users by producing a more elaborate model separating out the two considerations of domain and task. This separation of the domain and task models allows the application of domains in a variety of task environments.

After a domain model has been constructed, task based structures can be built over this model. Treating these structures as overlays on a portion of the domain model allows a flexible approach to the model building process as domains can have multiple task structures, based on different tasks, overlaid on the one domain; see Figure 4.15.

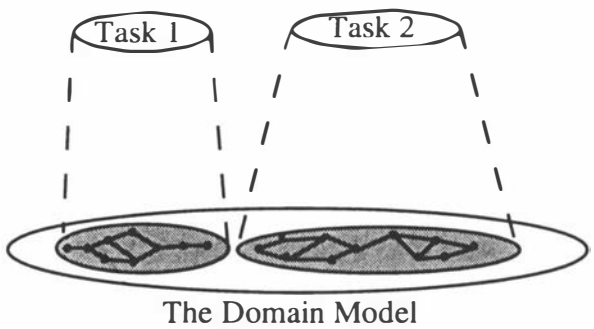


Figure 4.15 : Task overlays on the domain model.

This makes the task development process more economical as much of the base representation is already completed in the domain model.

In the current work, task structures are used to augment the feedback and guidance capabilities and are defined on two complementary levels. Firstly, situated control rules (SCRs) (Drummond, 1989) provide local

guidance for the current situation of an environment and secondly, a student task model (STM), which is a goal hierarchy of goals and sub-goals for a particular task, can be used. This is similar to the *situated explanation* learning approach (Huffman and Lairds, 1994).

Situated control rules were developed by Drummond to characterise the performance of possible actions in terms of a planning agent's current environment. They are synthesised through temporal projection over a state space and are used to constrain the behaviours produced by plan nets. Each SCR characterises the performance of possible actions in terms of an agent's current environment. This approach considers possible sequences of actions that may be carried out and pinpoints actions that are critical to the completion of a given task.

An intermediate step in the generation of SCRs is the production of a *projection graph*. This temporary structure is a transition net like graph where the possible actions in a state space environment are mapped out. Figure 4.16 shows an example projection for a microwave domain. This is a simplified microwave representation where there are only two objects in the domain, a cup and the microwave itself, where the microwave has been constrained so that the microwave is off whenever the door is open and on when the door is closed.

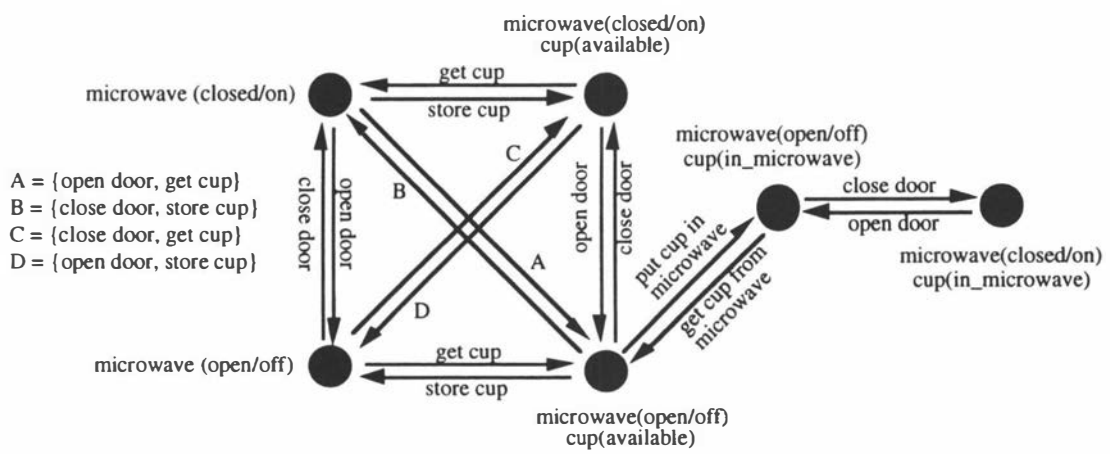


Figure 4.16 : Example projection graph for a microwave domain.

One crucial difference between standard transition networks and the projection graph is that projection graphs allow transitions between states with more than one action. As long as these actions are independent and can be carried out in any order, they can be mapped onto a single

transition. An example in Figure 4.16 is transition A, *open door, get cup*. Both these actions are currently independent and either can be completed without affecting the outcome of the other.

Once a projection graph has been generated, undesirable states and *critical choice points* in the graph can be identified. Undesirable states are states that are permissible by following the domain model but are not to be encouraged. Two undesirable states are marked in grey in Figure 4.17. In both these states the microwave is on while it is empty. Although this is a totally valid action in the domain, it is not recommended for actual microwave usage. Noting undesirable states helps to identify critical choice points in the projection graph.

Critical choice points are states in a projection graph that have an action or actions which can lead to an undesirable state. The critical choice points are the states where SCRs are needed. SCRs are used to indicate appropriate actions at the critical choice points. In Figure 4.17, a path to the goal *microwave(closed/on), cup(in_microwave)* has been highlighted.

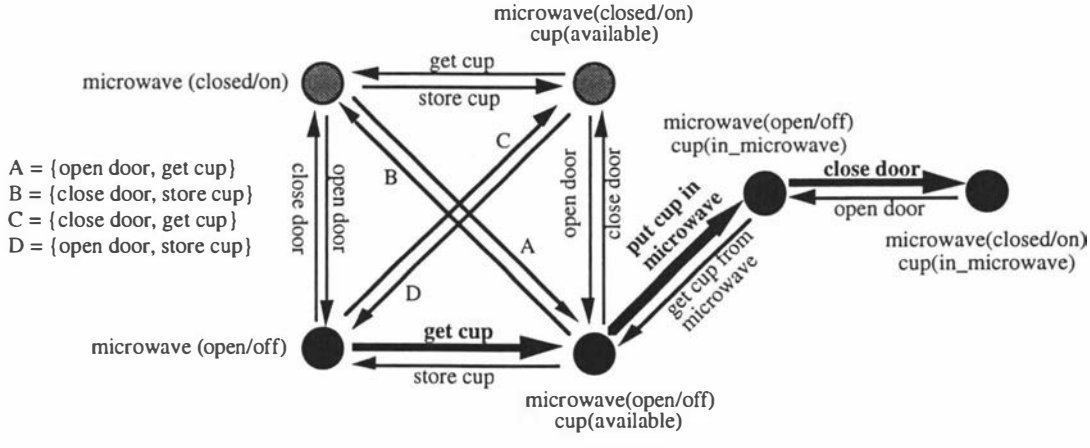


Figure 4.17 : Preferred path through the projection graph.

Once the critical choice points and a preferred path through the graph have been identified, the following SCRs could be generated:

SCR1: microwave(open/off) -> get cup
SCR2: microwave(open/off), cup(available) -> put cup in microwave
SCR3: microwave(open/off), cup(in_microwave) -> close door

It should be noted that these three rules have been developed for the specific goal *microwave(closed/on), cup(in_microwave).* If a less strict task structure was required, then only actions that lead to a dangerous state may be targeted. In this case, only the two critical choice points would need associated SCRs; see Figure 4.18.

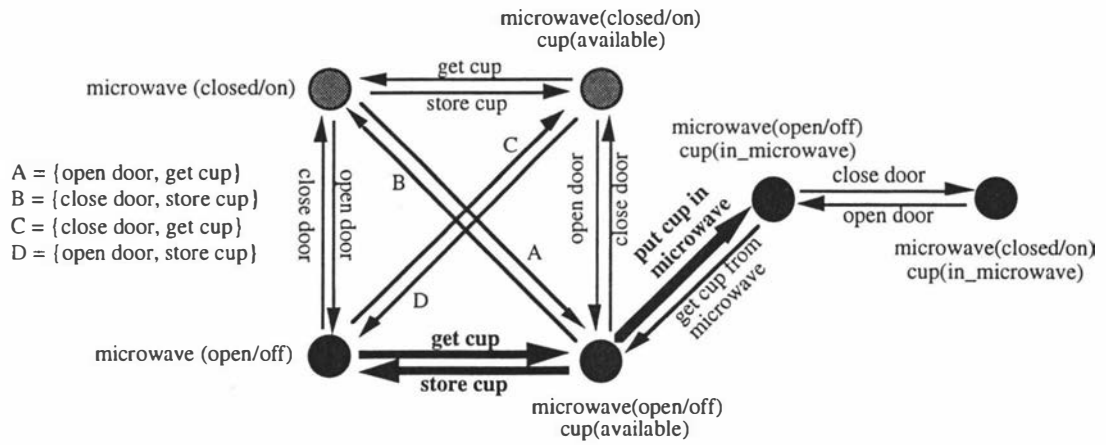


Figure 4.18 : Indicating encouraged paths through the projection graph.

SCRs for this general structure would be:

SCR1: microwave(open/off) -> get cup
SCR2: microwave(open/off), cup(available) -> put cup in microwave
SCR3: microwave(open/off), cup(available) -> store cup

Although SCR1 and SCR3 may cause an advice cycle, this may be acceptable if a general modelling approach is being taken.

One thing that neither of the previous two approaches have taken into account is the guidance that may be necessary if a user does find him/herself in an undesirable state. If the SCRs are not being enforced, a user may wander into such a state. In this case, guidance may be required to exit from this state. Figure 4.19 shows a possible solution by identifying all the desirable actions from critical choice points and the undesirable states. Thus, advice can be given to allow the user to exit from the undesirable state as soon as possible.

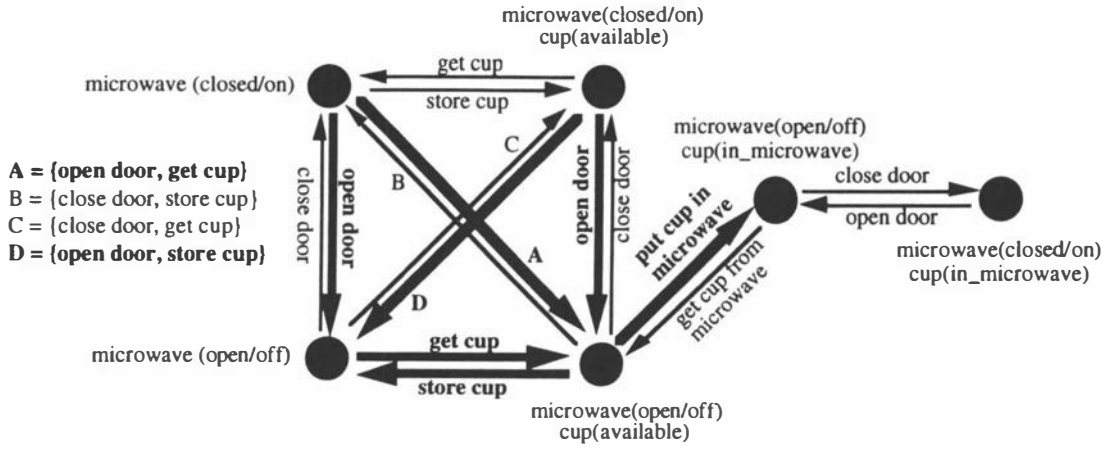


Figure 4.19 : General SCRs in the microwave domain.

As seen in Figure 4.19, SCRs can be used to provide a varied degree of advice when a user reaches a choice point. Bad or inappropriate actions can be indicated or a user may be forced to follow only appropriate paths. Critical choice points can be marked by a tutor on a projection graph for a defined domain and these can be mapped onto the domain model (Figure 4.20).

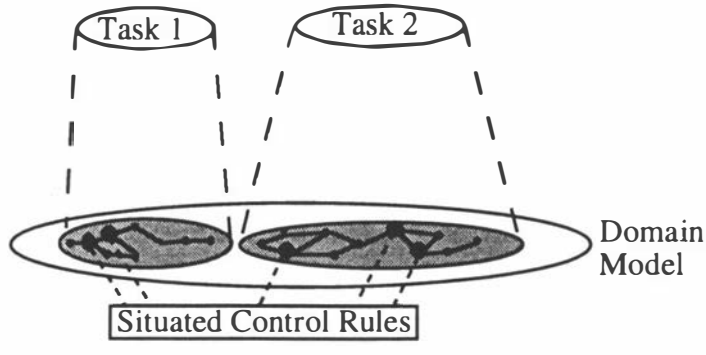


Figure 4.20 : SCR critical choice points.

SCRs are generated for a domain before users begin teaching sessions, therefore the overhead of producing a transition based projection graph does not affect the efficiency of the run-time use of the tutoring system. SCR generation and their use will be described in more detail in Chapters 5 and 6.

Whereas the SCRs are general rules for tasks in the domain, the student task model provides a task by task hierarchy of goals and sub-goals. This can be used as a student oriented overlay model for each task. As sub-

goals are completed these can be marked on the hierarchy. This provides a clear view of the students' current progress through a domain. An example of a student task model can be seen in Figure 4.21.

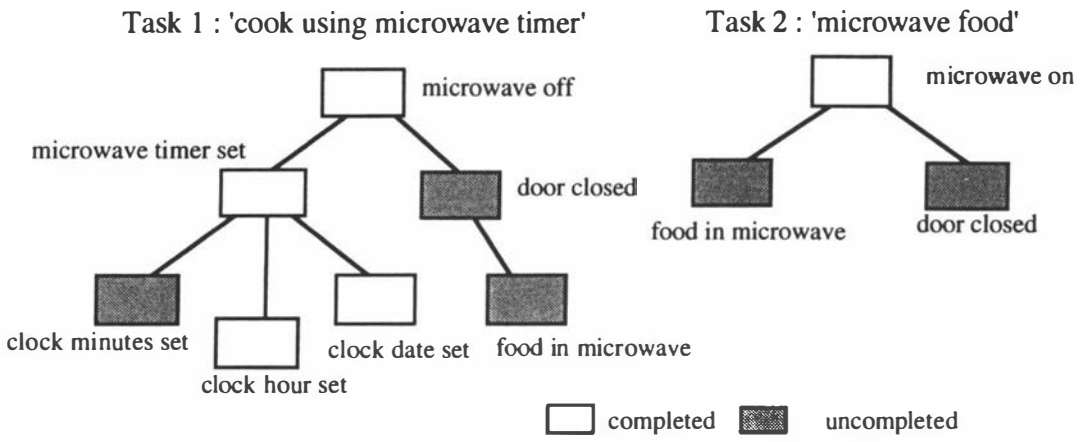


Figure 4.21 : Example student task models.

The student task model is not a hierarchy of decomposed goals and sub-goals but more of a desired time sequence for goal completion. Goals lower in the hierarchy should be completed before those higher up. The level and left to right relations between goals is arbitrary, only the parent/child tree relation is important. This relation provides a task precondition structure for goals. Task 2 in Figure 4.21 is a typical example. Although turning a microwave on has nothing to do with whether it has food or not in it, for this particular task, that action is the last that should be attempted. The student task model provides a general overview to task modelling whereas SCRs can be used as a focus for indicating problems in a domain and for supplying local task guidance.

The student task model does not presume to be an ITS 'student model' in the usual sense, only an aid in the generation of feedback. It is a mechanism to provide a basic view of where the student is, in terms of the tasks that have been defined for a domain.

Figure 4.22 presents how the task structures are fitted over the domain model. By using these two types of task information, both local and general task information can be defined and applied to the domain model.

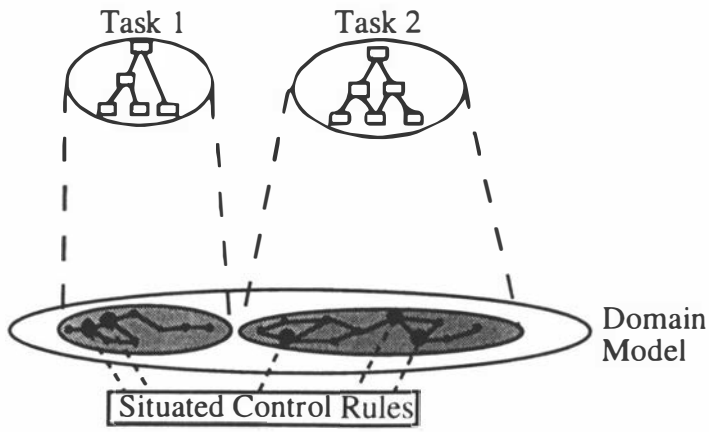


Figure 4.22 : Task structures over a domain.

The development of the domain and task models provides a knowledge representation basis for the definition of domain information. One problem with this definition is the size of the resulting domain knowledge that needs to be stored and manipulated within a teaching environment. Smith and Kemp (1995) examine some of the complexity issues of constructing POP tables and provide a comparison between POP tables and transition networks that shows that the POP representation is a more economical technique for domain representation. (Also see Appendix D.) One way to further increase the efficiency of model manipulation is the decomposition of large domains into independent sub-domains.

4.4 Sub-Domain Generation

When developing models for simulation, one of the major constraints on the overall development is the size of the model. This upper bound can be in terms of the physical space to store the model or the amount of processing that is necessary to manipulate large models. Effective utilisation of knowledge representation techniques can allow large models to be defined with a minimum of information stored but there will still be a limit to the size of a model beyond which it becomes unmanageable.

One solution to this problem is to limit the size of problems and models so that they can be easily and effectively manipulated. Although this may not be such a burden for small examples, this can be undesirable when

trying to model large domains as it limits the usefulness of the model by limiting it to smaller 'artificial' models.

Another solution is to break large domains into smaller sub-domains that are easily manipulated, with the condition that there is no detriment to the large model being represented. This is important when constructing paths through an environment, for example, the use of projection graphs to map areas of a domain. With large domains, the time required to generate these projection graphs increases exponentially (Drummond, 1989). To reduce the size of projection graphs that are generated, domains need to be split into sub-domains.

It is preferable that sub-domains are kept independent. Links between sub-domains add complexity to their projection and reduce the benefits of their decomposition. By removing any links between sub-domains, much smaller projections can be produced. This is comparable to Lansky's (1990) views on partitioning domains where the key idea is to partition a domain in such a way that domain properties are localised. That is, if a set of events is affected by a specific set of constraints, that set should be presented as a distinct region of activity.

This partitioning of domains is a difficult problem. Given a large domain, how can it be adequately split into appropriate sub-domains? This can depend on the skill of a domain expert in subdividing a domain description and on the nature of the domain itself. Manual decomposition of domains is one way to do this.

Manual sub-domain decomposition has been attempted and Figure 4.23 shows an example from a VCR domain.

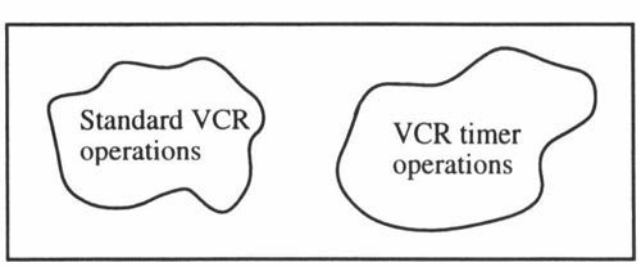


Figure 4.23 : VCR sub-domain separation.

Manual sub-domain decomposition may not always be desirable as the boundaries between sub-domains may not be clear cut. Also, manual decomposition may be inefficient or non-optimal. The use of automatic sub-domain generation would allow usable sub-domains to be defined and could be used to validate user-defined sub-domains. Before this happens, the requirements of a sub-domain need to be defined.

Sub-domains consist of a sub-set of conditions from a domain description. One method to develop independent sub-domains is to select operators whose preconditions can be enabled by operators currently in the sub-domain.

An example of two independent sub-domains can be seen in Figure 4.24.

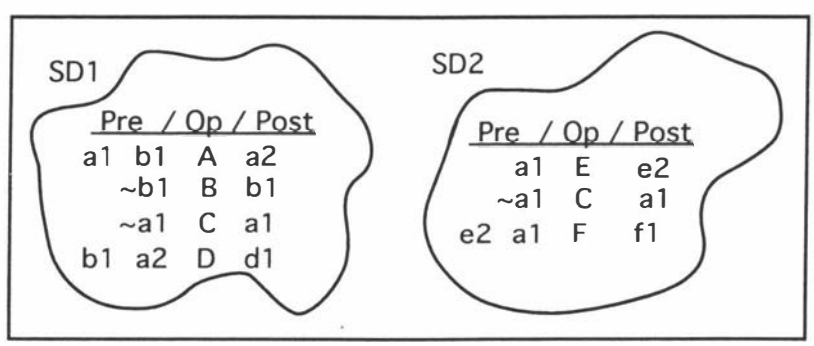


Figure 4.24 : Two independent sub-domains.

All the preconditions for operators in SD1 can be enabled by the use of other operators in SD1. Also note that the operators in the sub-domains do not have to be distinct, eg. operator C. This allows independent sub-domains to be defined without having to deal with the added complexity of connections between sub-domains.

The separation of domains into sub-domains had initially been attempted as a manual task where the domain expert selects areas of the domain that can be split into sub-domains. This is usually done by the identification of areas of functionality within a domain. These areas are defined as independent domains and can later be combined, as sub-domains, to allow the representation of some larger domain without the overheads of representing the whole domain. In device modelling, functionally independent components can easily be modelled as sub-domains.

Even when operators are common within sub-domains, with careful manual construction, sub-domains can be kept independent. For example, in a VCR domain four sub-domains have been developed: the basic operations of the VCR, the timer unit, the clock unit and the tuner unit. Figure 4.25 shows the overlapping operators in these domains and Figure 4.26 displays the final independent sub-domains that have been manually developed.

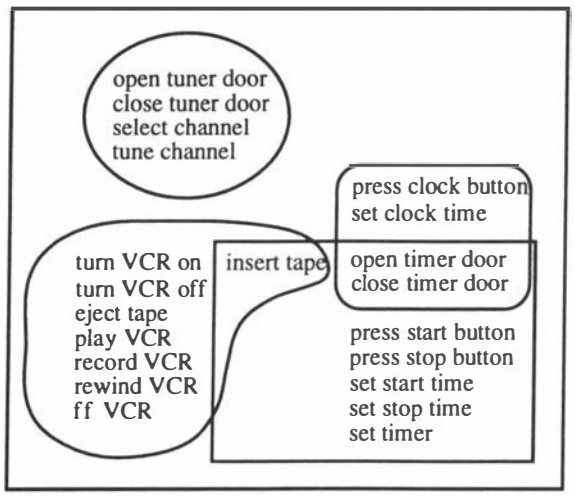


Figure 4.25 : The relations between VCR sub-domains.

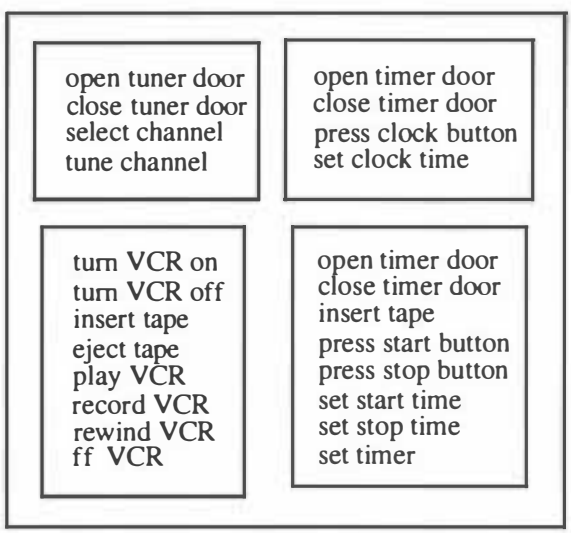


Figure 4.26 : Independent VCR sub-domains.

Although this breakdown by function seems adequate for this example, will it be useful in general and can such an approach be used to automate separation? For many domains there may not be clear-cut areas for sub-domain separation to be applied to. One problem with trying to automate

sub-domain partitioning is that the selection of areas to split is done using information about the domain in a certain context. Therefore, automated sub-domain generation may be impossible for all but a few well structured domains, i.e. domains where independent areas are already identified.

Whereas, automated generation of sub-domains based on the functionality of areas within domains seems impractical, what scope is there for automated generation from only the operators themselves? By ignoring the user view of sub-domains, can sub-domains be generated for internal system use only?

Sub-domains can be generated by grouping operators such that each sub-domain contains operators that enable the preconditions of the other operators, i.e. the operators make up an independent sub-domain. Using this method, many small sub-domains are generated, for example, using some operators from the VCR domain; see Figure 4.27.

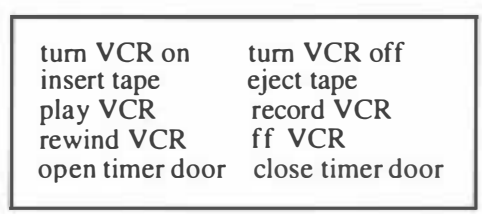


Figure 4.27 : Sample VCR domain.

This sample domain can be broken down into the independent sub-domains A-D as shown in Figure 4.28.

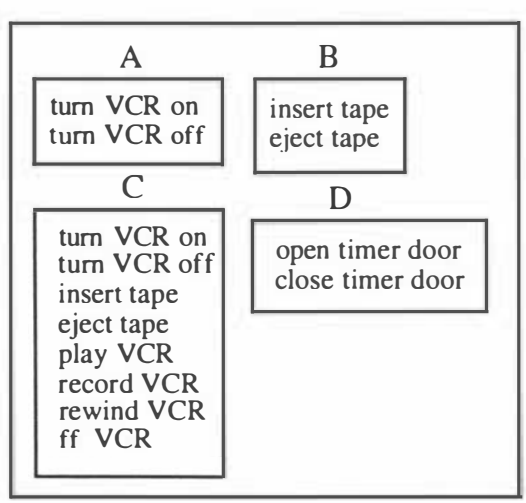


Figure 4.28 : Automatic generated VCR sub-domains.

Many small sub-domains are generated by this method. Although all these small sub-domains are valid, it may be beneficial to minimise them by combining them. In the example illustrated in Figure 4.28, sub-domains A and B are both subset domains of sub-domain C, thus it may be possible to discard A and B since the operators within them are covered by sub-domain C.

A program (Smith, 1994f) has been developed that takes a POP table representation and generates appropriate sub-domains. Output using a sub-set of operators from a VCR domain is shown in Figure 4.29.

```
All the sub-domains =

[turn_vcr_off, turn_vcr_on]
[turn_vcr_on, turn_vcr_off]
[close_timer_door, open_timer_door]
[open_timer_door, close_timer_door]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape]
[eject_tape, turn_vcr_off, off_insert_tape]
[turn_vcr_off, turn_vcr_on, on_insert_tape, eject_tape]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape, record_vcr]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape, rewind_vcr]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape, ff_vcr]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape, play_vcr]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape, record_vcr, stop_vcr]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape, rewind_vcr, stop_vcr]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape, ff_vcr, stop_vcr]
[eject_tape, turn_vcr_off, turn_vcr_on, on_insert_tape, play_vcr, stop_vcr]

Unique sub-domains =

[turn_vcr_off, turn_vcr_on]
[close_timer_door, open_timer_door]
[eject_tape, off_insert_tape, turn_vcr_off]
[eject_tape, on_insert_tape, turn_vcr_off, turn_vcr_on]
[eject_tape, on_insert_tape, record_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, on_insert_tape, rewind_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, ff_vcr, on_insert_tape, turn_vcr_off, turn_vcr_on]
[eject_tape, on_insert_tape, play_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, on_insert_tape, record_vcr, stop_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, on_insert_tape, rewind_vcr, stop_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, ff_vcr, on_insert_tape, stop_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, on_insert_tape, play_vcr, stop_vcr, turn_vcr_off, turn_vcr_on]

Minimised sub-domains =

[close_timer_door, open_timer_door]
[eject_tape, off_insert_tape, turn_vcr_off]
[eject_tape, on_insert_tape, record_vcr, stop_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, on_insert_tape, rewind_vcr, stop_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, ff_vcr, on_insert_tape, stop_vcr, turn_vcr_off, turn_vcr_on]
[eject_tape, on_insert_tape, play_vcr, stop_vcr, turn_vcr_off, turn_vcr_on]
```

Figure 4.29 : Generated sub-domains.

By removing the sub-set and non-unique sub-domains there is a considerable reduction, approximately 60% in the example illustrated, in the number of sub-domains that are generated.

4.5 Summary

The representation of knowledge is an integral part of most computer systems. In ITSs, it provides the context of the learning environment and the required information for the teaching process. As seen in this chapter, domain and task models can be represented using existing knowledge representation techniques. These allow the required information to be stored economically and in a form that is efficient to use.

For domain model representation, AI based techniques, plan nets and static axioms, have been applied with a modified STRIPS notation, the POP table. In the current work, this provides an efficient format for both a textual and graphical representation. Also from AI research, situated control rules have be refined to provide task oriented feedback and to supplement a basic task model.

The use of graphical techniques, based on a robust representation, can greatly enhance the ease of the domain definition process and provide a clearer view of the material that is being defined. This is extremely important when large scale domains are being developed. Plan nets are used as the graphical foundation for the package implemented in the current work. This will be described in Chapter 6.

The separation of domains into smaller sub-domains is an important consideration if large scale 'real world' models are to be developed. Manual and automatic sub-domain generation are two partitioning techniques that have been examined. The use of automatic techniques for sub-domain construction is especially relevant when domains consisting of independent operators are required. Automatic sub-domain generation has been shown to produce considerable savings when applied to certain domains. Also, the use of sub-domains to partition large domains can attempt to counter the huge computational overheads when these domains are being manipulated.

The representation of domain and task knowledge is the first stage in the development of an ITS. Next, teaching strategies that work with these representations are required. This includes the manipulation of the domain knowledge to provide feedback to the user and structure to a tutoring session. These ideas will be investigated in Chapter 5.

Chapter 5

Teaching Strategies

5.1 Introduction

Teaching is an extremely complex process. Not only must the subject matter to be taught be defined but also the responses of the student need to be considered. The view the student has of the teaching process is gained from the feedback that they receive during a teaching session. This feedback is their only guide to what they are currently doing and how successful their previous actions have been. Unfortunately, the type and amount of feedback that is required can vary from domain to domain and from student to student. To facilitate this within an ITS environment, the use of multiple teaching strategies within a tutoring system has become common place.

In this chapter the generation and use of feedback in an ITS environment will be examined and more specifically, domain and task feedback. Also the use of multiple teaching strategies and the switching between these strategies will be investigated.

5.2 Feedback

The use of feedback and guidance are major concerns when developing exploration based tutoring systems. Cox and Cumming (1990) note that for an effective and attractive environment for learning, there must be provision for advice and guidance. Providing feedback to the user in an

ITS is a delicate matter. Steinberg (1991) observes that the two main functions of feedback are to inform and to motivate. If too much feedback is provided, users may not have to think for themselves and may get the feeling that the environment is more of a static tutorial that they are being led through. If not enough feedback is provided, users may become frustrated as they may not see the results of their actions or may find themselves trapped in undesirable situations due to unnoticed mistakes. There are two major problems that are related to providing feedback, namely, when should it be presented to the user and how much should be presented.

Fundamental to instructional dialogues in interactive learning systems are the monitoring, feedback and corrective actions that are used to ensure that the learning/training process is progressing in the correct way (Barker, 1994). Depending on the tutorial environment, the timing of feedback can be very important for its effect on the user. *Instant feedback* and *delayed feedback* are two typical approaches that can be used.

Instant feedback is especially useful for positive feedback. When an operation has been completed correctly or the goal of a task has been reached, instant feedback can supply encouragement to the user and display a clear guide to their progress. This is especially the case when systems with graphical interfaces are being used. Both Mark and Greer's VCR tutor (1995) and Hollan, Hutchins and Weitzman's STEAMER (1984) provide instant graphical cues when actions are attempted. In the VCR tutor, currently invalid operations on the graphical view of a VCR model are greyed out after the completion of an action and in STEAMER, the state of the steam engine is instantly updated.

Anderson and Reiser (1985) contend that it is difficult for users to find and correct their own errors if they have made a mistake and then completed a series of steps based on that mistake until they have reached an impasse. The user may be overloaded with information when they try and backtrack to the source of the error and in the end may not learn the correct procedure. Instant feedback immediately indicates that an error has occurred. This allows the error to be pinpointed by the system and focuses the user's attention onto it.

Delayed feedback is more appropriate in an environment where more user experimentation is encouraged. Delaying error feedback gives the user the chance to discover their own error and provide their own solution. In ill-defined domains, where there may be many correct or adequate solutions, this allows the system to provide a more realistic environment. Delayed feedback works well within a scheme where the user can request help. Identified errors can be kept as a history by the system and relayed back to the user when they request addition feedback. Users can then be allowed to backtrack to the error points and to try alternative choices from these points. This backtracking and user experimentation can be useful if the goal of the system is to encourage the user to find multiple solutions to problems or to develop their own problem-solving strategies.

The amount of feedback that can be provided is dependent on the knowledge representation of the environment and the feedback approach that is being followed. The type of knowledge representation technique that is used to represent a domain within a system usually defines how its feedback is to be generated.

In small domains, common problems and errors can easily be identified and appropriate 'canned' text stored for presentation to the user when needed. This method is popular with systems that build libraries of common bugs and to a lesser extent case-based systems. For examples see BUGGY (Brown and Burton, 1978; Burton, 1982) and the LISP tutor (Anderson and Reiser, 1985). An advantage of explicitly storing feedback is that multiple levels of detail about a problem can be stored. This allows increasingly detailed explanations to be delivered.

Unfortunately, when domains become large, the amount of information that needs to be identified and stored makes this method of feedback generation unmanageable. Also, the information is brittle since context is not taken into account. A solution to this problem is to use a knowledge representation technique based on the relations between elements in a domain. As feedback is required, these relations can be examined and used as a basis for providing feedback. This approach can be generalised and based on rules defined by the structure of the knowledge representation technique involved and can be more flexible in the types and sizes of domains that can be modelled. One concern with this type of

feedback generation is that as the feedback is being generated dynamically, it may slow the response time of the system. This is a trade-off that must be considered based on the techniques involved and the computational resources available to the final system.

The amount of feedback that is provided to a user is a consideration similar to the previous feedback timing concern. If too much feedback is offered then the user will not be encouraged to think about the problem, but if too little feedback is supplied, users may become frustrated and give up. There is no general rule for supplying feedback as the amount necessary is dependent on the domain and the circumstances where the feedback is required.

5.3 Feedback Approaches

The overall control of feedback is determined by what feedback approach is currently being used. By combining feedback approaches, the amount and type of feedback can be changed to meet new user requirements. There are many different approaches that can be used when supplying feedback in an educational environment. Consideration must be given to the domain and the type of feedback that is desired when an instructional approach is to be decided on. Some useful approaches for guidance in an educational setting from the literature are: *optimal path, authoritarian, issue based feedback, device based feedback, use of primitive operations only* and the *use of high order operations.*

5.3.1 Optimal Path Approach

Many systems employ an optimal path guidance strategy where the student is forced to stay on an optimal path when completing a task in a domain. Actions initiated by the user off this path are met by appropriate feedback. ANDES (VanLehn, 1996) uses this approach. It stores every possible correct solution path to the problems it presents from the domain of classical Newtonian mechanics. If a student's entry is not along any of the correct solution paths, the student is given immediate negative feedback. In an interactive environment, this type of feedback is usually followed by help requests from the user in the form of *why* questions. A varied degree of feedback can be provided from an explicit description of

the next step to a hint message. These can either be to the current sub-goal or expressed in terms of the complete problem.

The main advantages in this approach are the ease of implementation, in terms of feedback messages and system control, that it does not allow users to get 'lost' in the simulation and that users do not develop false sub-goals that can slow down the process of acquiring correct procedures (McKendree, 1989). Unfortunately, there are some problems with this approach.

Firstly, it does not allow the students to experiment in the domain environment. This can be helpful to allow subjects to build their own mental models of the domain and to discover their own mistakes and correct them. Secondly, this environment does not resemble a 'real world', only an 'ideal world' setting. Reder and Klatzky (1994) maintain that as a general principle, having identical elements between the training and performance context facilitates learning transfer. This gives justification to the view that the teaching environment should be as accurate a simulation to the real environment as possible. However, this disadvantage is dependent on the domain involved. If an optimal path is the most important feature of the environment then a 'real world' system may not be desirable.

5.3.2 Authoritarian Approach

The authoritarian approach is similar to the optimal path approach, but differs in that the user starts with more freedom of movement within a domain but is quickly forced to stay on the path of an ideal student model (Specht, 1989). Users are focused onto an optimal type path after some initial experimentation within the environment. This method is used in Anderson and Reiser's LISP tutor (1985). Anderson, Boyle, Farrell and Reiser (1987) observed that more than half of the subjects' time in problem-solving sessions was actually spent exploring wrong paths or recovering from erroneous steps. Clearly this overhead is undesirable in a teaching environment. Kemp (1992) cites the work of Carroll and McKendree (1987) where a *penalty-free* exploration environment is developed. Here, a user may go on to see the impact of a poor decision but may then backtrack to the decision point to try alternative courses of actions.

5.3.3 Issue Based Approach

One problem when trying to provide feedback to a student is how to diagnose what feedback they need from their current actions. Burton and Brown (1982) note that in a domain where multiple skills are being used by a student, when an incorrect move has been made it is difficult to determine which skill is lacked by the student. This is because the system is indirect. It can only determine that the student did *not* make an appropriate move. Burton and Brown developed the paradigm of *Issues and Examples* to focus a coaching system on the relevant portions of a student's behaviour.

Important aspects of a domain are identified as a collection of *Issues* and are monitored by the system. The system uses two procedures to do this. An *Issue Recogniser* is used to build a student model from the student's actions, by determining whether a certain skill is present or missing and an *Issue Evaluator* is used to evaluate this model to ascertain if a student is weak in a certain issue. Once a weak issue has been identified, the system can provide an explanation of that issue, together with a better move to illustrate that issue. This provides the student with immediate feedback to a problem when it has been encountered.

5.3.4 Device Based Approach

The use of a device based (Mark, 1991) approach provides a flexible approach to system exploration and allows the user to complete problems using any possible solution to a fulfilled goal. This approach can be split into two sub-approaches where the first allows actions in any order as long as they are constructive towards the final goal and the second allows any possible action that is valid in the current environment.

The second approach closely resembles the real world, in that actions can usually be undone and allows more freedom for the user. Unfortunately, as noted in the optimal path approach, such unconstrained freedom can lead to incorrect assumptions being made about the domain and can thus be counter-productive. Also, destructive actions could be made without their immediate effect being realised by the user.

One possible solution would be to add an advice type feedback module that did not restrict but only commented on the actions currently being performed. This could make the user justify their current action to themselves in relation to the advice being given by the tutoring system.

The first device based approach gives a limited simulation of the real world but allows some experimentation within the bounds of positive actions in the simulation. Thus, only constructive actions could be attempted. Actions that have no effect on the environment (pointless actions) are ignored and no actions are allowed that cause an irreversible environment change that does not help the goals of the teaching session.

5.3.5 Primitive Operations Approach

Allowing users to apply only primitive operations forces them to be familiar with the environment at a low level of detail. In many cases this could be undesirable as keeping the learning in context to an overall goal could be extremely difficult. However, if based in an environment where there is a high level of internal functionality then manipulation of the domain at this level may be encouraged where higher order goals can be completed by explicit use of the primitive operations.

This approach forces the subject to complete all low level steps and to solve problems from basic goals upwards and allows users to see the whole problem domain piece by piece. Also, in such an environment, mastery can be easily used as an assessment measure. Unfortunately, this can limit the adaptability of the system as it does not allow for users learning to use higher order actions after mastering lower order sub-actions. Also it requires additional feedback when high order goals are completed.

5.3.6 High Order Operations Approach

A more expanded use of operators is seen in a feedback approach that allows the direct use of high order operations. If it is determined by the system, through current or past use, that the user has mastered sequences of lower ordered sub-actions, higher order actions can be added to the available action list to allow the user to work with higher level concepts. This allows the user to build primitive action relationships first and then

build a mental model of the higher order actions later. One problem with this approach is that it requires a complex control structure for the monitoring system to allow a mixture of usable and non-usable high level actions.

Once an approach to overall feedback has been chosen, consideration is needed to determine when and how much feedback is required. Anderson and Reiser (1985) note that there is considerable psychological evidence that humans learn better with immediate feedback. Similar views are also held by Bootzin, Bower, Zajonc and Hall (1986) and Feyock (1977). One problem with some types of immediate feedback is that this feedback may prompt the student into guessing the answer (VanLehn, 1996). VanLehn counters this by replacing minimal feedback with more focused feedback whenever it is likely that receiving minimal feedback would allow the student to guess the correct answer rather than use other methods of solving a problem.

Hill Jr. and Johnson (1994) maintain a tutoring system need not intervene in a heavy-handed fashion. It can serve as an information resource that the student can turn to for assistance as needed. Thus, depending on the current situation, different levels of feedback may be necessary. Error feedback can be provided based on what the learner has done or left undone. The advice that is provided, on user request, can be based on the same criteria as error feedback.

In the current research, an approach is being taken that is based on device-based feedback as described in section 5.3.4. This approach has been augmented with some of the more advantageous components of the other defined approaches.

Several complementary teaching strategies have been developed that can be used for instruction within a discovery learning environment. A full description of the teaching strategies will be detailed later in this chapter. However, before that is done, a look at the low level types of and responses to feedback is required.

5.4 Domain Feedback

The approach being taken in this research is a device-based approach in a discovery learning environment with the teaching of procedural skills. A domain model is used as the basis of the environment and is presented to the user. This is very similar to a real world device or the environment that is being modelled. The user can then perform actions within this environment and can witness the effects this has on the elements of the domain. Varied levels of feedback can be supplied, dependent on the domain being modelled and the demands of the user.

The actions that the user can perform in a domain can be roughly grouped into one of six action types: *correct*, *premature*, *destructive*, *redundant*, *impossible* and *unknown*.

Correct actions are valid in a domain and do not cause any detrimental effects on the modelled domain. These actions are reversible, so that any effects that the action has had on the current domain situation can be undone. Premature actions are ones which are valid in the domain but that have preconditions for enablement which are not currently satisfied. They are not necessarily premature in terms of the time sequence of a tutoring session, but are premature in regard to the action that is currently being attempted and its unsatisfied preconditions. Destructive actions cause effects on the domain that are not reversible. Redundant actions produce an effect that is already in place in the current situation of the domain. These can be repeated with no change to the current domain. Impossible actions are valid in a domain but due to some circumstance are no longer possible. These usually occur after a destructive action. If an action is tried that is not valid in the current domain then it is an unknown action. Only limited feedback can be supplied after an unknown action since the context of the action may be impossible to determine.

After an action is attempted, immediate feedback can be given to a user. Similar success feedback can be provided after correct, destructive and redundant actions. Success at completing an action and any other effects on the environment can be relayed to the user. After an impossible, unknown or premature action the environment is not changed but the user still needs to be notified that the action was not successful. Typical

user responses after these types of actions are *why* or *how* queries. Five general help commands are used to provide appropriate user requested feedback. These are *help*, *commands*, *hint*, *why* and *how*.

The *commands* command is the most basic general feedback command. It provides a list of all the valid actions in the current environment. A subset command to *commands* is the *help* command. It provides a list of only the currently valid actions in the environment. After this command, only the correct, destructive and redundant actions are listed.

The *hint* command is related to task completion in the environment and provides a suggested action for the next step towards a current goal. This will be discussed later in section 5.5.

The *why* command is used to provide a second level of feedback after an unsuccessful action or as justification after a successful action. Figure 5.1 shows typical feedback generated after a *why* command.

Feedback Type	Feedback Response
Why - after a correct action.	Action X_action added Y_conditions to the current situation. Action X_action removed Z_conditions from the current situation.
Why - after a destructive action.	Action X_action added Y_conditions to the current situation. Action X_action permanently removed Z_conditions.
Why - after a redundant action.	Action X_action added nothing to the situation. Action X_action removed nothing from the situation.
Why - after a premature action.	Action X_action is premature in the current domain situation. Action X_action conditions Y_conditions are not satisfied.
Why - after an unknown action.	Action X_action is not possible in this domain.

Figure 5.1 : Feedback from the *why* command.

The *how* command is typically used after a *why* command. It has a more limited scope as it can only really be used after an error action. After a successful action, the *how* command has no meaning. The responses for the three error action types can be seen in Figure 5.2.

Feedback Type	Feedback Response
How - after a premature action.	Condition Y_conditions is not satisfied. To satisfy condition Y_condition try action X_action.
How - after an unknown action.	Action X_action cannot be done in this domain.
How - after an impossible action.	Action X_action can now no longer be completed.

Figure 5.2 : Feedback from the *how* command.

The actual responses generated by the feedback operators described above are best seen in an example. Figure 5.3 shows part of a dialogue in a car maintenance domain. The task the user is currently trying to perform is to remove the tyre of a car. The dialogue has a pictorial view of the domain and has a text component that is in three parts. Firstly, the current situation in the environment is listed. Then a user command is described (in italics) and finally, the feedback response from the system that is appropriate for that action in the current circumstances is given.

The dialogue for steps (1) and (5) are the standard positive feedback messages that accompanies a correctly completed action. Step (2) shows a premature action where a valid but currently unenabled action is tried. Steps (3) and (4) are typical responses to a premature action. Firstly, the cause of the failed action is defined and then a possible solution to overcome the problem is provided.

Step (6) provides a reason after an action is correctly completed. It allows the user to see exactly what effect their last action had on the environment. Steps (7) and (8) are examples of the use of help and hint feedback.

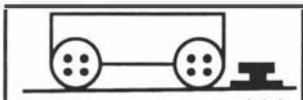
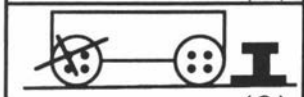
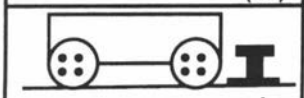
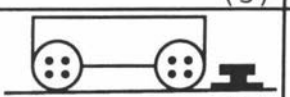
 (1)	[jack(available), wheel_brace(available)] Command? *use jack* Ok.
 (2)	[jack(up), wheel_brace(available)] Command? *remove nut 1* That cannot be done at the moment.
 (3)	[jack(up), wheel_brace(available)] Command? *why* The condition : car(on,jack) is not satisfied.
(4)	[jack(up), wheel_brace(available)] Command? *how* A step towards car(on,jack) is 'use jack'.
(5)	[jack(up), wheel_brace(available)] Command? *use jack* Ok.
 (6)	[jack(available), wheel_brace(available)] Command? *why* After 'use jack', jack(up) was removed from the situation and jack(available) was added.
(7)	[jack(available), wheel_brace(available)] Command? *help* Valid commands at this point are : 'place jack under car' 'store jack' 'store wheel brace' 'use jack'
(8)	[jack(available), wheel_brace(available)] Command? *hint* A desirable step at the moment would be : 'place jack under car'.

Figure 5.3 : Example of feedback dialogue in a car maintenance domain.

The help, commands, hint, why and how commands are the base level user requested feedback commands. They help the user at a local level in the exploration of an environment to gain some extra feedback from the system. More task oriented feedback is provided by task feedback.

5.5 Task Feedback

Basic level exploration in a discovery learning environment allows the user to explore a domain with minimal system intervention. However, when procedural skills are to be taught, it may be necessary to apply some type of task structure over the domain material. This task structure can be passive and act as a guide or reference structure for the topic within the

domain to be covered. Alternatively is may be active and force the user to meet appropriate goals in the domain or to complete tutor-defined tasks.

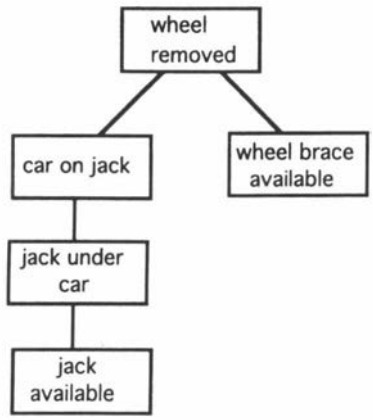


Figure 5.4 : Example task structure for a car maintenance domain.

As the teaching area focused on in the current work is based on using a discovery learning environment, only limited support for task feedback has been developed. A hierarchical task structure is overlaid over the domain and defines the relations between the different goals in a domain and their sub-goals. An example task structure can be seen in Figure 5.4.

This structure is used as the main component of the student task model (STM). Three feedback operators are provided to work with the STM. These are *task*, *try* and *hint*.

The *task* command is used to display the current state of the STM. The user may select this for two reasons. Firstly, to see their progress through the tasks set for the current domain or secondly, to aid in the selection of the next task to attempt.

Initial and subsequent tasks are selected automatically by the system once the previous task has been successfully completed. However, if the user wishes to complete an alternative task first, to change the focus of the session, this can be made using the *try* command. This command notifies the system that the user is now attempting a new task. Although this mechanism is laborious in a text version interface, it can be used effectively with a flexible graphical interface.

The *hint* command is used to allow the user to receive task-oriented feedback on the task that they are currently attempting. The *hint* command uses either the local task information provided by situated control rules or dynamically creates a projection network from the current situation to determine what the next most appropriate actions would be. In the current implementation, hint feedback is limited to the 'best' candidate operator from the current situation. The best candidate operator is selected by finding the shortest path to the current goal node in a projection graph. Therefore, a complete path of operators from the current situation to the final goal state is not provided to the user, only enough feedback is provided to enable them to move closer to the final goal.

When task-oriented feedback is to be applied, both situated control rules and projection networks can be used to augment the feedback that can be generated. As the user is navigating through a domain space, pre-defined SCRs act as beacons or signposts for encouraged actions. Especially when a particular task is to be completed, SCRs are usually defined for critical choice points in the domain space where it is important that either the user makes some choice or is made aware that certain actions may be more appropriate. As the SCRs are constructed before domain navigation begins, they are a useful source of quick reference feedback.

A similar feedback technique can be used with projection networks but with the drawback that projection networks must be generated dynamically at run-time. Although this generation is not as efficient as the use of SCRs, projection networks can be generated from any domain situation and are thus more flexible and do not have to be explicitly defined by a tutor before the tutoring session is started.

The effectiveness of using the projection networks to find valid paths between current situations and goal situations is dependent on the level of detail that has been defined in the task/sub-task model. If a detailed STM is provided, then smaller projection networks can be generated, while a limited STM may force the need for more extensive projection networks to be constructed.

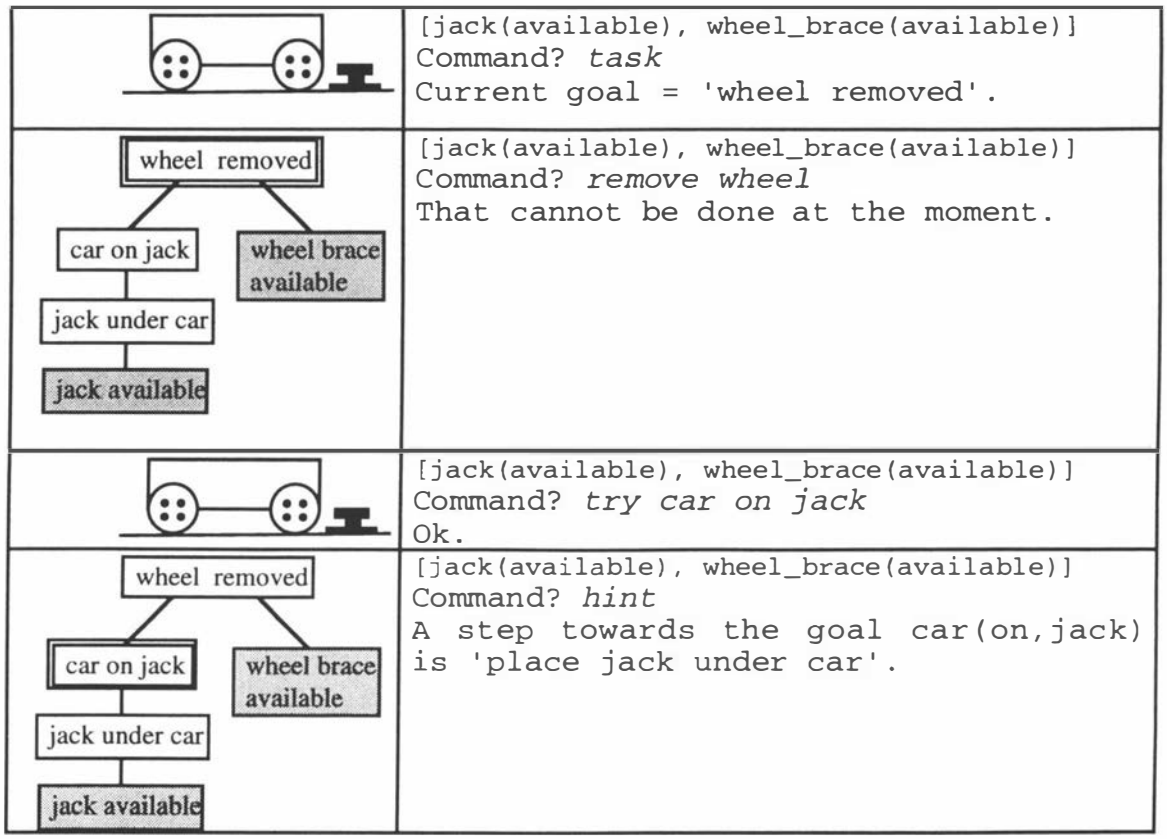


Figure 5.5 : Example dialogue of task feedback in a car maintenance domain.

As seen in Chapter 5, the size of generated projection networks is an important issue due to the computational overheads of their generation. The size of projection networks can be reduced if appropriate sub-goals can be found early in the projection process. This breakdown of goals to sub-goals is not essential but does increase the speed of feedback generation and improves the continuity of the tutorial session.

An example of the use of the three task oriented feedback operators can be seen in Figure 5.5. User commands are in italics, completed tasks are greyed out on the task model and the current task has a double border around it. The task, try and hint commands are task oriented feedback operators and are made available to help generate feedback in respect to the local context of the domain environment and a student task model. To guide the overall performance of the user and to focus the user to a particular learning method, teaching strategies are used to help enforce teaching methods.

5.6 Teaching Strategies

For an ITS to effectively achieve its aims, appropriate feedback needs to be provided for the user so that relevant information about the environment can be supplied. Typically this is done by the tutoring component of an ITS. This component is responsible for determining *what* information to give to the user, *how* that information should be conveyed and *when* it is appropriate to provide such information. The way that it does this is commonly referred to as its *teaching strategy*.

The teaching strategy is the way the tutoring system controls the overall interaction with the user. In ITS literature, the use of multiple teaching strategies is encouraged so that a variety of user learning styles can be provided. Kass (1989) maintains that a good tutoring system will have a variety of strategies it may employ to help a user. The tutoring component not only contains knowledge of these strategies, but must be able to determine the strategy most appropriate for a given situation.

Using multiple teaching strategies in an ITS can improve the flexibility and usefulness of the tutor. A tutoring system needs a variety of strategies in order to be able to be responsive to the users needs (Spensley, *et al.*, 1990). Spensley *et al's* DOMINIE is a tutoring system that operates in a number of domains and supports multiple teaching and assessment strategies. Spensley *et al* believe the potential for providing an adaptive program of instruction depends upon having a variety of presentation techniques to select from. Only with sufficient variety can the system respond to the cognitive needs of each individual user. Eight teaching and assessment strategies are utilised within DOMINIE:

(i) Cognitive apprenticeship - Cognitive skills can be taught similar to the way crafts are learned from experts. Learning is situated in a context similar to those in which the skills will be used (Lajoie and Lesgold, 1992). The apprentice begins by observing the expert and asking questions. As time passes the apprentice is allowed to perform small parts of the task. These parts increase in size until the apprentice is able to perform the whole task.

(ii) Successive refinement - A top-down approach to teaching. Topics are explained at increasingly detailed levels as the user's knowledge of the subject increases.

(iii) Discovery learning - A teacher sets up an environment for learning and the pupil explores it. The teacher can provide guidance when requested or if the user is in difficulty.

(iv) Discovery assessment - This approach follows discovery learning and when the tutor provides guidance, it is in the form of progressively more detailed hints which use an analogy with previous knowledge the user has acquired.

(v) Abstraction - Users are provided with a global view of the tasks which can be carried out, rather than teaching specific activities.

(vi) Socratic Diagnosis - The user is assumed to have a model of the domain with some flawed components. The system identifies the flaws and takes the user through a series of educational interactions so that the user realises they have an error in their model because it makes incorrect predictions. The system then provides the user with information to correct their mental model.

(vii) Practice - The tutor monitors the user performing a task. The tutor generates a plan of the task and compares it to the user's actions. If the user deviates from the plan the tutor can provide remedial feedback. This form of assessment follows cognitive apprenticeship.

(viii) Direct assessment - Assessment of users declarative knowledge using the traditional question and answer format. Typically done with multiple choice or form-fill in answers.

The choice of teaching strategy in DOMINIE is determined by the task that has been chosen and the user's interaction history. Also, if any material is to be re-taught, a different strategy from the original one chosen is selected. Spensley *et al* remark that the teaching strategies embodied in DOMINIE are fairly coarse-grained but this again is unavoidable with a modular, domain-independent system. The strategies had to be implemented as context-independent approaches to the general

knowledge structures. The way DOMINIE uses its different strategies is to allow it to change its strategy after each interaction with the user. Major (1993) notes that this meta-strategy is based on such factors as the need to achieve a balance between teaching and assessment, the appropriateness of a strategy for a domain, the user's previous success with a particular strategy and the user's preferences.

COCA (Co-operative Classroom Assistant) (Major, 1993) is a system that gives teachers control over domain material, teaching strategies and meta-teaching strategies. Strategy rules are partitioned into four sets. Each set corresponds to a separate heuristic decision. These are:

(i) Next concept to use.
(ii) Nature of the next interaction.
(iii) Content of the next interaction.
(iv) Response to the student.

This system allows teachers to define rules based on these decisions and when COCA teaches, these strategic decisions are cycled through. At the end of each cycle, meta-strategy decisions allow the teacher to modify these rules. This permits the strategies to be fine-tuned as a tutoring session is in progress.

COCA has been developed as a flexible system and can be used to define existing teaching strategies. Major (1993) discusses this flexibility in describing the strategies used by DOMINIE and WHY (Stevens, et al., 1982). Initial evaluation of COCA shows that it is best suited to describing systems that follow a teaching cycle in which explicit teaching decisions are made sequentially before interaction with the user and systems which use hierarchically-based domain representations are likely to be the easiest to reconstruct.

Following Major's work, RAPITS was developed by Woods and Warren (1995). They are interested in developing a practically useful ITS in a non-specific domain that can teach intelligently, i.e. adapt its style of tutoring to the user and the topic. Woods and Warren note that in COCA, a user rating is used to guide the meta-strategy, whereas it may be better to use the complete student history. This would allow faster feedback and make responses more sensitive. RAPITS has been developed to be an adaptive

system which can compare student and domain models and can automatically change its teaching style. It is based around the concept of an *electronic book*. Authors build teaching primitives in the pages of the books where the domain knowledge is created. Templates for various teaching styles are used when a new domain is to be defined. There are three main processes in the electronic book environment:

(i) Create an electronic book with each teaching primitive as an object on the page.
(ii) Generate the teaching strategies for the topic.
(iii) Create the meta-strategy to determine when the student history indicates a need for a change in teaching strategy.

Woods and Warren propose that even though there is no separation of the teaching strategies from the domain knowledge, the definition process has no significant detrimental effects if appropriate teaching templates are provided.

More recently, Srisethanil and Baker (1996) have developed an ITS in the field of engineering that employs four teaching styles, each of which support the learning of engineering subjects. ITS-Engineering is a domain independent ITS within a range of engineering subjects. The four teaching styles that it supports are:

(i) Instructor-oriented - The reproduction of knowledge is promoted by stimulating memory recall and the use of task repetition. An instructor presents information to the user and controls the pace of the session.

(ii) Guided discovery - Here, there is an emphasis on the production of new knowledge. Students are encouraged to think over information provided and to infer and discover further information.

(iii) User initiated - Students select a topic area of interest and set parameters for custom made problems in that area.

(iv) Exploratory - Students have the freedom to select which topics to learn without a specific sequence. This enables a student to look over the contents of a subject at their own pace.

These different teaching styles enable ITS-Engineering to deliver adaptive instruction reflecting a student's preferences, subject matter and the type of learning outcome to be achieved.

The four systems described above all allow the use of multiple teaching strategies. However, the use of domain-independent strategies can be dependent on the type of teaching that is required. If it is to be a truly general tutoring system, then promoting independent teaching strategies will allow for flexibility in the re-use of the teaching information over multiple systems.

Separation of the teaching strategies from the domain information follows the domain and task separation work described in Kemp (1995) and Kemp and Smith (1994b). Several teaching strategies have been developed that restrict user navigation in a domain and augment standard feedback to cater to user requirements.

The separation of the teaching strategies from the domain descriptions promotes the re-use of strategies over a range of domains and facilities their possible re-use in other ITSs. This separation is similar to the separation of the knowledge representation technique from the domain material. If the teaching strategies are developed so that they only use the domain information as defined by the knowledge representation technique, then the independence of the knowledge representation enables the teaching strategies to also be independent.

This is consistent with one of the aims of this research which is to keep the overall system independent of the domains that are to be modelled. To facilitate this, all feedback must be able to be generally inferred from the relations between elements of the domain knowledge.

The teaching strategies and knowledge representation have a close association as the teaching strategies are based directly on the structure of the knowledge representation. This can be seen in Figure 5.6 where the tutoring component of a system is the interface between the teaching strategies and the defined domain knowledge. The teaching strategies and the knowledge representation technique must be described in general terms if they are to form a general structure for building different ITSs.

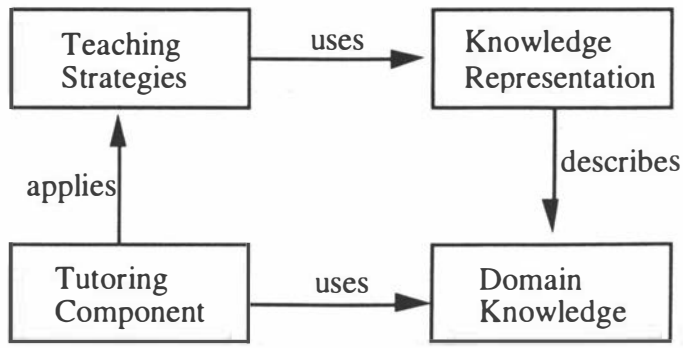


Figure 5.6 : Teaching strategy and knowledge representation association.

Keeping the teaching strategies independent from the domain material, with formalised links to the tutoring component, make it possible for new strategies to be added as needed. New strategies need only be based on the existing knowledge representation model and be connected to the tutoring component.

The specific environment for this research is a guided discovery learning environment. Two general strategies have been identified for use in this environment, namely, *domain exploration* and *task related exploration*.

Domain exploration is a general strategy that allows users to navigate their own way around the domain environment. The user is only constrained by the physical properties of the actual domain. This is a guidance-free 'free roam' in the environment. This strategy allows the user to get a feel for the domain by exploring the environment and finding its limits, i.e. what can and cannot be done.

Feedback for this strategy is generated directly from the POP tables. For example, premature actions can be identified by operators that have preconditions that are not currently satisfied and successful operations can be indicated and their postconditions added to the current situation of the domain. No guidance is given unless the user expressly requests help to complete an operation.

Task related exploration makes use of the pre-generated situation control rules (SCRs), dynamically generated projection networks and the student task model (STM) to provide task-oriented feedback. Task related exploration is divided into two different strategies. The first is *task-guided*

exploration which is similar to the domain exploration 'free roam' but with added overall goal guidance as defined by the student task model. The user has a task to complete but is still given freedom to find their own path through the environment. On request, task related hints can be generated. These are referenced from the STM or SCRs, if they are present. If not, a projection network can be generated and used to find a path, with the associated valid actions, from the current situation to the next desirable sub-goal location.

The other strategy is *task enforced exploration* where SCRs are used to focus the user's path through the domain environment. Three levels of SCR oriented guidance have been defined, ranging from simple advice to enforced suggestions. These are *advice, limited* and *extreme* guidance.

Advice guidance is the least strict task strategy and uses SCRs to warn the user about critical choice points when they are encountered. Only a warning is provided and the user may choose to ignore this. If requested, alternative routes can be suggested by the system.

The second mini-strategy is *limited guidance.* In this strategy, users are forced to take the paths indicated by SCRs when they are reached. If the current situation has no applicable SCRs, then the user is free to explore. Only when a critical choice point is encountered is the user required to follow a path indicated by a SCR.

Extreme guidance is the final SCR related strategy and forces the user to follow an optimal path to the current goal. Where possible, SCRs are used to guide the user but where none exist, projection networks are generated to provide feedback and to validate user actions.

These strategies have been summarised in Figure 5.7.

Strategy	Summary
Domain Exploration	• Free roam. • No guidance unless requested.
Task Related Exploration	• Free roam. • No guidance unless requested. • Hints provided using SCRs and localised projection networks.
Task Enforced Exploration	• Advice guidance. Use SCRs to warn users at critical choice points. • Limited guidance. Enforce SCRs at critical choice points. • Extreme guidance. Enforce an optimal route using SCRs and localised projection networks.

Figure 5.7 : Teaching strategy summary.

These strategies can be used in a variety of combinations by a tutor to structure a learning course in a domain. For example, initially users may be given a free roam session to familiarise themselves with the domain environment. Then, a session with extreme guidance could be used to demonstrate the correct procedure and finally, a session with advice guidance could be used to allow users to find their own alternative solutions to problems while having potential hazards in the domain highlighted by the system.

Providing multiple teaching strategies allows flexibility when structuring a teaching program in a discovery environment. The four strategies above allow the advantages of discovery learning to be used along with the benefits of having guidance during an exploration session. In relation to the feedback approaches discussed earlier, a discovery/device based environment with components of the free-roam, optimal path and authoritarian approaches can be provided.

Although the nature of the teaching strategies have been defined, their selection at run-time in a tutoring system must also be determined.

5.7 Teaching Strategy Selection

Selection of the appropriate teaching strategy for a particular user is a difficult task. Several ITSs use complex student modelling and diagnosis techniques to try to provide an accurate view of the user and to allow teaching strategies to be modified accordingly. Examples of this can be seen in the previously described systems DOMINIE (Spensley, *et al.*, 1990) and RAPITS (Woods and Warren, 1995), whereas ITS-Engineering (Srisethanil and Baker, 1996) switches its teaching styles based on the student's preference, the learning outcome required and the subject matter.

The use of an accurate student model can greatly enhance the tutoring process. It can be used to determine gaps in the users knowledge and note which teaching strategies the user prefers, or seems to understand better, and react accordingly. If a user is having problems learning with one strategy, the system can automatically change to an alternative strategy, especially one for which the user has previously shown a preference.

Automatic selection of teaching strategies is the most desirable of the selection techniques that can be used. This is the way a human tutor reacts to a user when they are having trouble understanding some concept. Typically, the same information is presented again in a slightly different form. This allows alternative feedback to be generated about the same problem and hopefully, ease the learning process.

Unfortunately, the construction, use and maintenance of these student models is extremely difficult. Roberts (1993) maintains that the definition of the student model and the subsequent interpretation of user actions with respect to that model becomes a daunting problem. It is also a non-trivial task for a student model to be verified as it is a model of some part of the users current understanding. Also, to make a model that is detailed enough to be useful requires a large amount of work. Muhlhauser (1990) observes that the enormous complexity of a human mind makes a good model hard to build and can lead to a lack of general applicability of the ITS developed.

Also, the development of a student model is not a finite process. It must be continually updated and maintained so that it can refine and adjust its

model in order to present a more accurate view of the user. This can be done dynamically through a tutoring session, which can increase the computational requirements of the system, or after a session, which requires recording the session and extensive post-session processing.

When a student model is being used in a tutoring session, the system will be continually comparing the model to the current actions of the user. This requires a comprehensive diagnostic and monitoring component for the system which may be difficult to keep independent from the domain material. This is particularly a problem when alternative types of user interfaces are to be developed, e.g. textual or graphical interfaces. In this case, providing a general purpose monitoring system may be impossible.

If a student model is not to be used then alternative approaches for teaching strategy selection must be found. One approach is that of user selection. Before a tutoring session has commenced, either a tutor can select an appropriate strategy for a user (for example, based on previous performance) or a user may select his/her own strategy of preference. Once a selection has been made, this strategy is used throughout the remainder of the tutoring session. If the user wishes to change the strategy, the current session is exited, a new strategy selected and the session recommenced.

A major benefit of this approach is that the huge overhead of development and maintenance of a complex student model is removed from the tutoring system development process. Also, it allows the user to control how they learn and can encourage interactive tutor supervision between a human tutor, the system and the user. Another benefit is that when one teaching strategy is selected, the teaching method is focused for that session. This allows the user to concentrate on the task at hand and they do not have to contend with alternating teaching styles, which may be distracting.

There are two main disadvantages with this approach. Firstly, the tutoring system is not totally independent. Some user input is required before a teaching session can begin. This may only be undesirable in some domain or particular problem-solving areas where a totally independent system may be required. Secondly, once a session has been started, the teaching strategy cannot be changed until the session is terminated. A

possible solution to this would be to allow sets of similar strategies to be grouped and allow the user to interactively upgrade the current strategy to a different one in its strategy set. This may be necessary if different teaching strategies are associated with different types of user interfaces. For example, all the strategies involving a graphical interface may be grouped, as textual based strategies would not be appropriate for the same session and vice versa.

5.8 Summary

The generation and presentation of feedback are two fundamental issues that need to be considered in the development of any teaching system. Basing the feedback around the domain and task models provides a firm foundation for its generation and use by a tutoring system. Linking multiple teaching strategies to these models can allow a flexible teaching structure for the tutoring component of an ITS.

In this chapter some of the issues involved with the use of complex student models and how these reflect on the use and selection of teaching strategies have been discussed. Allowing user selection of teaching strategies can provide an adequate mechanism for a discovery learning environment.

In the next chapter, the implementation of some of the theories discussed in this and the previous chapter will be described. The current system, TANDEM (Task ANd Domain Environment Model), provides an environment for the definition of domain and task knowledge and their application in a computer based tutoring environment.

Chapter 6

TANDEM

6.1 Introduction

The development of an environment to aid in the authoring of ITSs is one of the aims of the current research. This chapter presents TANDEM (Task ANd Domain Environment Model) (Figure 6.1), the implementational component of this thesis which is written in the language Prolog, with graphical support provided by the Macintosh graphics toolbox from within LPA MacProlog32 (Johns, 1994).

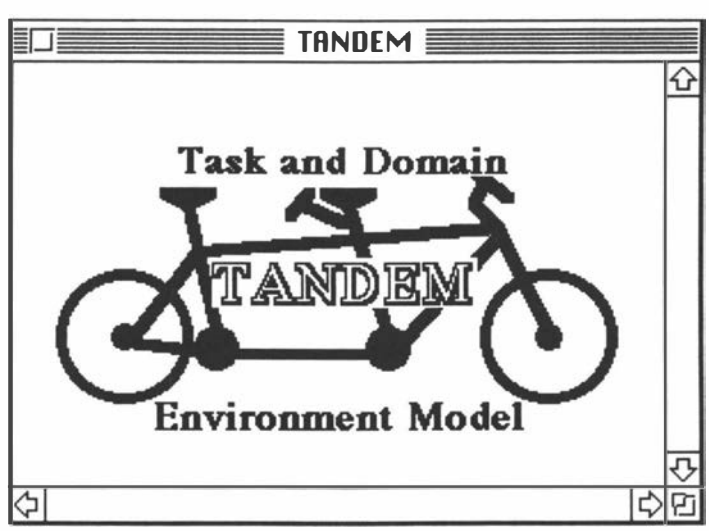


Figure 6.1 : TANDEM welcome screen.

Although a significant part of TANDEM is implementation based, it also encompasses a methodology for ITS construction. This will be the first

topic discussed in this chapter, followed by the two major components of the system, namely, the Domain Construction Environment (DCE) and the General Domain Interpreter (GDI).

The DCE is a direct manipulation environment for domain and task model definition. From graphical descriptions, domain and user interface code is generated to be applied with TANDEM's tutoring engine, the GDI. To promote reuse, the GDI is kept domain and platform independent by the use of standardisation layers between the different components of TANDEM.

The current implementation is very much a prototype system. It is presented as a physical example of the theory described in the previous chapters. At the present time, it has been completed to a point that enables the major components of TANDEM to be used, three examples of which can be seen in Chapter 7. However, not all of its components are fully functional. This includes the incorporation of sub-domain definition techniques and their automatic generation as described in Chapter 4. Although provision is made in TANDEM for sub-domain usage, their actual utilisation is not currently supported.

As will be discussed in section 6.2, TANDEM encompasses three major areas: domain, task and interface components. However, the explicit construction of domain interfaces is outside the scope of the current work. As TANDEM is a domain independent environment, the cost of this independence is that it is difficult to provide a generic interface for context dependent domain interfaces. What is provided is an interface template for each domain defined which allows custom interfaces to be linked to the interface side of TANDEM. This template generation and how it is kept independent from the domain definition is described further in sections 6.3.7 and 6.4.1 of this chapter and Appendix C.

6.2 TANDEM (Task ANd Domain Environment Model)

TANDEM is the environment for the definition, testing and production of procedural skill based tutoring systems. Its basic methodology can be split into three distinct components, namely a domain environment definition tool (the domain construction environment (DCE)), a domain

independent tutoring engine (the general domain interpreter (GDI)) and an interface development component. The use of TANDEM by an author to develop a tutoring system can be viewed, at a basic level, as a sequential flow through these components (Figure 6.2).

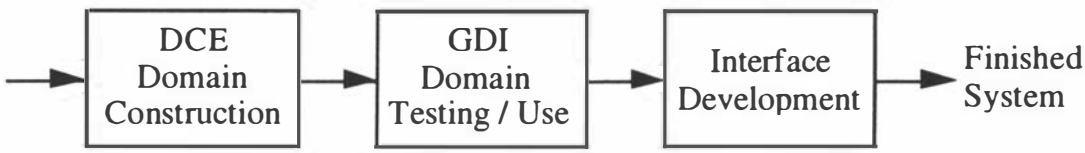


Figure 6.2 : Basic authoring sequence.

Unfortunately, the progression through the development process is not so straight-forward. Due to the nature of domain definition, an interactive prototyping approach is more appropriate.

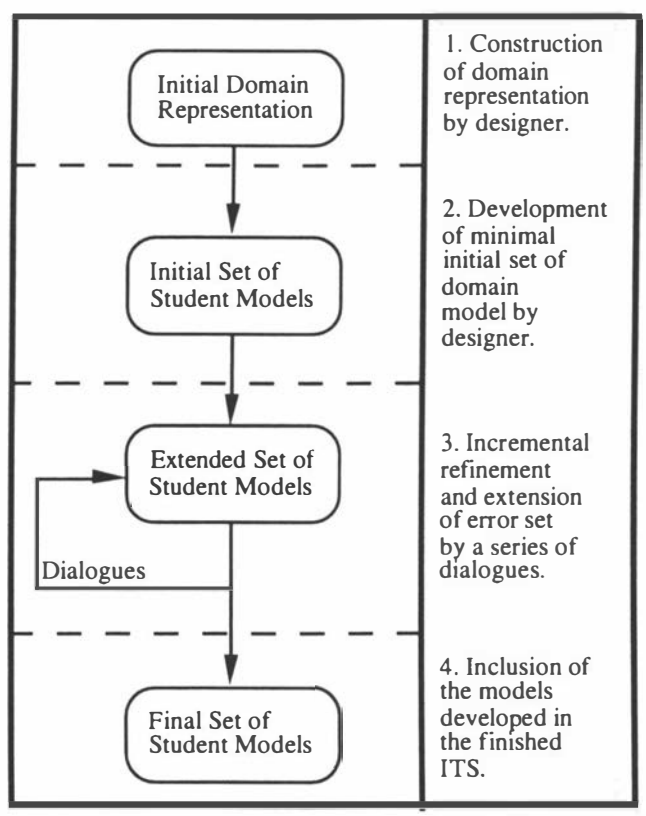


Figure 6.3 : The Natural Laboratory; from Cerri, Cheli & McIntyre (1992).

The definition process in TANDEM is primarily concerned with the construction of models, eg. domain and task models. In this, it is similar to the *Natural Laboratory* methodology as used by Cerri, Cheli and McIntyre (1992) in their development of student models for the Nobile

project. The objective of this methodology is the incremental construction and refinement of student models. The basic steps to be performed can be seen in Figure 6.3. An initial model is defined and then evaluated. Incremental refinement is then used to expand the models until a final set of models have been developed.

In line with the idea of incremental construction and refinement of models is the work discussed by Kemp (1995). He presents some preliminary work in the area of a procedural task tutor design system and provides an automated scheme for developing a procedural task tutor (Figure 6.4).

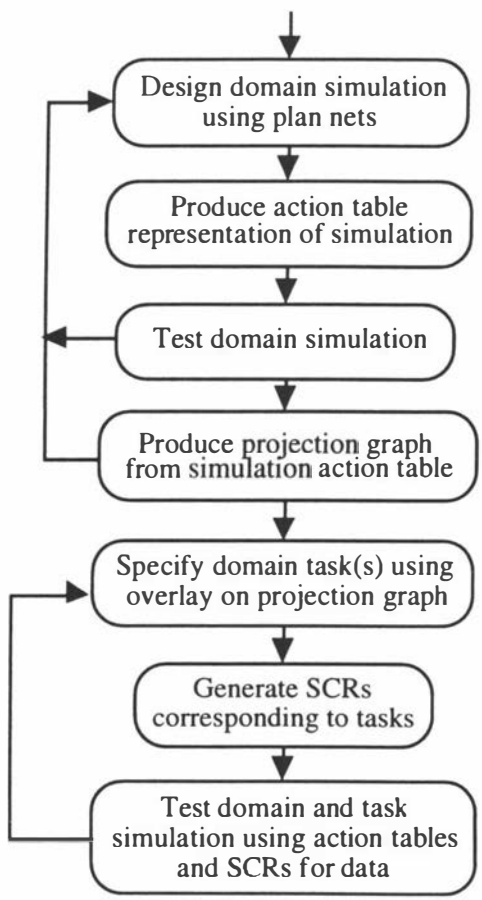


Figure 6.4 : Automated scheme for developing a procedural task tutor; from Kemp (1995).

The domain expert defines the behaviour of the environment and then incrementally tests and modifies the design until an acceptable model has been constructed.

Kemp's scheme continues his focus on the separation of domain and task, with the first three processes in his automated scheme concerned with the domain model and the next three concerned with developing the task model. The final process is the testing of both the domain and the task for model acceptance by the author.

Although this scheme was developed in conjunction with several pre-TANDEM implementations (Smith, 1994a; Smith, 1994b; Smith, 1994c; Smith, 1994d; Smith, 1994e), a more complex scheme is necessary for the current TANDEM implementation (Figures 6.6-6.8). This scheme has been developed to allow automated validation of portions of the domain representation. Also the incorporation of addition feedback allows the developer a complete construction environment. Domain and task separation are again emphasised and an interface definition section has been included. Figure 6.5 is a high level view of the process flow for using TANDEM.

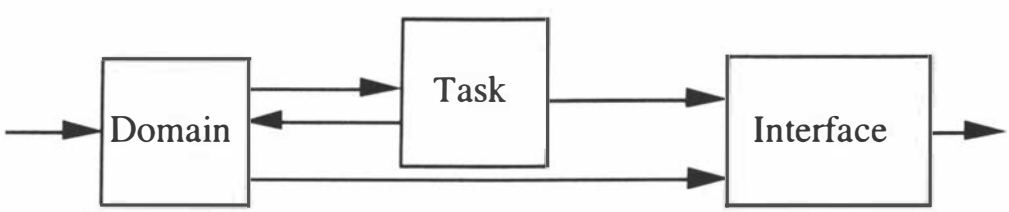


Figure 6.5 : High level view of the TANDEM methodology.

Each of the components of Figure 6.5 can be decomposed into its own diagram describing domain, task and interface components of the TANDEM methodology (Figures 6.6-6.8). In these figures, the boxes represent user processes while the bold text items are performed by the system.

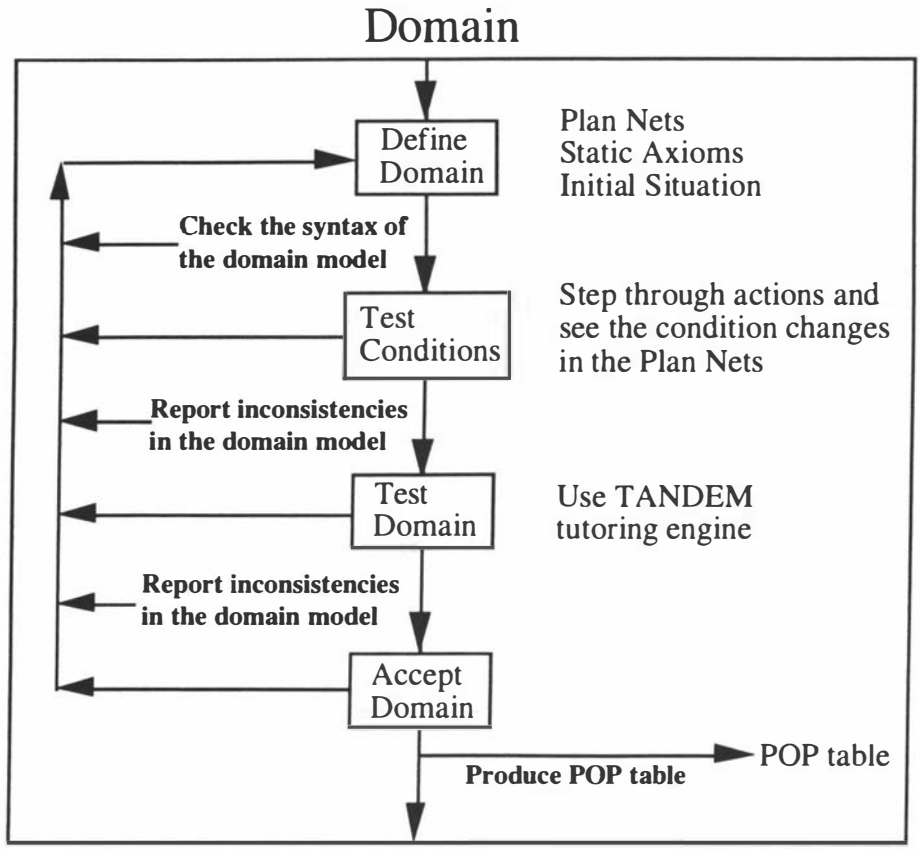


Figure 6.6 : Domain component of TANDEM.

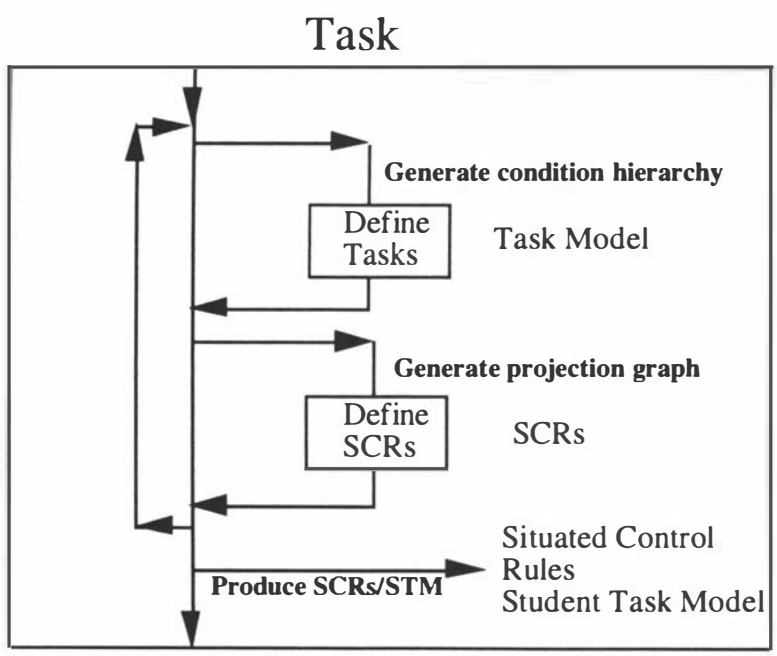


Figure 6.7 : Task component of TANDEM.

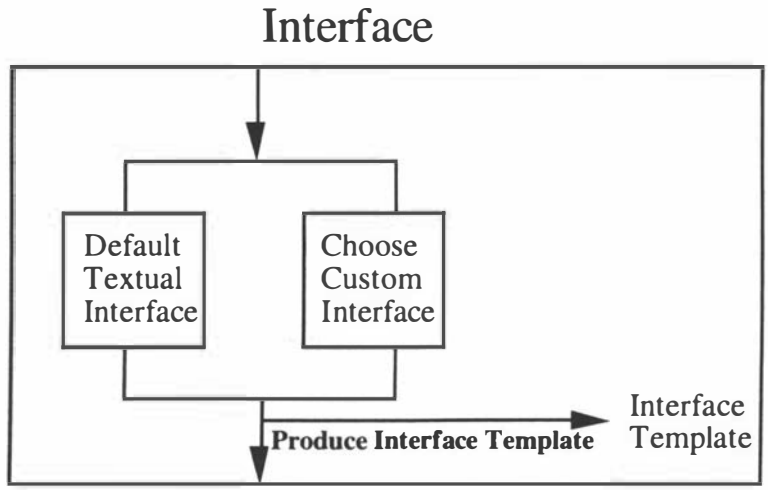


Figure 6.8 : Interface component of TANDEM.

Rapid prototyping was used as the base methodology for the TANDEM implementation since, although the features and components of TANDEM had been defined, the usability of the environment was relatively unspecified. Prototyping is a useful technique for software development when system requirements are unclear (Humphrey, 1995, Sommerville, 1996; van Vliet, 1993). Figure 6.9 shows the TANDEM implementation cycle. Through this cycle, the components of TANDEM were iteratively re-implemented. This was to improve usability and flexibility in the domain model definition process in terms of ease of use and model correctness.

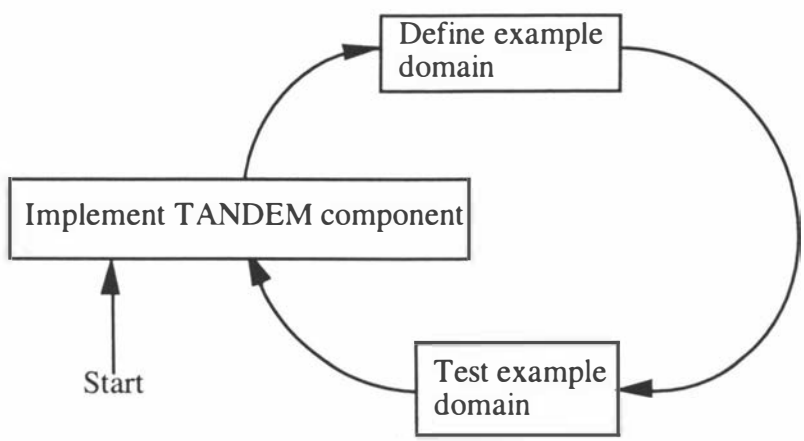


Figure 6.9 : TANDEM implementation cycle.

The TANDEM software environment was developed on a Power Macintosh 7100/66AV, configured with 16M of RAM, a 500M hard disk and MacOS version 7.1.2 (Apple, 1994). TANDEM is currently

implemented using LPA MacProlog32. Initial work had been completed on UNIX workstations using Poplog Prolog (Integral Solutions, 1995). Unfortunately, the available versions of Poplog and subsequent versions of SICStus Prolog (Carlsson and Widen, 1993) on the UNIX platform did not provide the graphical toolkits that would be necessary for the graphical TANDEM implementation.

LPA MacProlog32 combines advanced programming techniques with a user-friendly interface and enhanced graphics of the Apple Macintosh, making LPA MacProlog32 a rich and sophisticated programming environment. Also, LPA MacProlog32 is compatible with LPA Prolog for Windows (LPA, 1994) and Quintus Prolog (Quintus, 1996) on the UNIX platform, making is possible to port the system to other platforms if desired.

One question that can arise in the choice of programming language is that why use a high-level programming language and not an *end-user* programming tool? End-user programming tools, like HyperCard on the Macintosh have been used for CAI development and can reduce the development time for system construction. Unfortunately, end-user environments are usually too restrictive in certain areas, especially program control and interface construction (Ngan, 1992). In an environment where a variety of different domains are to be modelled, generality in interface approaches for possible interfaces is very important. Also, selecting a language that has implementations across platforms allows greater flexibility if unforeseen circumstances require a change of platform.

6.3 DCE (Domain Construction Environment)

The DCE is the environment where authors define the domain information that comprises the domain and task models. There are five author oriented processes that take place in the DCE, namely, domain definition using plan nets, static axiom definition, situated control rule construction by identifying paths through projections in the domain space, task definition and testing of the defined models. Other processes are automated by TANDEM to ease the authoring process. These include the generation of code from the graphical descriptions, syntactic

validation of the POP tables, POP sub-domain generation and domain interface template generation. The separation of processes into these groups becomes necessary as only the author can do the context sensitive tasks. Thus the processes the system handles are the domain independent ones.

This chapter will illustrate, with examples from the current TANDEM implementation, the author oriented processes, the model definition and testing, and then the system oriented processes, the POP table generation, POP table validation and the domain interface generation.

6.3.1 The DCE Interface

The DCE has been designed as a direct manipulation interface to allow domains to be defined and edited graphically. It supports multiple windows and as a domain is decomposed into sub-domains, a new window is opened for each sub-domain. Figure 6.10 shows the various parts of the DCE, which include:

- DCE option menu to give the author access to system commands.
- Drawing window for graphical domain definition.
- DCE tools for manipulation of on screen objects.
- Zoom box for a reduced view of the current window.
- A set of minimised windows, which allows a number of relevant windows to remain on screen.

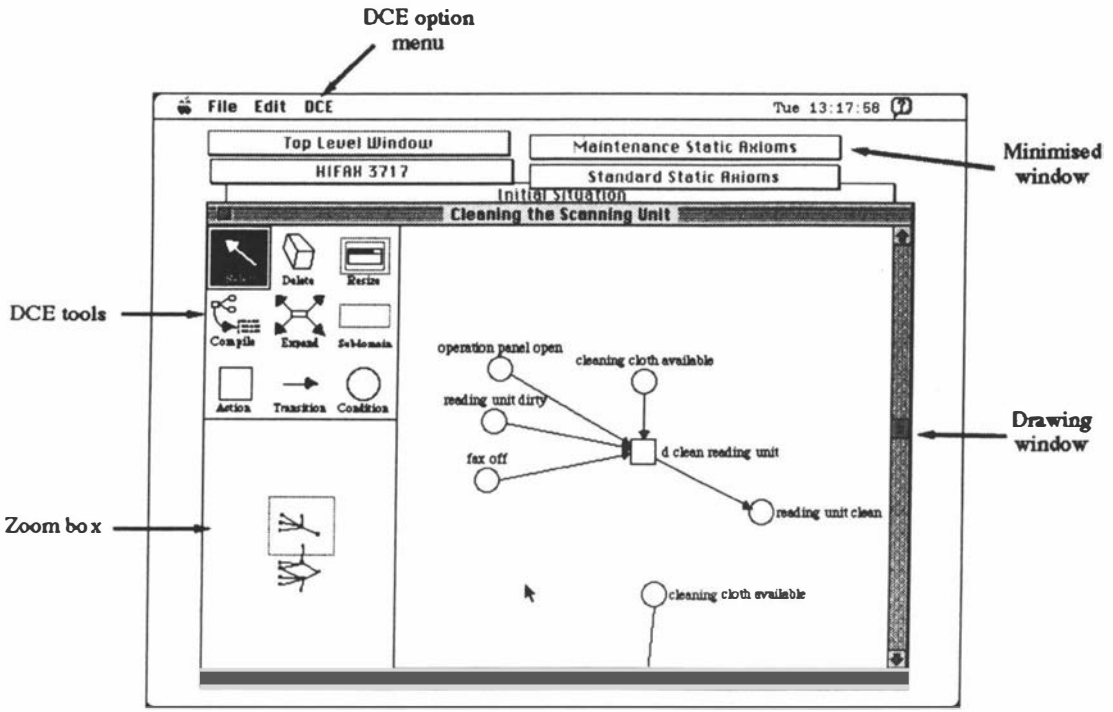


Figure 6.10 : The DCE environment.

Input in the DCE is by use of the one-button Macintosh mouse and a standard keyboard. The term *pointer* is used in this chapter to define the cursor displayed on the screen that tracks the mouse movements. As Macintoshes only use a one-button mouse, a combination of mouse presses and command key presses are used to increase the functionality of the mouse. Examples of this will be described where appropriate.

The DCE tools used in this environment can be separated into three different types. Firstly, there are the object manipulation tools: *select*, *delete* and *expand* (Figure 6.11).

Figure 6.11 : DCE object manipulation tools.

These tools are used to manipulate on screen objects in the DCE environment. The *select* tool allows the selection of an object currently at the pointers' location. It can also be used in a typical Macintosh fashion to

move selected objects and to drag-select on multiple objects. The *delete* tool is used to delete single and groups of selected objects. The *expand* tool is used exclusively to decompose sub-domain objects (see below).

Secondly, there are the environment manipulation tools *compile* and *resize* (Figure 6.12).

Figure 6.12 : DCE environment manipulation tools.

The *compile* tool is used to compile the currently defined domain from the plan net representation into a POP table representation. The *resize* tool is used to resize the canvas dimensions of the current drawing window. The current drawing window is only a view port to the whole window canvas. The canvas can be re-sized to allow more window real-estate or to allow it to be reduced.

The final tool type is the object tools (Figure 6.13). These tools are used to add plan net components in the drawing window. Actions and conditions are placed and labelled at the current pointer location. Transitions can then be drawn between actions and conditions as necessary. Labelled subdomains can be placed in the drawing window and decomposed into their own window using the expand tool.

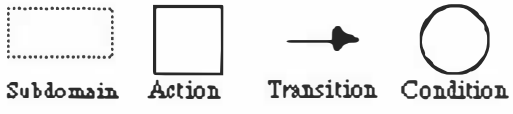


Figure 6.13 : DCE object tools.

All the tools in the DCE are highlighted in the tool bar when selected and the current tool can easily be determined as the pointer changes shape depending on which tool is currently selected, eg. a square for the *action* tool or a circle for the *condition* tool (Figure 6.14).

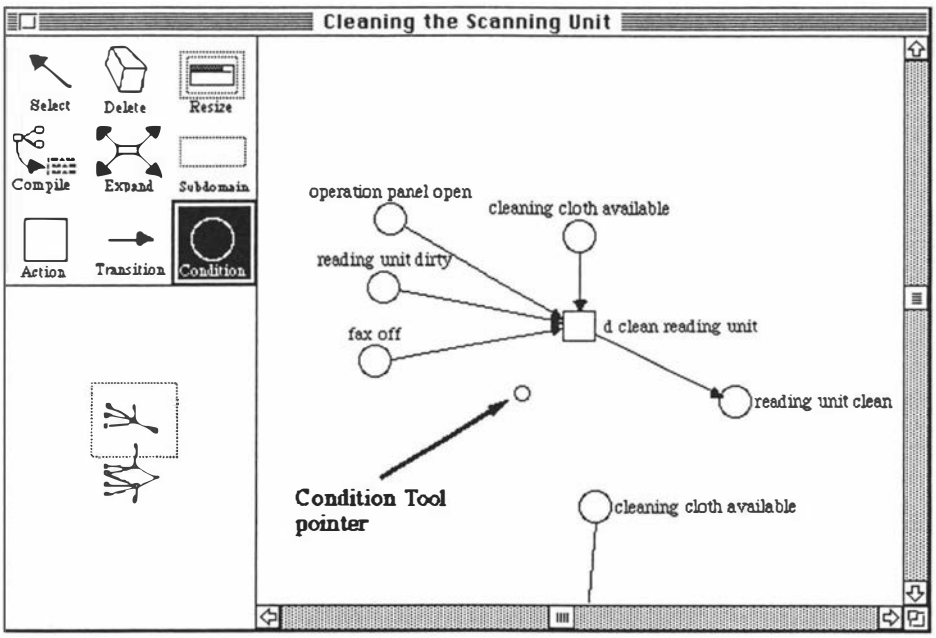


Figure 6.14 : Pointer mode change.

The DCE menu (Figure 6.15) provides the author with access to DCE system commands: loading and saving of domain descriptions, defining tasks, testing the domain description (see later in this chapter) and a windows menu (Figure 6.16) to allow easy navigation between windows.

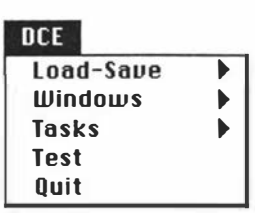


Figure 6.15 : The DCE command menu.

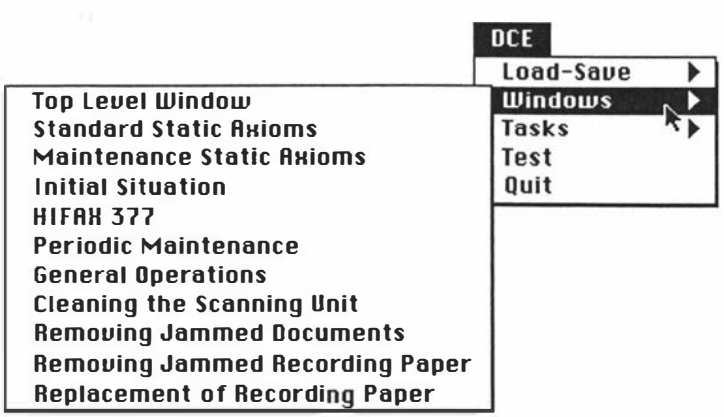


Figure 6.16 : The DCE windows menu.

6.3.2 Defining the Domain Model

The domain model is constructed graphically in a number of windows, some of which allow decomposition so that the user can separate areas of functionality. There are five main windows that user can use to define the domain behaviour which can be compiled into a POP table. These are the top level window, sub-windows, initial situation, standard static axiom and maintenance static axiom windows. Throughout the descriptions of TANDEM's features, a fax machine maintenance domain will be used for examples.

The *top level window* is the starting point for domain model definition. The only objects that can be placed at this level are decomposable sub-domain boxes. This forces the user to identify at least one level of decomposition that can be used later as the basis for sub-domain generation. For device modelling, areas of functionally in the domain can be identified and make good choices for sub-domain definition. Figure 6.17 shows the format of a typical top level window.

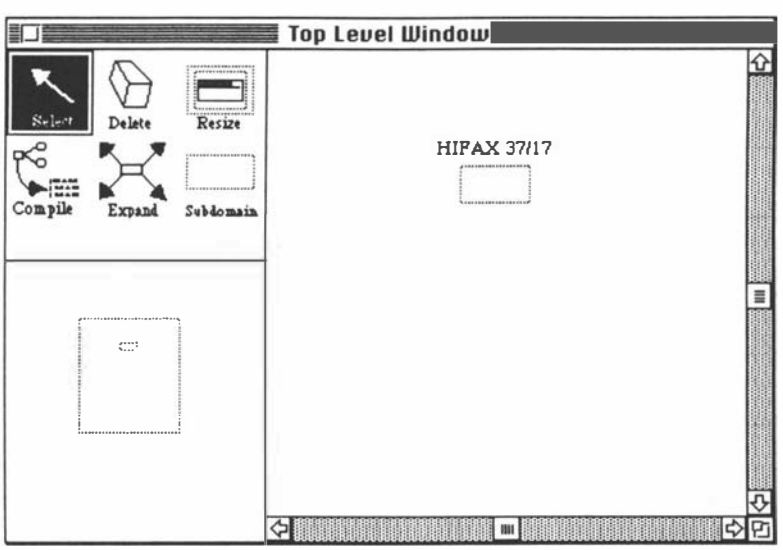


Figure 6.17 : Top level DCE window for a fax machine domain.

Beneath each decomposible sub-domain box can be a plan net sub-window or another level of decomposible sub-windows. The behaviour of the domain is described using plan nets. Further levels of decomposition can also be applied at this level. Figures 6.18 and 6.19 show a second level of

sub-window definition and one of these sub-windows being defined using plan nets in the DCE.

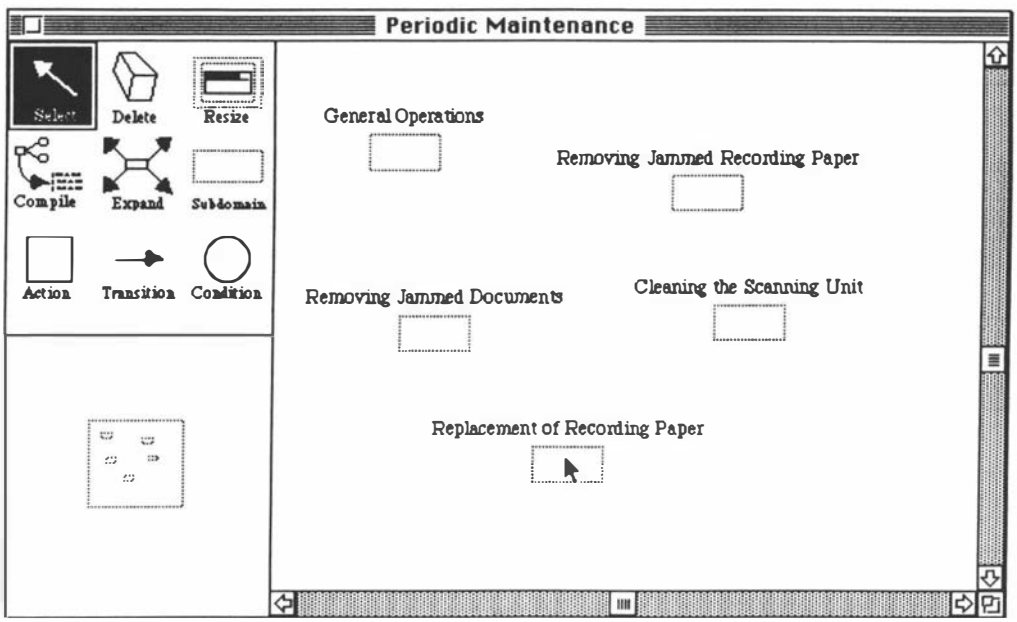


Figure 6.18 : Sub-domain separation.

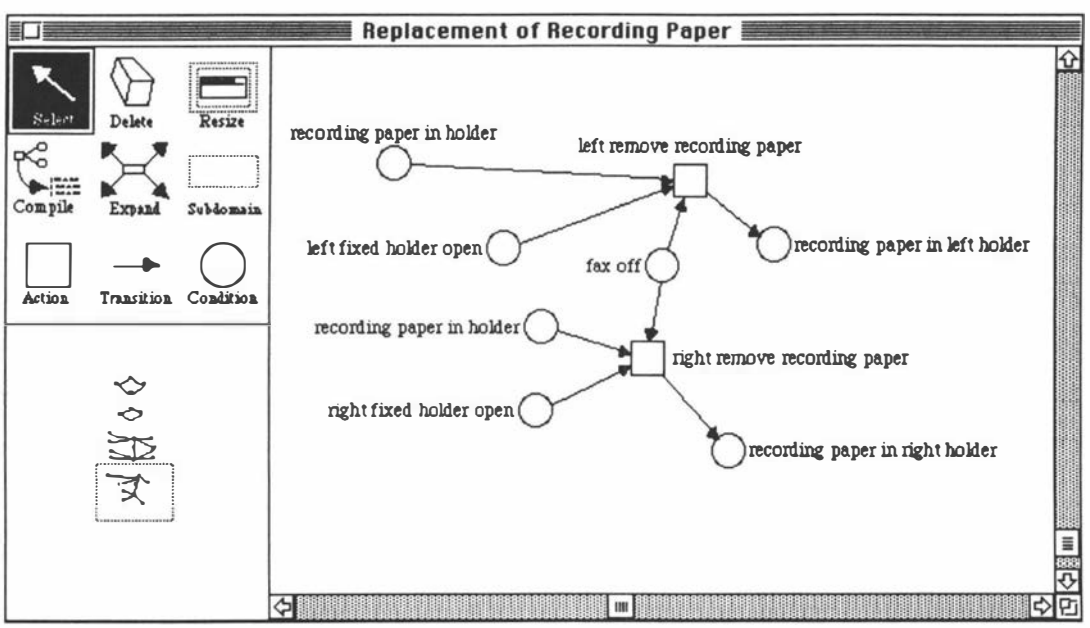


Figure 6.19 : POP table definition using plan nets.

The plan net model of a domain only provides information for the behaviour of that domain. Something that is also needed is the initial situation or a collection of conditions that make up the starting conditions of the environment. In the current implementation, this is separated

from the task definition so that the discovery/exploratory nature of the environment can be emphasised. A separate window is provided for the initial situation definition. Figure 6.20 shows a typical initial situation with several conditions selected for the current domain.

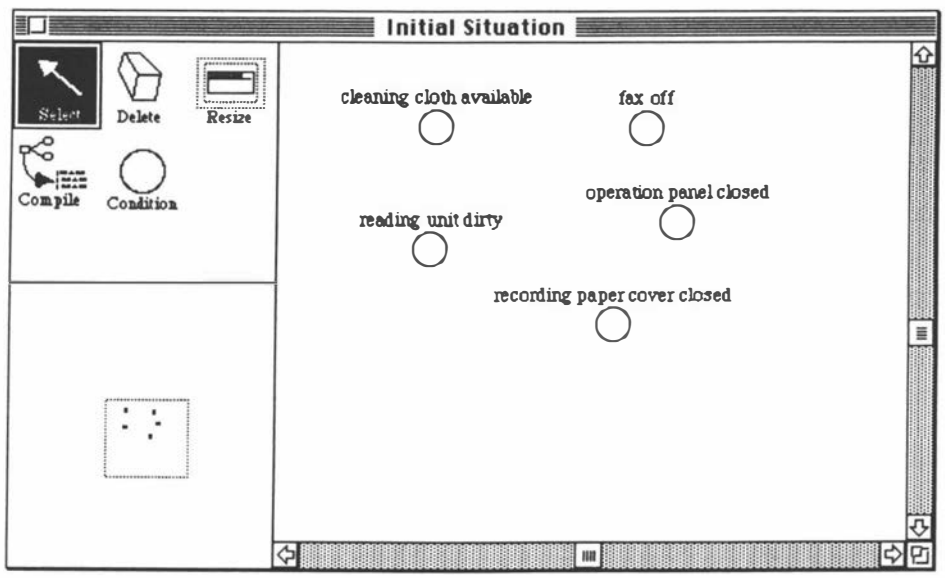


Figure 6.20 : Initial situation definition.

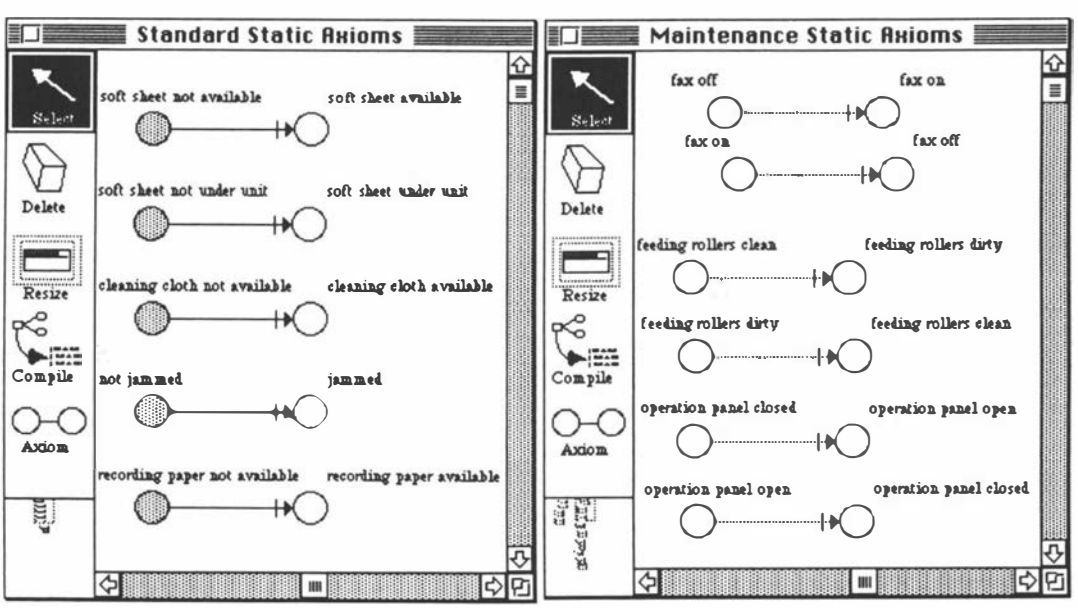


Figure 6.21 : Standard static and maintenance axiom definition.

Standard and maintenance static axioms are currently all defined in their own single window. Static axioms are typically independent from each other and may have relevance to many sub-windows. Therefore, the

need to allow decomposition within this window is minimal. The developed graphical notation (as described in previous chapters) allows THIS..IF and THIS..THEN rules to be easily defined and edited. Again the utilisation of a consistent graphical notation for the components in the domain model definition is applied. Figure 6.21 shows some static axiom definitions, where greyed out conditions indicate that they are inessential.

6.3.3 Testing the Domain Model

Although editing a domain description in the drawing window aids in the construction of the domain model, it is important to be able to check that the described environment accurately reflects the target domain being simulated. To enable this, TANDEM can simulate the effects of actions on the environment at a diagrammatic level. The author inputs a set of conditions that make up the current state of the environment. Then actions can be attempted and the changing state of conditions in the environment can be monitored. After an action, the currently satisfied conditions are highlighted.

Figure 6.22 shows a screen from TANDEM during the simulation of the maintenance of a fax machine. The black nodes indicate that these conditions are currently enabled in the simulation.

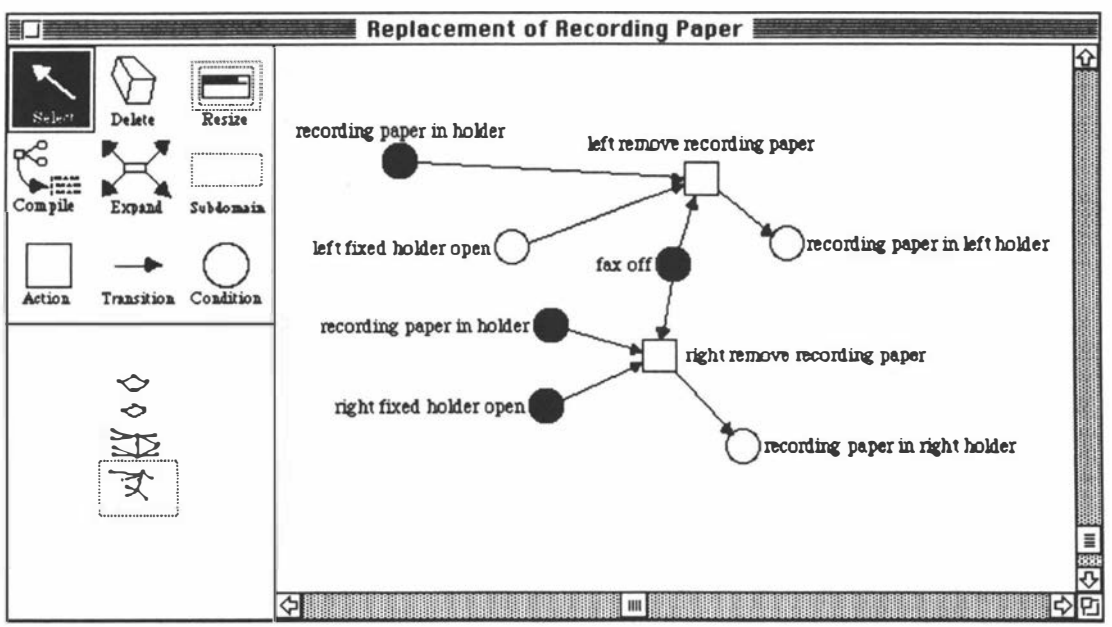


Figure 6.22 : Testing the domain model.

The author can thus verify that the defined domain is behaving in a way that is consistent with the actual environment that is being modelled. Once the author is satisfied that the domain model is correct, the task model can be overlaid upon it.

6.3.4 Defining Task Information

There are two types of task information that can be defined in the DCE, the task model and situated control rules (SCRs). The task model is a hierarchy of tasks and sub-tasks that can be completed in a domain. It is constructed by selecting a set of conditions that signify a goal state and by identifying relevant sub-condition states that are important to the main goal for task completion. Conditions are selected by the author to comprise a task situation and the relevant preconditions from that situation are automatically generated and displayed to the author (Figure 6.23).

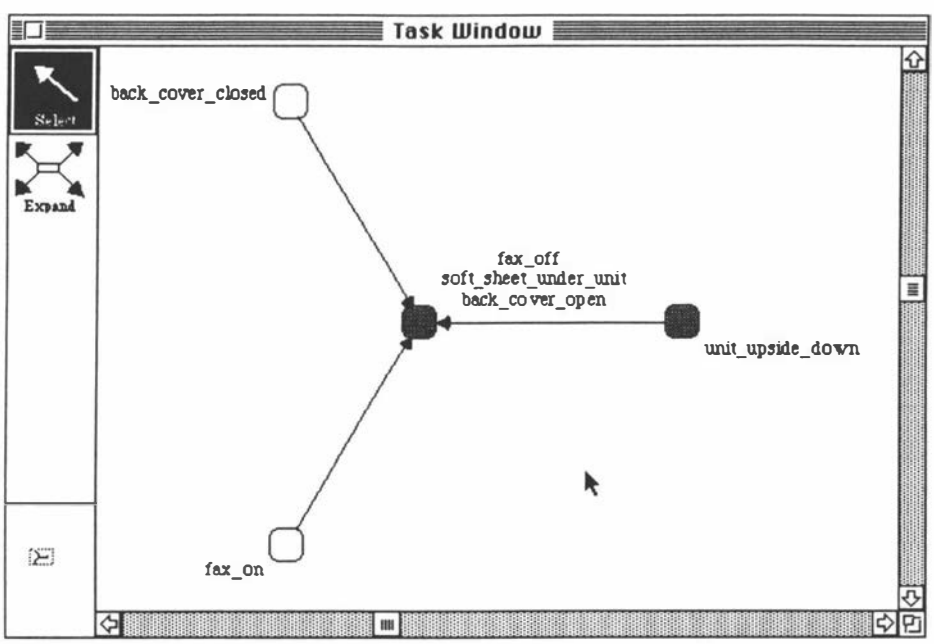


Figure 6.23 : Building a task definition.

The most relevant conditions can be selected by the author and are highlighted in the task window (see *unit_upside_down* in Figure 6.23). Single conditions or groups of conditions can then be selected and decomposed into their own window with their own preconditions displayed. This process continues, enabling a hierarchy of goals and sub-

goals to be constructed. This is a top down process where the author starts with the first goal situation of a task and then breaks it down into it important components. This hierarchy can then be displayed to the author for verification (Figure 6.24). A task model template can then be generated for use with the GDI environment.

Figure 6.24 : Displaying the task hierarchy.

The task model is only half the task information that can be specified by the author. SCRs can also be defined to provide local task pointers within a domain.

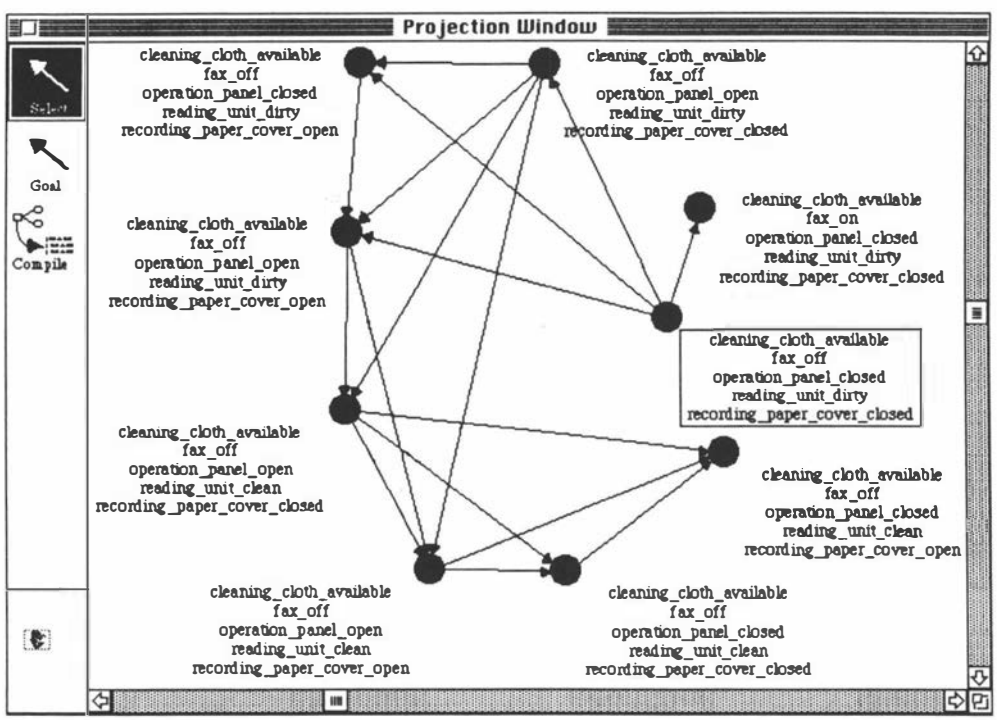


Figure 6.25 : Constructing a projection network.

In the DCE, an author selects a start/initial situation for the generation of SCRs. A projection network from that position is then constructed. An interactive iterative deepening approach has been adopted for the construction of the projection network to allow partial projection networks to be generated at each iteration of the projection (see Appendix A). This has the advantage of allowing reduced projections to be presented to the user. As levels of the projection network are produced, they are displayed to the user who is then asked whether the current projection is adequate. If not, the projection can continue. Otherwise the projection algorithm is concluded (Figure 6.25). The use of reduced projections can shorten their generation time as it may not be necessary for the whole projection network to be generated.

If a node is selected in the projection network with the goal tool, the optimal path from the start node to this goal node is displayed. This path can be annotated on the projection network so that other relevant paths can be added or removed to allow the author to directly specify appropriate paths through the projection (Figure 6.26).

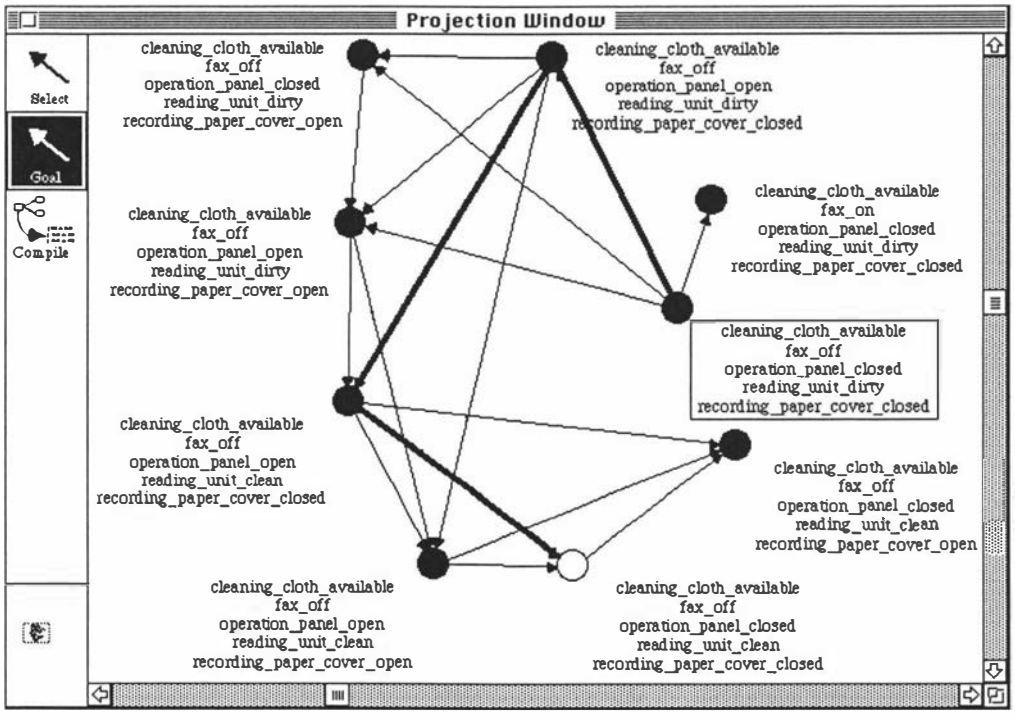


Figure 6.26 : Path selection within a projection network.

The paths selected by the author are used to generate SCRs for all the nodes on these paths. These SCRs can then be displayed to the author (Figure 6.27).

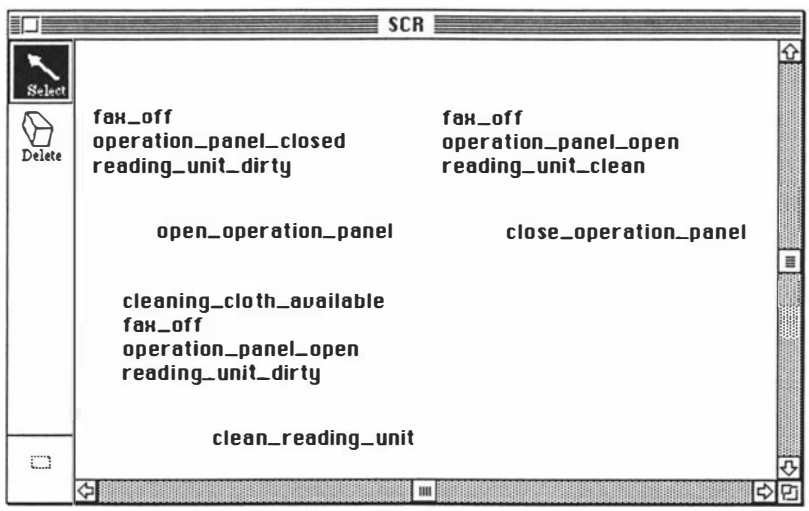


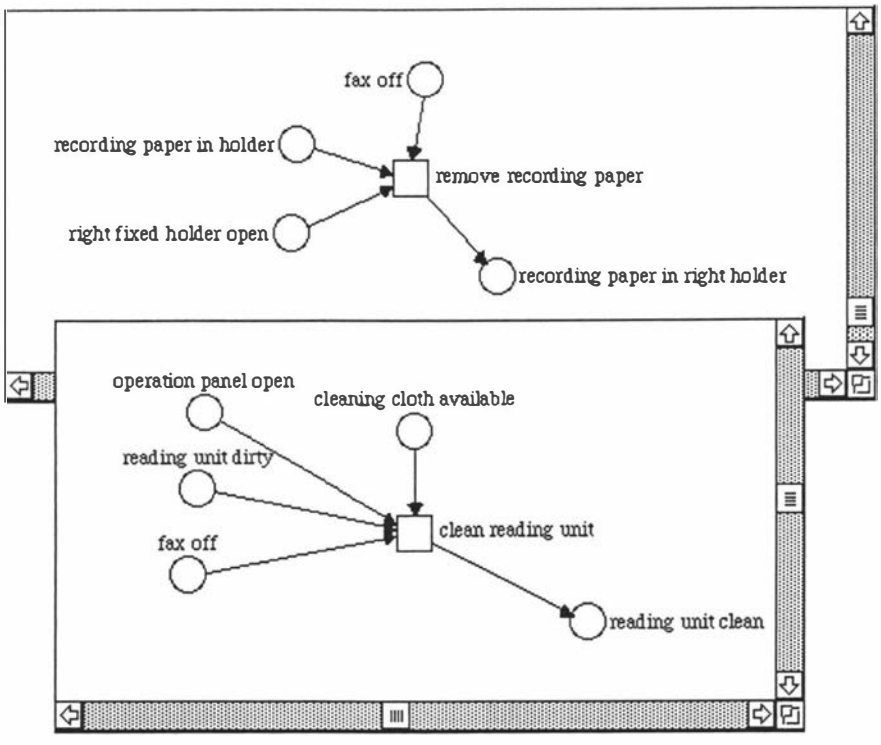
Figure 6.27 : Displaying the SCRs.

The SCRs are displayed as pairs of text items. The top item is a set of conditions which need to be enabled in the current situation for the specific SCR to be relevant. The bottom item is the operator that should be followed in this situation. This is similar to the operator enablement in the POP tables, with the set of conditions acting as preconditions for the SCR operator.

6.3.5 POP Code Generation

Once the domain behaviour of an environment has been defined and tested within the DCE, it needs to be converted into a POP table for use by the TANDEM tutoring engine: the GDI.

As noted in Chapter 4, there is a direct relation between the graphical plan nets and static axioms and the textual POP tables. This allows a straightforward conversion between the two. As entries in a POP table are independent of other entries in the table, multiple windows of plan nets from the DCE can be converted into one POP table (Figure 6.28).




```
operator(remove recording paper).
operator(clean reading unit).

pre(remove recording paper, [fax off, recording paper in holder,
            right fixed holder open]).
pre(clean reading unit, [cleaning cloth available,
            operation panel open, reading unit dirty, fax off]).
post(remove recording paper, [recording paper in right holder]).
post(clean reading unit, [reading unit clean]).
```

Figure 6.28 : Plan net to POP table conversion.

The final conversion between the graphical and the textual representations has two outputs: firstly, the POP table of the plan nets, static axioms and SCR information and secondly the task model for the current domain. The task model is a set of list structures where subgoals are stored as tails in goal lists. This structure is used as a template to generate individual user STMs as necessary in the GDI tutoring engine (Figure 6.29). Both the task model and the STMs are used in the tutoring environment: the first for general task related guidance and the second for more individualised help.

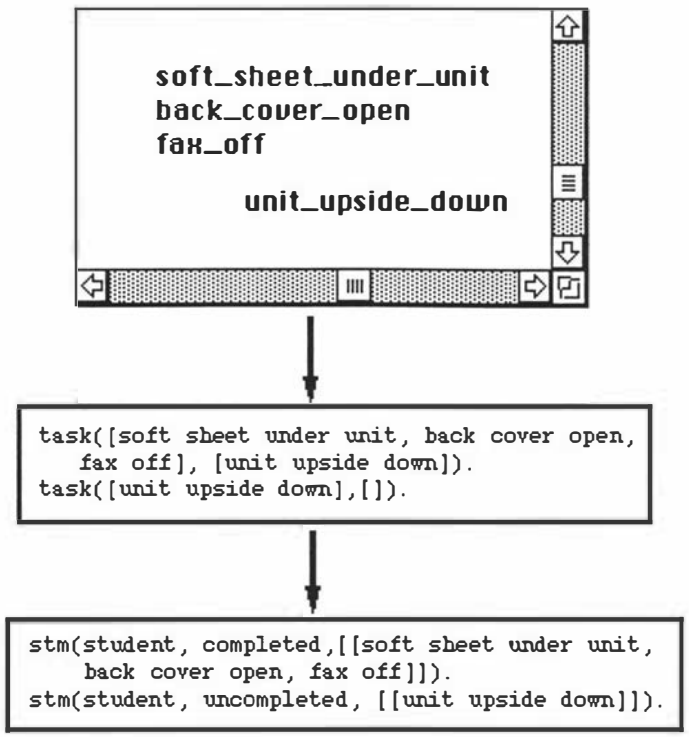


Figure 6.29 : Graphical task to task model to STM.

6.3.6 POP Code Validation

After the initial domain simulation has been defined, the user may be ready to test its validity to see if the domain defined is as accurate as needed or whether it contains any semantic (or domain-dependent) errors. Before this happens it would be an advantageous to run the generated POP table through an automated validation routine to check the POP table is syntactically correct by the fundamental axioms of the POP representation.

When large domains are to be modelled, it may not be possible to test the domain by enumerating all the possible situations and automated validation may be the only realistic check that can be done. Also, before projection networks are generated, automated validation can help identify ill-defined POP tables which can cause problems, such as infinite loops, in the projection network generation process.

Only a limited validation process has been implemented based on the axioms of the POP table representation that have been identified. The

following axioms are currently used to enable syntactic validation of POP tables.

(i) A POP table consists of at least one of each of the following:
A domain name.
A set of operator names.
A set of preconditions for each operator.
A set of postconditions for each operator.
A set of conditions for the initial state of the domain.

(ii) Every operator must have only one precondition list and one postcondition list.

(iii) Every essential precondition condition may only be enabled by either a postcondition or an initial situation condition.

(iv) Every inessential precondition may only be enabled by either a postcondition or an inessential condition.

(v) All essential conditions are either from the initial situation or are postconditions.

(vi) Standard static axioms can only be enabled by preconditions.

(vii) Standard static axioms cannot be enabled by essential conditions.

(viii) Maintenance static axioms can only be enabled by postconditions.

(ix) Maintenance static axioms only disable essential conditions.

(x) Every condition must be either a precondition, postcondition, initial situation condition or a static axiom condition.

6.3.7 Interface Template Generation

One problem with authoring tools is that it may seem advantageous to provide a general interface for the system. This has the advantage of reducing the development time of the system and can provide consistent features between applications. Unfortunately, there is a price to pay for such convenience, the loss of fidelity of the interface. Even when teaching tasks of a similar nature, eg. problem solving or procedural skill learning, the domains where this teaching is to be applied can be extremely varied. In the current work, experiments in a range of domains have been attempted, for example: VCR modelling, block world simulations, car tyre puncture repair, fax machine usage, waste water system maintenance and food handling hygiene techniques. In each of

these domains, distinct interfaces are needed to immerse the user into the appropriate environment for learning.

The DCE produces an interface template for a given domain that can be used as the basis for interaction between an interface and the GDI tutoring engine. Also, a default textual interface is provided so that domains can be tested in the GDI environment. However, the DCE does not provide an interface construction environment as this is outside the scope of the current project. In TANDEM, once a domain has been defined, there are two ways that the DCE allows an interface template to be saved, either as a default textual interface or as an interface template. (Also see Appendix B.)

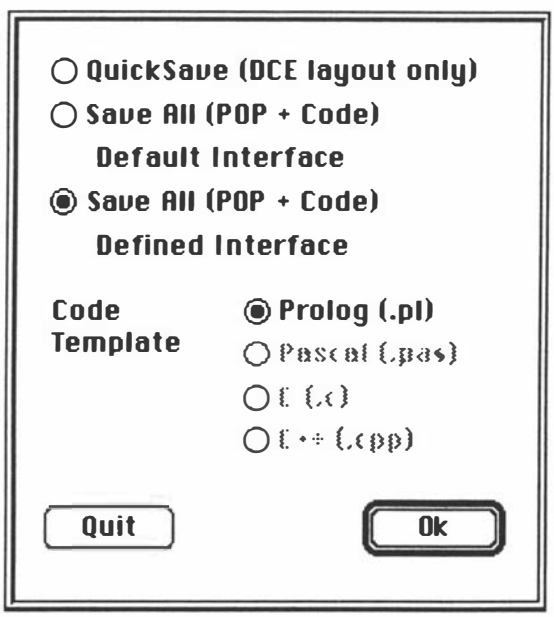


Figure 6.30 : DCE save dialog.

Figure 6.30 shows the DCE save dialog. Firstly, the *Quicksave* option saves only the diagrammatic features of the defined domain in the DCE. No code is generated and this option is useful when incomplete domains are to be saved. Secondly, the domain can be saved with a default textual interface. The POP table is generated and saved with the DCE layout information and a basic textual interface template. Thirdly, an interface template based on the operators in the current domain is constructed and saved to a file. Currently, only interface templates for Prolog are supported, but it is envisioned that it would be possible to allow interfaces to be built in a variety of languages and have templates from the DCE to define the interaction with the GDI tutor.

The domain dependent interface templates are required for four major tasks in the interface to/from GDI interaction. Namely:

- Interface initialisation / de-initialisation.
- Passing events in the interface to the GDI.
- Passing help commands in the interface to the GDI.
- Passing command responses from the GDI to the interface.

Interface initialisation and de-initialisation is needed when a session in a domain is initiated and finished respectively. When events occur in the interface, especially in graphical interfaces, only events that correspond to operators from the domain need to the passed to the GDI engine. Thus a basic template for each operator can be defined in the DCE when the domain is compiled. Help events in the interface are special cases that have their own parameter format when dealt with by the GDI. Also, after an operator has been processed by the GDI, the appropriate result message is passed to the domain interface. (Also see Appendix C.)

6.4 GDI (General Domain Interpreter)

The General Domain Interpreter is the domain independent tutoring engine in TANDEM. Output from the Domain Construction Environment is provided in a standardised form that the GDI can use to allow a discovery learning environment to be presented. This presentation is through a domain dependent interface that is kept separate from the GDI to keep the GDI independent of the domain. Figure 6.31 shows where the GDI fits into TANDEM.

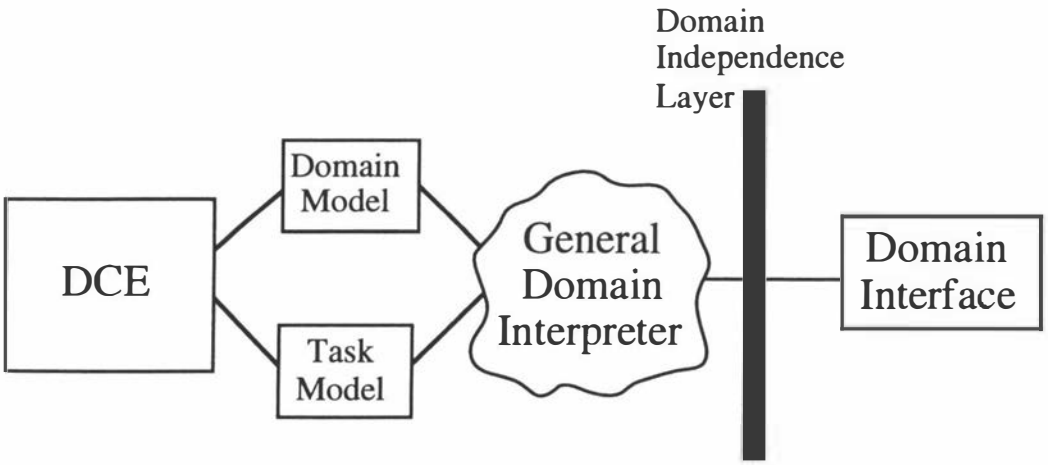


Figure 6.31 : The GDI in TANDEM.

The domain layer is provided so that a variety of different interface styles can be used with the current system. Also, each domain will have its own interface so this layer provides part of a domain independence layer for the GDI. The use of domain dependent interfaces and their linking to the GDI are the focus of section 6.4.2.

The GDI is a tutoring engine for teaching procedural skills in a DCE defined domain. POP tables generated from the DCE are used to generate feedback in response to user commands. This feedback shows the behaviour of the domain being simulated and can be used to generate explanations and help about this behaviour.

There are five stages to a GDI session. Firstly, the user must identify themselves with a unique login name so that their Student Task Model can be accessed, if it exists, or a new one created. Secondly, the type of domain interface that is required must be chosen. Thirdly, the teaching strategy that is to be used in this session must be selected. Next, the tutoring session itself can be started. Finally, after a GDI session is completed, it can be concluded or retried with new parameters for name, interface and teaching strategy. Figure 6.32 shows the process flow through a GDI session.

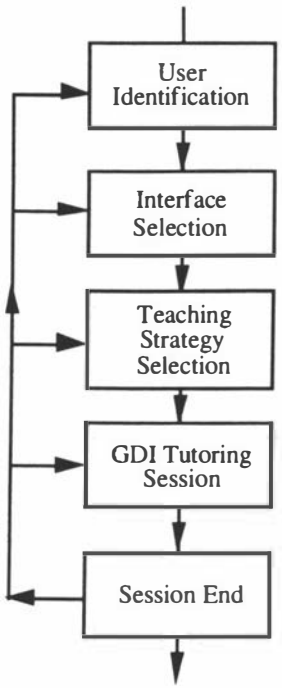


Figure 6.32 : The GDI session process flow.

The collection of the pre-tutoring session information is done by a basic platform dependent interface. Ideally this interface needs to be kept separated from the GDI to allow the GDI to be kept platform independent. (Platform independence will be discussed in the next section.) To enable this, a standard GDI interface template has been defined to ease the construction of this interface for different platforms. The current GDI start-up interface for a Power Macintosh can be seen in Figure 6.33.

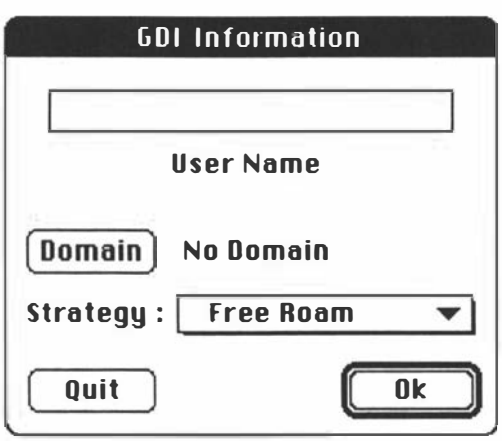


Figure 6.33 : The initial GDI dialog for a Power Macintosh.

Several teaching strategies have been implemented that restrict user navigation in a domain or augment feedback depending on the current type of session. The teaching strategies in the current version of TANDEM correspond to the strategies described in Chapter 5.

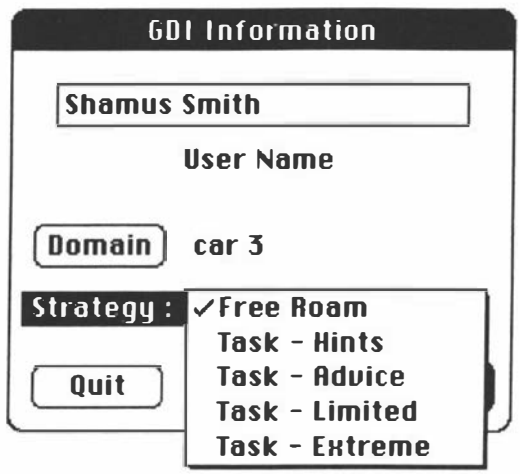


Figure 6.34 : Pre-tutoring session teaching strategy selection.

The interaction between a user and the GDI within a tutoring session is determined by the domain interface that is being used. A selection of different domains and domain interfaces will be examined in Chapter 7. When a session is concluded in a domain interface, control is returned to the GDI start-up dialogue (Figure 6.34). Now the user can quit or retry the session, specifying any new parameters as necessary.

6.4.1 Domain Independence and TANDEM

An important requirement for any authoring tool is that it must be able to model a wide range of different environments. This is especially relevant for ITS authoring tools. Also, the tutoring component of the ITS must be able to model a variety of different domains. As this part of an ITS is extremely time-consuming to build, its reuse with multiple domains increases its potential worth. The main difficulty with the reuse of the tutoring component is the problem of keeping it independent from the domain that is being taught.

The tutoring component of TANDEM is the GDI. It has, by definition, been kept separate from the domain definition process and is provided as a general tutoring engine. This generality from the domain has been

achieved by a domain layer between the GDI and the domain interface (Figure 6.35).

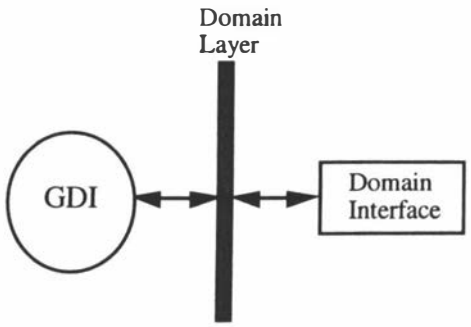


Figure 6.35 : GDI and domain interface separation.

The domain layer converts the domain interface events into the relevant GDI commands. This is achieved through the code generated by the DCE in the interface template. The interface template contains code stubs for all the domain operators for a particular domain. Although this allows the domain interface to be isolated from the GDI in context of the domain, there is still the consideration of the implementation platform and the GDI.

6.4.2 Platform Independence and TANDEM

One important consideration when an ITS is to be developed is how much of the system is required to be reusable. Typically, reusability of a system can be viewed in two ways: firstly, how domain dependent the system is (i.e. whether components of the system can be easily and economically reused in the development of future systems) and secondly, on how platform dependent the system is. Changing requirements in a systems specification may result in the need for a change of platform for the system. Especially for general tutoring systems, the ability to be implemented in multiple platforms greatly increases the potential of the system.

The code in TANDEM is separated into two types, the platform dependent code and the platform independent code. Due to the graphical nature of both the DCE and defined domain interfaces for TANDEM, both these sections are highly dependent on the implementation platform. However, the GDI has been written in standard Prolog and avoids the use

of any platform specific functions. The aim of this is to enable the tutoring engine portion, the GDI, to be easily ported to alternative platforms if desired. To keep the GDI platform independent, there needs to be some insulation between the GDI, the DCE and any domain interfaces (Figure 6.36). This can be achieved by providing a platform layer between the GDI and its interactions with the other components of TANDEM.

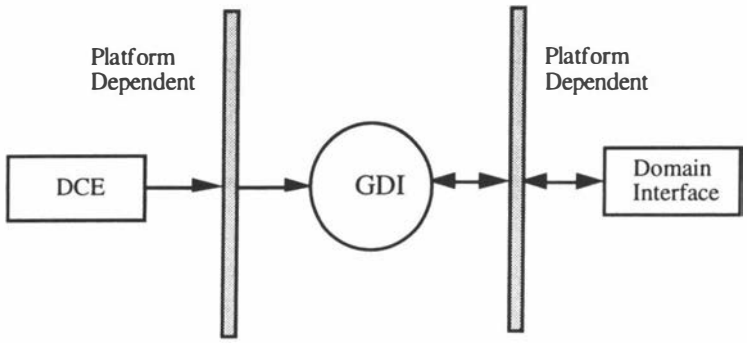


Figure 6.36 : Insulating the GDI.

Firstly, let us consider the DCE to GDI interactions. The DCE produces the domain definition files for use with the GDI, namely the POP table and the task model. These files have been given a standardised format that the GDI can process. Therefore, the DCE component could be implemented using any language, platform or interaction method and as long as its output meets the GDI input file standard, they can be used with the TANDEM tutoring system. The second, more difficult, interaction is the interaction between the GDI and the domain interface. As has been discussed earlier, the GDI is kept independent from the domain using a domain layer. A similar idea can be used to extend a platform layer between the domain layer and the domain interface (Figure 6.37).

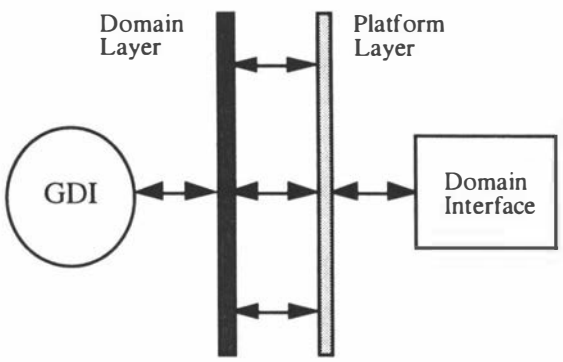


Figure 6.37 : Domain and platform layers in TANDEM.

The platform layer converts the interface dependent commands into domain events which are in turn converted by the domain layer to GDI commands. This can be best illustrated with an example (Figure 6.38).

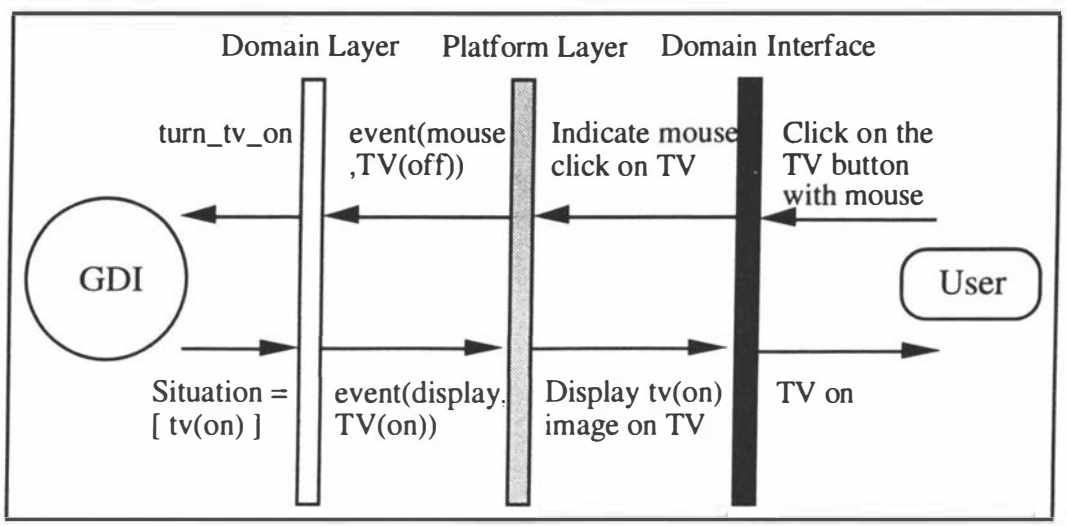


Figure 6.38 : GDI and domain interaction example.

In this example, the user uses a mouse to select the *on* button on a graphical representation of a TV. This is registered by the domain interface as a mouse event in the proximity of the TV graphic. This information is passed to the platform layer as the actual event, eg. *mouse click on TV*, generalising it away from the interface graphical context. This event can be passed to the domain layer which puts the event in context of the current domains operators, eg. *turn_tv_on*. This command can now be passed to the GDI tutoring engine for processing. A similar reverse process is used to pass specified operators through the generalising layers to create a specific event in the domain interface.

Domain and platform independence are important in the area of reusable authoring tools as they help shorten the development time for system construction. Also, it can allow a range of different domains and interface types to be developed and used with the same tutoring system. (Examples of finished domains and their use in the TANDEM environment can be seen in Chapter 7).

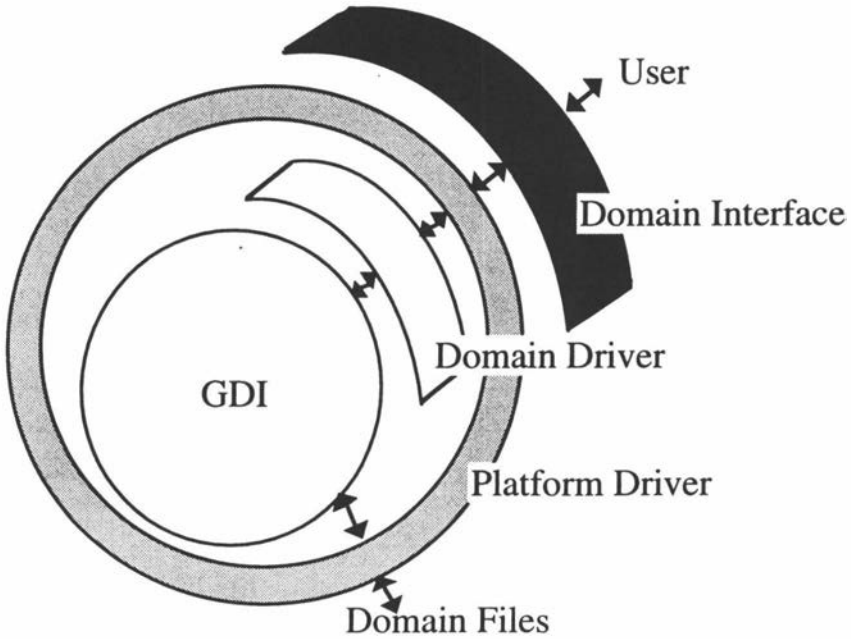


Figure 6.39 : GDI independence model.

Figure 6.39 shows the GDI independence model and how the separation between the platform, the domain files and the domain interface are managed. By using standardised input files and standardised output messages, the GDI has been kept domain and platform independent. This allows it to be reused as the tutoring engine in any number of domains and on a variety of platforms. The only constraint on the use of the GDI for a domain is that the domain can be modelled in the DCE (or by using another tool that produces the appropriately formatted files).

6.5 Summary

TANDEM is an environment for ITS construction. It is specialised to developing ITSs for procedural skill teaching in a discovery learning environment. It has two main components, the DCE and the GDI.

The Domain Construction Environment is a graphical domain definition tool. It provides tools for domain model and task model definition, model testing, code validation and generation, and the production of interface templates for domain interface construction.

The General Domain Interpreter is a domain and platform independent tutoring engine that can be used to provide a discovery learning based tutoring environment for teaching procedural skills. Several teaching strategies have been implemented to augment domain and task feedback from within a tutorial session.

Consideration of domain interface construction issues have been heeded in the development of both the DCE and the GDI so that interface construction is not limited.

Examples of different domains and how they are integrated into a tutoring system from the DCE definition to domain interface use is the focus of the next chapter with examples from three TANDEM defined domains.

Chapter 7

TANDEM Examples

7.1 Introduction

This chapter presents three specimen tutoring environments that have been defined using TANDEM. Firstly, an environment based on a textual user interface is presented. Next, two examples with custom built graphical interfaces are shown. However, before these are discussed, a brief description of the tutoring component, the General Domain Interpreter (GDI), is required. The GDI is the tutoring engine of TANDEM and its links to platform and domain dependent components of a system need to be considered.

7.2 GDI Sessions

The GDI is a domain independent tutoring engine. Two mechanisms are used to keep it domain independent. Firstly, format standardisation of the domain information, as generated by the Domain Construction Environment (DCE), allows the domain model to be provided in a consistent form.

Secondly, a platform/domain layer is inserted between the GDI and the student user interface so that interface issues do not complicate the

teaching environment. These two methods of providing domain independence are shown in Figure 7.1.

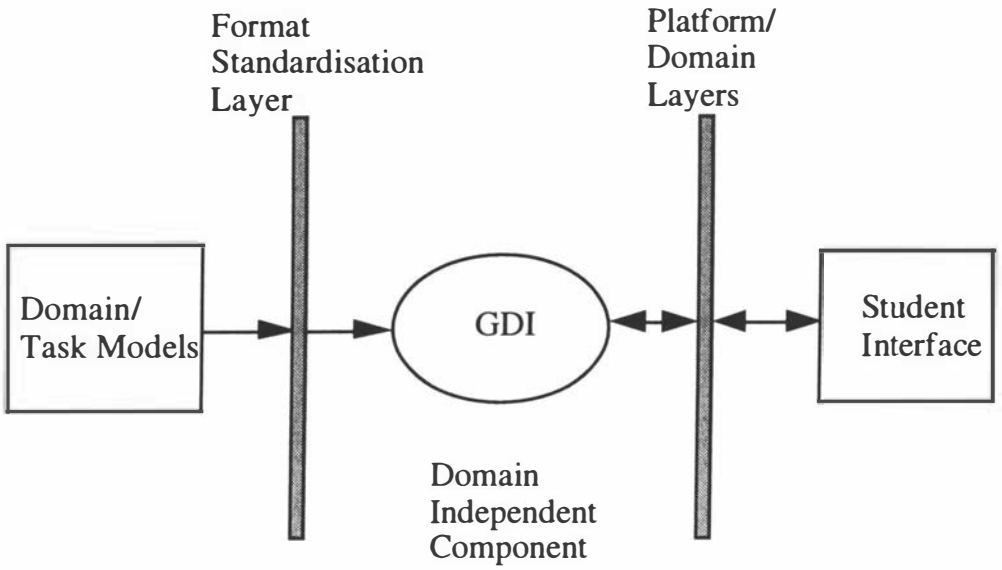


Figure 7.1 : Keeping the GDI domain independent.

Although the code for the GDI has been developed to also be platform independent, for final implementation and testing, platform dependent code is required, especially for interaction with the user. As with the rest of TANDEM's platform dependent code, the GDI interface routines have been implemented using Prolog on a Power Macintosh.

A platform dependent dialogue called the GDI session manager is used to control the TANDEM teaching environment. It collects any pre-session information that is required for a teaching session from the user. It then passes control to the current user interface and finally it concludes the session when requested. Following is the sample GDI session manager from the current works Macintosh platform.

7.3 A Sample GDI Session

Before a GDI session can take place, there are three system variables that need to be set by the user, namely, the user's name and what teaching strategy and interface are to be used for a session. Figure 7.2 shows this dialogue from the current version of TANDEM.

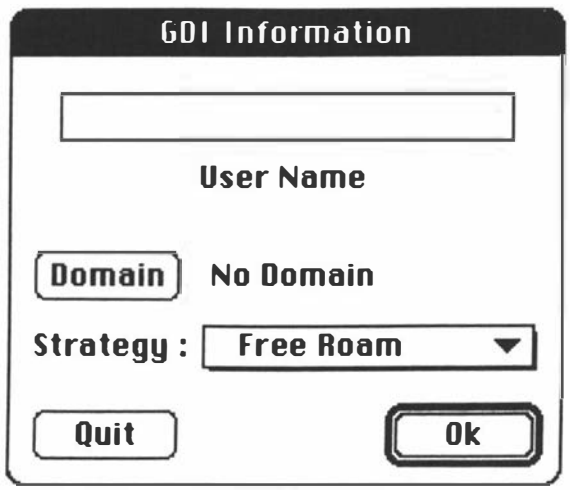


Figure 7.2 : GDI session manager dialog box.

A name is required so that it can be associated with a task model for a particular user. This is used to store a history of GDI sessions.

The teaching strategy selection allows the user to define the degree of help that is available or enforced during a session. Figure 7.3 shows the different teaching strategies that are available and Figure 7.4 is a brief summary of these strategies.

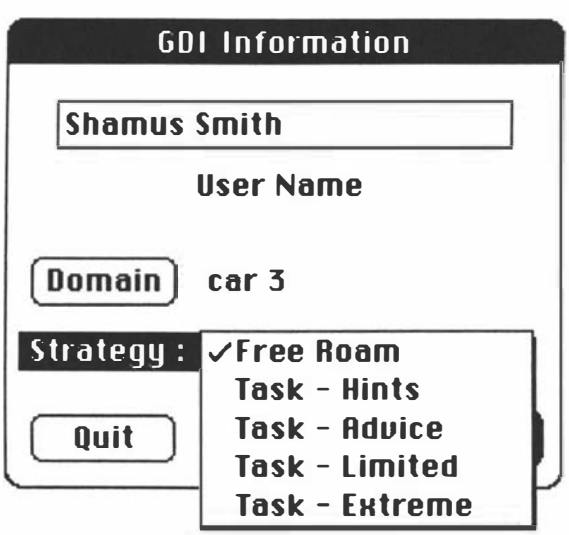


Figure 7.3 : Teaching strategy selection.

Teaching Strategy	Summary
Free roam	No guidance unless requested.
Task - Hints	No guidance unless requested. Hints provided using SCRs and localised projection networks.
Task - Advice	Use SCRs to warn users at critical choice points.
Task - Limited	Enforce SCRs at critical choice points.
Task - Extreme	Enforce an optimal route using SCRs and localised projection networks.

Figure 7.4 : Teaching strategy summary.

There are two types of interfaces that can be used with a GDI session. The session can either be conducted using a default text based interface or by using a customised interface. Both these interface options are platform dependent. However, the default text interface is part of the platform dependent GDI session manager implementation. An example of the Prolog default text based interface can be seen in Figure 7.5.

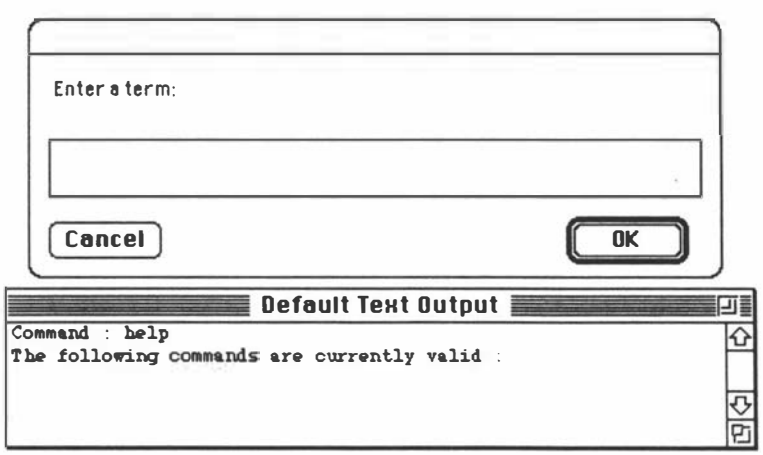


Figure 7.5 : Default text based interface.

Once the three system variables for a session have been defined, control of the session is passed to the appropriate domain interface. To demonstrate the teaching component of TANDEM, three examples will be presented. Firstly, one using the default text based interface and secondly, two using custom built domain interfaces.

Each example will begin with a brief overview of the domain in question and the focus for use in TANDEM. Portions of the domain description from the DCE will be presented and a sample dialogue of a typical tutorial session will be described. Where appropriate, the interaction between the domain interface and the GDI tutoring engine will be highlighted.

7.4 Example 1 : Facsimile Machine Troubleshooting

7.4.1 The Domain

There are many devices in the modern day office that have components that periodically need replacing or have mechanical parts that may become jammed through day-to-day usage. Typical examples of these are printers, photocopiers and facsimile machines. Also, much of this equipment is very expensive and many contain parts that are easily damaged.

Comprehensive manuals are usually supplied with such devices but the possibility of equipment damage through inexperienced handling can be costly in both equipment repairs and the time loss that is incurred due to the equipment being out of operation. The maintenance of a facsimile machine is an example from this area that has been defined in TANDEM. However, the actual use of this device to send and receive documents is outside the current area of interest and only the maintenance tasks have been considered.

There are numerous different makes and models of office equipment but, for the purpose of this example, one specific model has been chosen, the Panasonic Panafax UF-V40 (Panasonic, 1994). Although this example is focused to this particular model, alternative descriptions of a similar nature could easily be defined in TANDEM. The Panasonic Panafax UF-V40 can be seen in Figure 7.6.

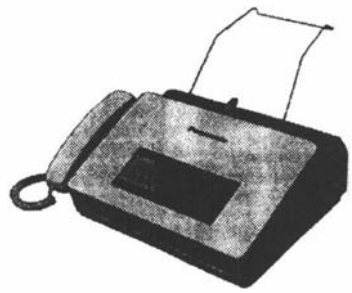


Figure 7.6 : Panafax UF-V40 fax machine; from Panasonic (1994).

7.4.2 The Domain Definition

The behaviour of the fax machine was modelled using the DCE component of TANDEM. A portion of this description is shown in Figure 7.7.

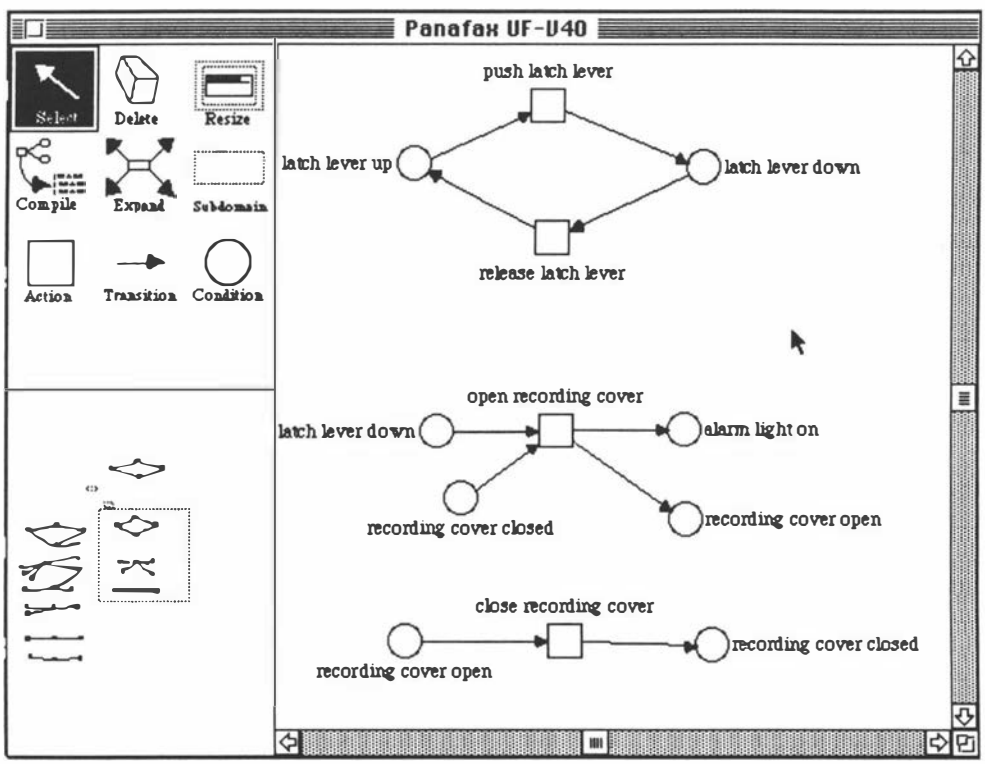


Figure 7.7 : Fax domain definition example.

There are five main maintenance tasks that can be associated with a fax machines daily use. These are replacing recording paper, cleaning the document scanning area, replacing the verification stamp, cleaning recording paper jam and clearing a document jam.

Cleaning the document scanning area is the task that will be used for this example. This task requires that the fax machine is partially disassembled, its scanning area wiped clean and the fax machine reassembled. The major sub-tasks for this problem can be seen in Figure 7.8.

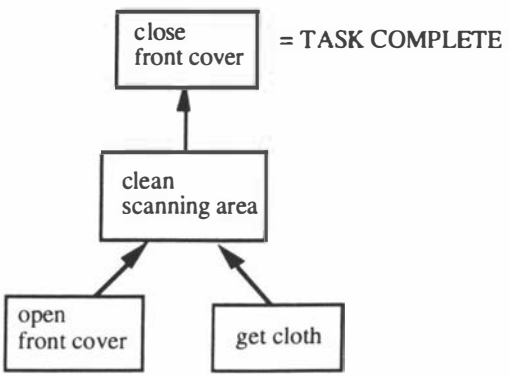


Figure 7.8 : Task hierarchy for *Cleaning the scanning area.*

7.4.3 The Interface

For this example, the default text based user interface that is provided by the Prolog implementation of TANDEM was used. In this interface, text commands are entered through a standard Prolog dialogue window and feedback is presented in a scrollable text window (Figure 7.9).

Figure 7.9 : Fax machine domain dialogue windows.

When a session is complete, the cancel button on the input dialogue window is selected to return control to the GDI session manager.

7.4.4 A Sample Session

Using the current task as an example, the following dialogue is from a typical GDI session in the fax domain (Figure 7.10).

User Commands	System Responses
## start session : FREE ROAM :	
?help	The following commands are currently valid : get cloth turn fax on open document tray Ok
?wipe scanning area	That cannot be done at the moment
?why	'front cover open' is not enabled yet.
?open front cover	That cannot be done at the moment
?why	'front cover partial open' is not enabled yet.
?how	To enable 'front cover partial open', try to: 'partial open front cover'
?open document tray	Ok
?help	The following commands are currently valid : store cloth partial open front cover close document tray turn fax on Ok
?partial open front cover	Ok
?release lock lever	Ok
?open front cover	Ok
?wipe scanning area	That cannot be done at the moment
?why	'cloth available' is not enabled yet.
?get cloth	Ok
clean scanning area	Ok
?help	The following commands are currently valid : close front cover store cloth wipe scanning area turn fax on
?close front cover	Ok
?close document tray	Ok
## end session	

Figure 7.10 : Fax machine domain session example.

7.5 Example 2 : Car Maintenance

7.5.1 The Domain

The area of car maintenance is one that readily lends itself to the use of procedural teaching methods. Many tasks in this environment consist of groups of sequential actions. Also, there are several areas of functionality in this domain that can be easily separated and considered as independent domains, for example, a car's electrical system, its braking system and its engine system.

Car maintenance was initially considered for a graphical test domain in TANDEM's early development. Therefore, a relatively small but significantly complex example was examined. This was the changing of a flat tyre on a car. Simulation of this environment required not only the simulation of the major elements, eg. the car and tyres, but also the workings of other complex devices, eg. a car jack.

7.5.2 The Domain Definition

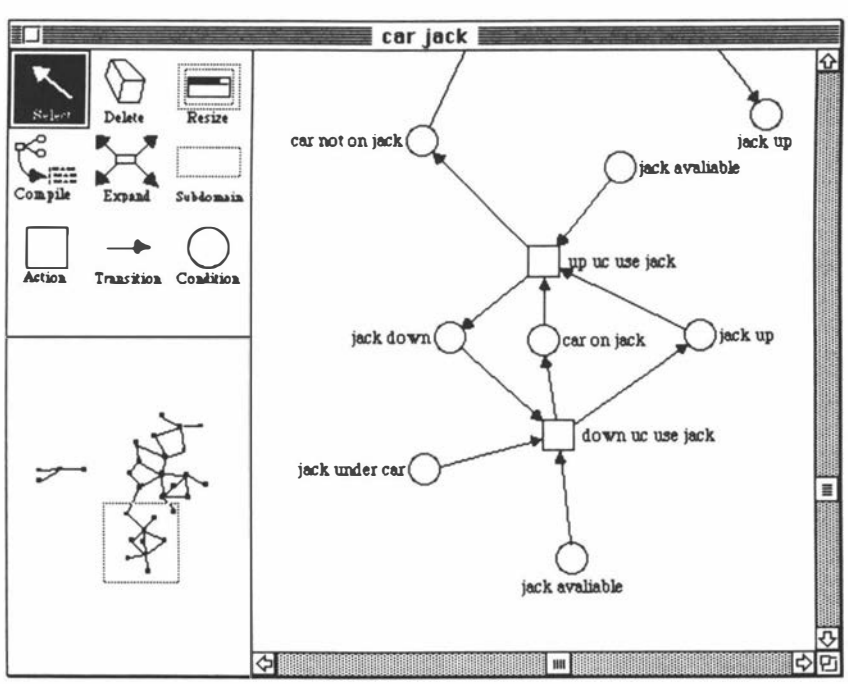


Figure 7.11 : Car maintenance domain definition example.

Part of the domain description for the car domain can be scene in Figure 7.11

Changing the car tyre consists of the six major steps, namely, jacking up the car, removing the wheel nuts, removing the flat tyre, placing the spare tyre, replacing the wheel nuts and the removal of the car jack. These steps can be defined in a task hierarchy over the basic behaviour of the individual components in this environment (Figure 7.12).

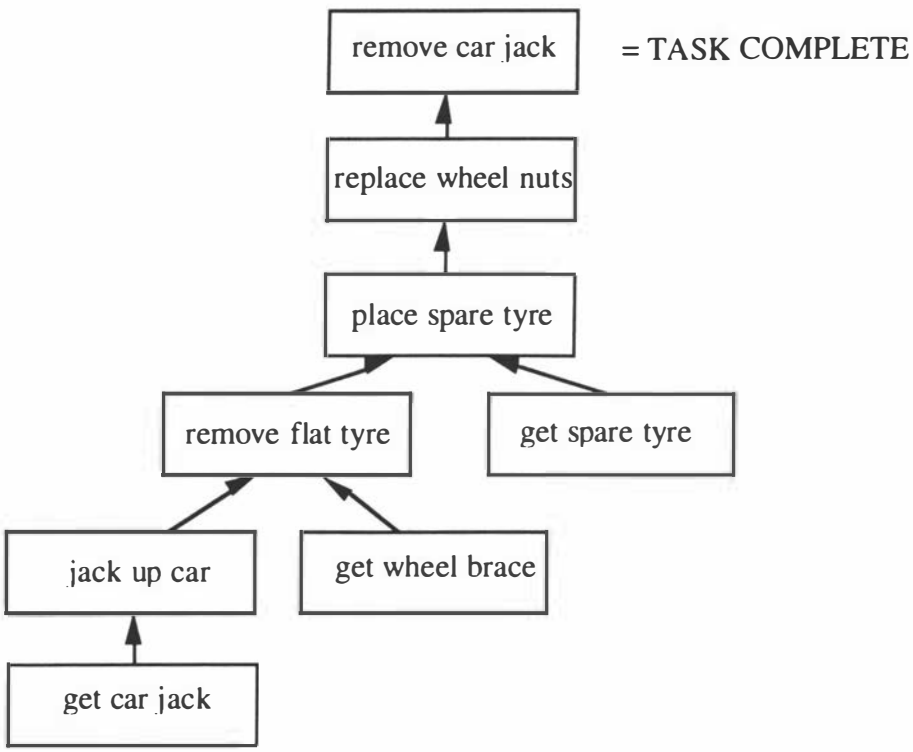


Figure 7.12 : Basic task hierarchy for the car domain.

7.5.3 The Interface

There are six main objects in this domain, namely, the car, the flat tyre, the spare tyre, a car jack, a wheel brace and some wheel nuts (Figure 7.13). Most of these objects can be manipulated independently in the domain. It is the job of the task hierarchy to guide the user to the completion of the task at hand.

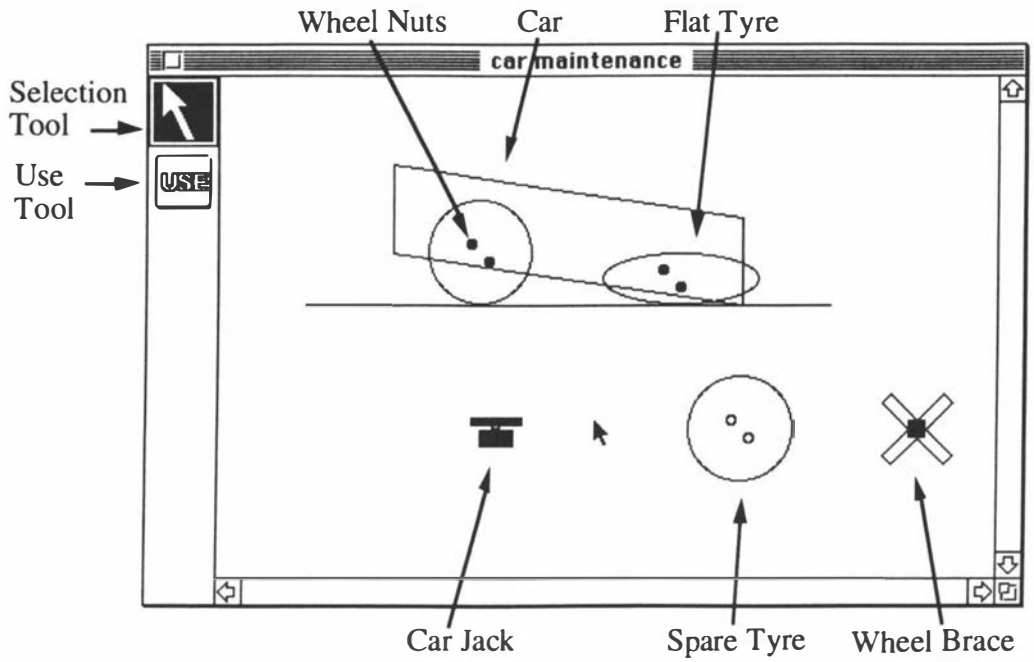


Figure 7.13 : Car maintenance interface.

There are two interface tools that can be used in this domain, the *selection* tool and the *use* tool. The selection tool is used for selecting and moving objects around this environment, eg. the car jack, the tyres and the wheel brace. The use tool is utilised to apply some action to a selected object. Using the use tool while the car jack is selected will result in the car jack being displayed in the up position. If the jack was currently under the car, then the car will be placed on the jack (Figure 7.14).

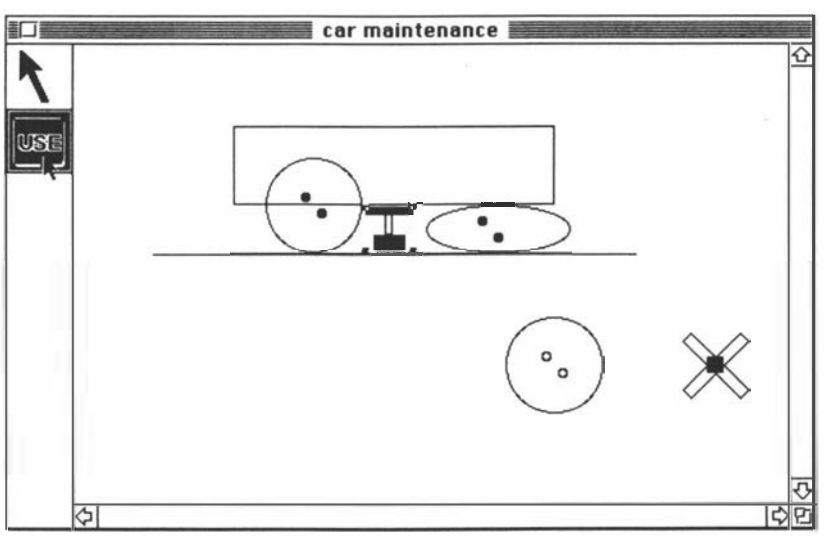


Figure 7.14 : Car jack *used* while under the car.

A similar positioning and action pairing are defined for the wheel brace, eg. if the wheel brace is used while over the wheel nuts, then the wheel nuts would either be removed, if they are in place, or replaced, if they are absent.

The car maintenance domain has been implemented with a graphical interface to show the interface flexibility that is provided when domains are defined in TANDEM and that use the GDI tutoring engine. As described in the Chapter 6, the DCE generates interface code for custom domain interfaces. This code represents the calls that the GDI can process for the chosen domain and must be linked to any custom interface that the user may require.

An example for using the *use* command in the car maintenance domain can be seen in Figure 7.15.

Code Description	Interface Code
Use the jack when it is down.	(a1) `interface_event(use,(Win,[jack_pic]),Return) :-` (a2) `get_pic(Win,jack_pic,jack(Y,X,down)),` (a3) `call_gdi(d_use_jack,Return).`
Use the jack when it is up.	(b1) `interface_event(use,(Win,[jack_pic]),Return) :-` (b2) `get_pic(Win,jack_pic,jack(Y,X,up)),` (b3) `call_gdi(u_use_jack,Return).`
After use command, redraw any items that need it.	(c1) `process_gdi_return(use,[jack_pic],` `[valid_op|Rest],[Win]) :-` (c2) `draw_items(Win,Rest).`

Figure 7.15 : Interface to GDI code.

At a mouse down event in the graphical interface with the use tool on the car jack, an interface event (a1) is generated by the *use* tool. (a2) specifies that the car jack is currently down and if this is true then the GDI is sent the command *d_use_jack* (a3). A similar process happens when the car jack is in the up position (b1-b3).

Where appropriate, each interface event is matched to a GDI command. These GDI commands are processed by the GDI tutoring engine and are then returned to the interface. The *process_gdi_return* predicate is used

to update the domain interface with the new situation that was generated by the GDI. At (c1) in Figure 7.15, any new conditions can be returned by the GDI and the domain interface can be updated (c2).

7.5.4 A Sample Session

A portion of a session in the car maintenance domain can be seen in Figures 7.16a and 7.16b. This session has been started with task advice being provided but not enforced.

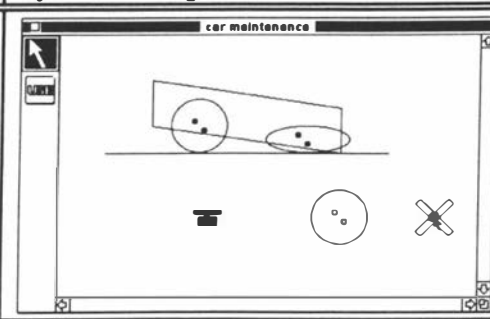
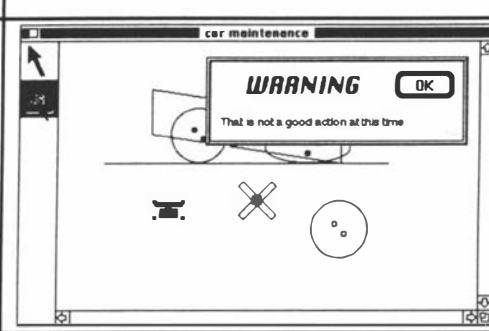
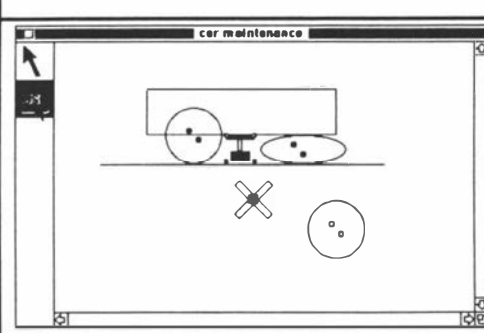
User Commands	System Responses
## start session : TASK : ADVICE	
move wheel brace	
move car jack	
use car jack	
place jack under car	
use jack	
place wheel brace over nut	

Figure 7.16a : Car maintenance domain example session.

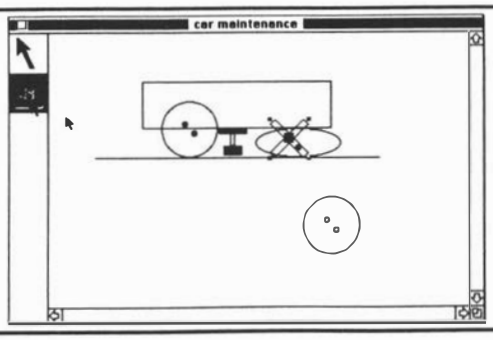
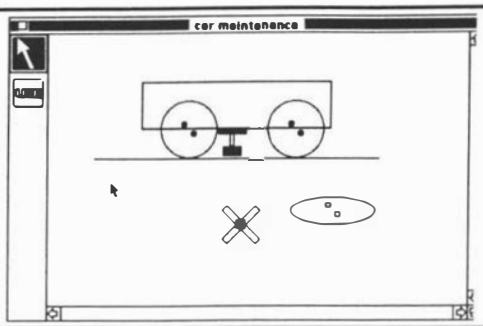
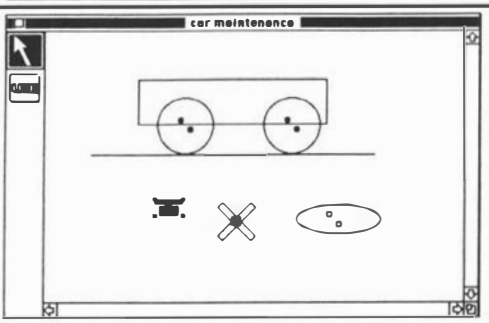
use wheel brace	
move flat tyre, move spare tyre	
replace nut, move wheel brace	
use jack, move jack	

Figure 7.16b : Car maintenance domain example session.

The car maintenance domain has a totally graphical interface and the user commands columns in Figures 7.16a and 7.16b are only included for reader reference.

7.6 Example 3 : Radio Studio Usage

7.6.1 The Domain

A disc jockey (DJ) in a modern radio station is surrounded with a collection of electric devices, from compact disc (CD) players and tape cartridge (cart) players to a studio mixing desk and on-air volume controls.

Hilliard (1985) notes that training programmes in many radio stations consist of the newcomer watching an old-timer perform their duties for a few days. This is typically done with a series of demonstration sessions followed by hands-on practice. This requires a large time commitment on the part of the trainer (McGraw, 1994). Also, many radio stations operate for 24 hours a day, seven days a week and would find it inconvenient and uneconomic to take the station off-air to allow the training of DJs. Another problem would be that a professional station's image may be damaged if untrained DJs were used during normal operating hours.

The aim of this example was to provide an environment to familiarise prospective DJs with the use of on-air equipment before they are unleashed into an actual studio. By allowing initial training to be simulated, DJs can become familiar with the on-air environment and final training can be completed in an actual on-air show, thus reducing the attendance time of the trainer DJs.

7.6.2 The Domain Definition

As with the car maintenance example, the domain definition process for this example was extensive. Not only did the overall studio environment need to be modelled but also the behaviour of each device, eg. CD players, cart players and record turntables.

To reduce the size of the domain, two main restrictions were applied. Firstly, only one of each device is present in the studio, for example, one CD player, one cart player and one turntable. Although this is unrealistic, in terms of the number of devices in an actual studio, it does represent the equipment that is typically found in a radio studio. Secondly, volumes on the studio mixing desk (see Figure 7.19) were supplied in discrete partitions. These are off, medium and high volumes. Again, although this simplifies the environment, it does not detract from the use of the simulation.

Two examples from the radio studio domain model can be seen in Figures 7.17 and 7.18.

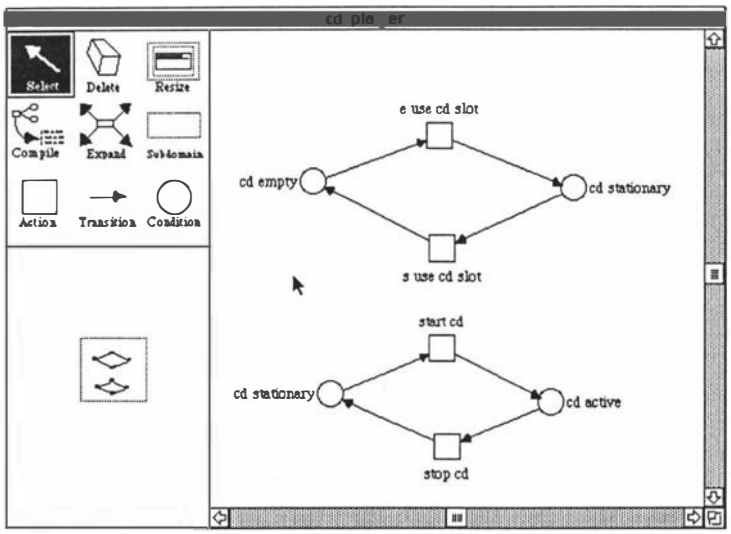


Figure 7.17 : Radio studio domain model example 1.

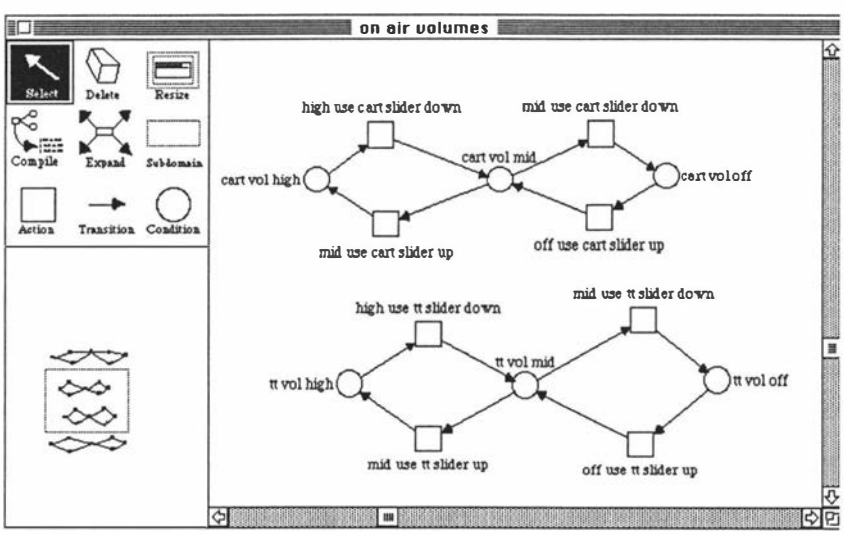


Figure 7.18 : Radio studio domain model example 2.

No explicit task model was defined for this example as its aim is the general use of the simulated environment. However, SCRs are still used to aid the teaching strategies so that appropriate feedback can be provided to the DJs when operating equipment in this environment.

7.6.3 The Interface

This simulation is based on a hands-on training environment, so it is appropriate that a graphical interface is used. To aid in the identification of components of this environment, much of the equipment is labelled, as this is normal in typical radio station studios (see Figure 7.19).

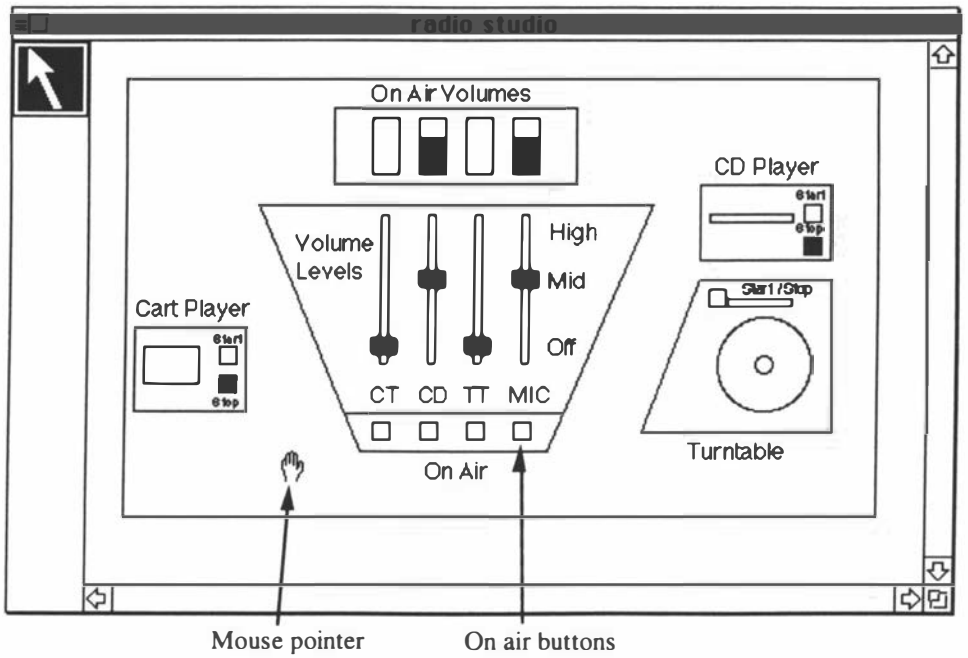


Figure 7.19 : Radio studio interface.

Equipment in this environment is manipulated by the hand pointer (by use of a mouse controller) and the one-button Macintosh mouse. As mentioned previously, the on-air volumes are in discrete units and these are determined by pressing the mouse button on a volume switch and then selecting the appropriate choice (Figure 7.20).

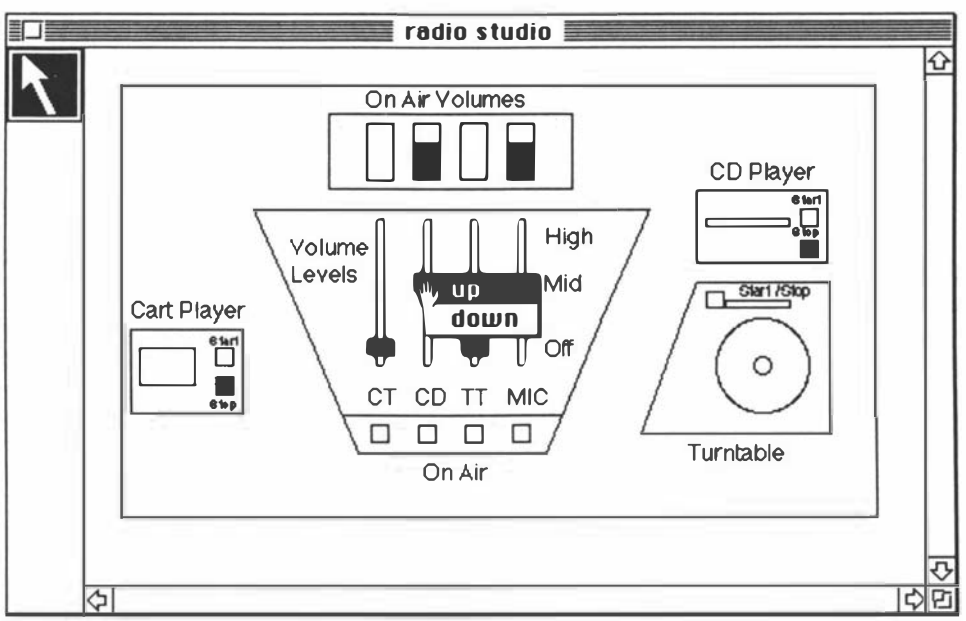


Figure 7.20 : Changing the on-air volumes.

A similar process is utilised to request help and other help related feedback. If the mouse button is depressed in the studio window but not on a valid device hot-spot, then a help pop-up menu is generated (Figure 7.21).

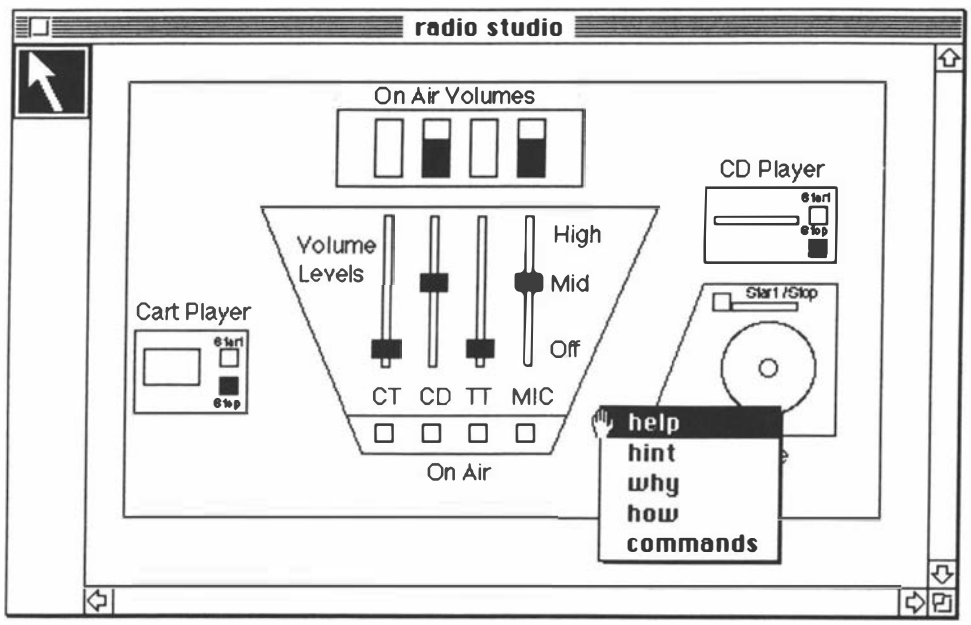


Figure 7.21 : Selecting a help command.

The presentation of feedback has also been implemented graphically. For example, if the help command is selected from the help menu, all the graphical items that are currently valid flash three times on the interface. On a colour monitor, they flash green. A similar process is used for the *hint* and *commands* commands. Thus the user can easily see which items can currently be used and it also aids in the identification of device hot-spots on the interface.

7.6.4 A Sample Session

Due to the graphical nature of this example, a table of its use, as seen in the previous examples, is not appropriate. What is presented are some screen captures from a typical radio studio session (Figures 7.22-7.24).

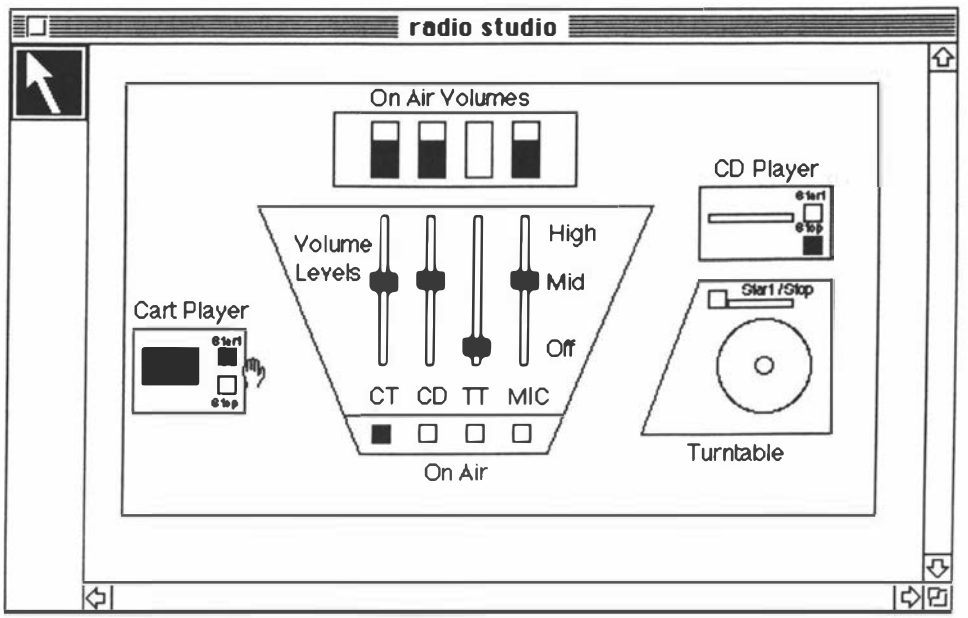


Figure 7.22 : Playing a cart.

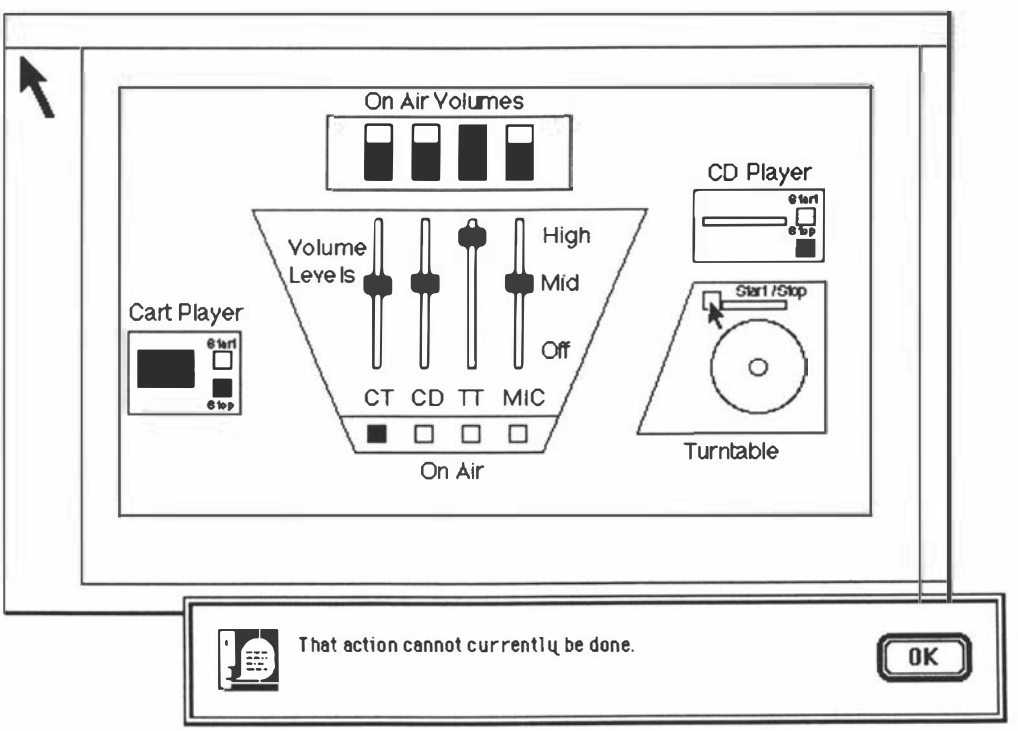


Figure 7.23 : Starting an empty turntable.

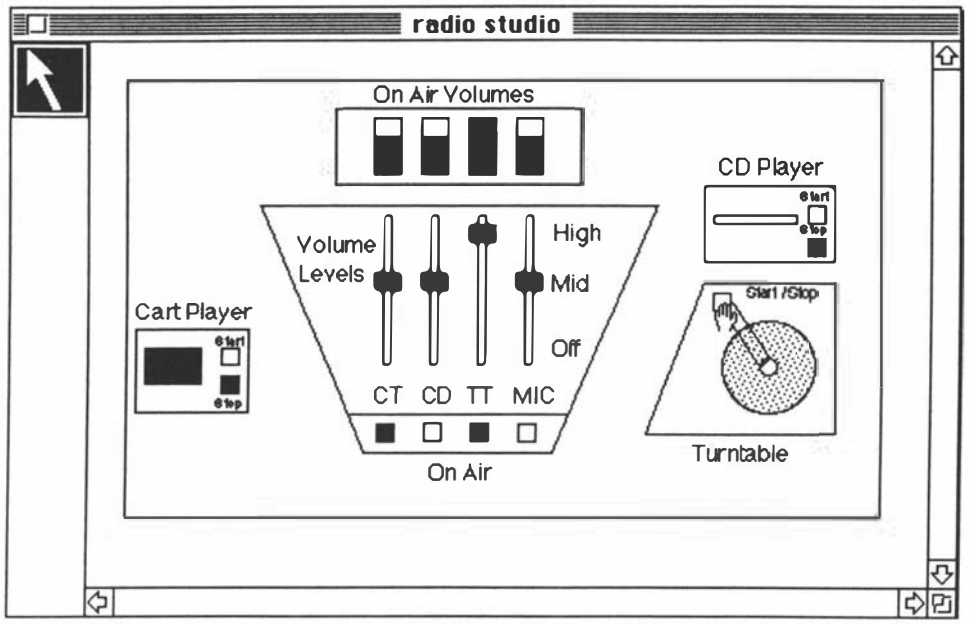


Figure 7.24 : Playing a record.

7.7 Summary

In this chapter, three sample tutoring environments developed in TANDEM have been presented. They demonstrate the use of the GDI tutoring engine and the practical application of methods to preserve the domain independence of the GDI.

Two examples using graphical interfaces and one example using the default GDI textual based interface have been defined. Typically, the text interface can be used to test a domain in the early stages of its definition before a more complex custom interface is developed.

Code generated by the DCE after the domain definition process can be used to enable custom domain interfaces to be developed and linked to the GDI tutoring engine. This provides a customisable instruction environment with the reuse of the teaching strategies and tutoring components.

Chapter 8

Conclusions and Future Work

8.1 Introduction

This chapter presents a summary of the research described in this thesis, the contributions of the research and a description of relevant areas for future work.

8.2 Summary of the Research

The research described within this thesis has two main aims. Firstly, to incorporate several ideas from the area of Artificial Intelligence (AI) into an educational environment and secondly, to implement an authoring system based on these ideas.

The use of AI techniques allows a fresh perspective to be taken in the area of computer based tutoring systems. This is important as it can provide solutions from one research area to another. The use of several AI techniques in a computer based environment has been considered by Kemp (1995). These ideas have been refined and integrated into an ITS authoring environment. Other relevant ideas proposed by Kemp include the separation of domain and task considerations and the teaching of procedural skills.

These concepts have provided a firm foundation for the current work and have been advanced to provide the theory which is the basis for the TANDEM environment. This includes a methodology for the construction of procedural skill based tutoring systems (Figures 6.4-6.7), a prototype authoring tool and a general tutoring engine (Chapter 6).

TANDEM is an authoring tool for the construction of tutoring systems teaching procedural skills in a discovery learning environment. It comprises two main components. The Domain Construction Environment (DCE) is a package that allows the development of domain descriptions. Both domain and task models can be defined and tested. This package has been aimed at domain experts who wish to develop their own tutoring systems. The DCE provides an environment where planning and knowledge representation techniques from AI research can be transparently presented to a user. Users without specialised knowledge representation skills can define domain behaviour without the necessity of choosing or defining their own knowledge representation techniques. Also, the DCE produces domain descriptions in a standardised form so that they can be reused over several domain definitions as required.

The General Domain Interpreter (GDI) is a reusable tutoring engine for domains defined within the DCE. Its domain independence is maintained by separating the domain definition process from the domain interface. For a tutoring session it provides several alternative teaching strategies based on a tutor defined task model and by dynamic path finding through an environments state-space. As the GDI can be reused in many different tutoring systems, the time needed to define new systems can be reduced. Reducing the development time is an important issue for any authoring tool. In TANDEM, this is facilitated by keeping the GDI domain-independent. This is best summarised by noting that if a domain can be defined in the DCE then the GDI can tutor it.

A third component of a tutoring system that needs consideration is its user interface. The development of user interfaces is a huge area of research in its own right. In the current work, it has been discussed only superficially, although its interaction within TANDEM has been investigated. The DCE provides an interface template for each domain defined within it. This can be used to build custom interfaces for domains

which will be compatible with the GDI. This process has been demonstrated with examples in Chapter 7.

Problems that are encountered when representing large domains have also been discussed. The use of sub-domains to help reduce these problems have been investigated and techniques to generate these sub-domains, both manually and automatically have been presented.

8.3 Contributions of Research

The contributions of the research presented in this thesis can be viewed in two complementary areas, namely as theory and practical components. These will be summarised below.

8.3.1 Theory Work

Several important issues were investigated to provide a firm foundation to base the current research on. These included:

- A review of authoring tools for ITSs is presented with a focus on the requirements of these tools.

- Investigation of domain and task modelling issues and the definition of several AI-based techniques. The main technique is the POP (Precondition/Operator/Postcondition) table. Along with Tenenberg's (1991) static axioms, POP tables provide the basis for domain model definition. Drummond's (1989) plan nets have been used as a graphical POP table representation. Also, graphical representations for static axioms and task models have been defined.

- Sub-domain definition and generation have been investigated. Several measures for domain representation complexity have been developed and techniques for sub-domain generation presented.

- Teaching strategies for teaching procedural skills in discovery learning environments have been investigated. Using multiple teaching strategies to aid domain and task feedback during a tutoring session have been discussed. The problems encountered with

teaching strategy selection for tutoring sessions have also been considered.

8.3.2 Practical Work

The specific purpose of the research project has been to investigate whether it is possible to provide a general tutoring system construction environment and to produce a piece of software that demonstrates its feasibility. TANDEM is the software package which has been developed and includes the following features:

- Kemp's (1995) methodology for developing procedural task tutors has been extended and refined to encompass domain, task and interface issues.

- The Domain Construction Environment (DCE) is a direct manipulation tool that allows domains to be graphically defined, edited and tested. The DCE has the following features:

    (i) Definition of domain behaviour using plan nets and static axioms.
    (ii) Definition of task information for a domain using projection graphs and situated control rules.
    (iii) Testing of the domain model.
    (iv) Generation of POP tables from graphical descriptions.
    (v) Domain interface template generation.

- The General Domain Interpreter (GDI) is a domain independent tutoring engine for procedural tasks with the following features:

    (i) A reusable tutoring engine.
    (ii) Provision for four teaching strategies.
    (iii) A default text based domain interface.
    (iv) Domain and platform independence.

- Example domains, including custom interfaces, have been developed to demonstrate the features of TANDEM and its potential as an authoring environment.

8.4 Future Work

The implementation component of TANDEM is very much a prototype system. As it is an ongoing project, parts of TANDEM are continually being refined as it is applied to different domain examples. The example domains presented in this thesis are only a sub-set of the domains that have undergone definition in the TANDEM environment. TANDEM has currently been completed to a point where many of its features are available, at varying levels of robustness. This has allowed the tutoring process from domain description to interface development to be tested with sample domains. This provides an authoring environment framework upon which future research can be based.

Following are some of the areas that were identified during TANDEM's development that would provide interesting territory for future research.

- The ground work for sub-domain generation has been started. Measuring the complexity and trade-offs between sub-domain generation techniques could be very rewarding. Possible benefits include easier domain manipulation and domain reuse.

- The use of sub-domains in TANDEM is not supported. Provision for their use in generating projection graphs is included in the projection generation algorithms but without efficient sub-domain generation, they cannot be effectively used. Also, their use in simplifying SCR generations could be beneficial.

- The generation and display of projection graphs is very expensive in terms of computational resources and screen real estate respectively. One approach to displaying projection graphs has been developed (see Appendix A) but alternative approaches may simplify this process.

- The use of student modelling has been reduced in the current work due to the specialisation of the system to discovery learning environments and the teaching of procedural skills. If TANDEM was required to tutor in alternative environments, the use of increased student modelling could be appropriate.

- The research presented in this thesis is centred on the development of the TANDEM system. The next logical phase, after a full version of TANDEM is completed, is a field test. A pre-Macintosh version of the GDI on a UNIX platform was tested by several postgraduate students early in its development and their feedback was incorporated into the TANDEM construction process. Evaluating the DCE with potential domain authors and the resulting GDI based systems could provide motivation for future research.

- Currently, all the code generated by the DCE is in Prolog. This is required for the GDI but not necessarily for the domain interface. Providing alternative code types for domain interface templates would increase the flexibility for interface development.

- The GDI is domain and platform independent. Moving it to alternative platforms would increase its possible reuse. Two early versions of the GDI have already been ported. Firstly, from a UNIX environment to a Macintosh LC platform running MacProlog version 2.5. Secondly, it was moved to the current platform which is running on a Power Macintosh running MacProlog32. A port back to UNIX or to a Microsoft Windows environment would be an interesting test of the current GDI's platform independence.

- Associated with a port of the GDI, could be the implementation of the DCE on an alternative platform. As long at the output from the DCE is kept to the standard form required by the GDI, new DCE implementations could be used for domain and task definition.

References

Anderson, J. R., Boyle, C. F., Farrell, R. and Reiser, B. J. (1987). Cognitive principles in the design of computer tutors. In P. Morris (Ed.), *Modelling Cognition* (pp. 93-133). Chichester: John Wiley and Sons.

Anderson, J. R. and Reiser, B. J. (1985). The LISP Tutor. *BYTE*, (April, 1985), 159-175.

Apple. (1994). MacOS 7.1.2. USA: Apple Computer Inc.

Apple. (1997). HyperCard 2.3. USA: Apple Computer Inc.

Arienti, G. and Cazzaniga, T. (1990). Coupling Deep and Shallow Knowledge in an Intelligent Tutoring System. In F. Gardin and G. Mauri (Eds.), *Computational Intelligence II* (pp. 11-18). North-Holland: Elsevier Science Publishers.

Asymetrix. (1997). Tool Book. USA: Asymetrix Corporation.

Barker, P. (1994). Designing Interactive Learning. In T. de Jong and L. Sarti (Eds.), *Design and Production of Multimedia and Simulation-based Learning Material* (pp. 1-30). Dordrecht, Netherlands: Kluwer Academic Publishers.

Barr, A. and Feigenbaum, E. A. (1982). *The Handbook of Artificial Intelligence*. USA: William Kaufmann, Inc.

Bell, B. (1995). Towards Modularity in Specialising Authoring Tools. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 4-9). Washington DC, USA.

Bell, M. A. and Jackson, D. (1993). CALVIN - Courseware Authoring Language using Visual Notation. In *IEEE Symposium on Visual Languages* (pp. 225-230). Bergen, Norway: IEEE.

Bertels, K. (1994). A Dynamic View on Cognitive Student Modelling in Computer Programming. *Journal of Artificial Intelligence in Education*. 5(1), 85-105. USA: AACE : Association for the Advancement of Computing in Education.

Bessiere, C. and Vacherand-Revel, J. (1990). Graphical Authoring of Multimedia Courseware. In A. McDougall and C. Dowling (Eds.), *Computers in Education* (pp. 871-876). Amsterdam: North-Holland.

Beveridge, M. (1989). The Educational Implications of Intelligent Systems. In L. A. Murray and J. T. E. Richardson (Eds.), *Intelligent Systems in a Human Context* (pp. 117-136). Oxford: Oxford Science Publications.

Blessing, S. B. (1995). ITS Authoring Tools : The Next Generation. In J. Greer (Ed.), *AI-ED 95 : World Conference on Artificial Intelligence in Education* (pp. 567). Washington DC, USA: AACE : Association for the Advancement of Computing in Education.

Bloom, B. S. (1984). The 2 Sigma Problem : The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher*, 4-16.

Bloom, C. P. (1995). Roadblocks to Successful ITS Authoring in Industry. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 10-13). Washington DC, USA.

Blumenthal, R., Meiskey, L., Dooley, S. and Sparks, R. (1996). Reducing Development Costs With Intelligent Tutoring System Shells. In D. Suthers (Ed.), *ITS'96 : Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs*. Montreal, Canada.

Bootzin, R. R., Bower, G. H., Zajonc, R. B. and Hall, E. (1986). *Psychology Today* (Sixth ed.). New York: Random House.

Bordier, J., Paquette, G. and Carrier, S. (1990). Building Discovery Environments Using Generic Software. In A. McDougall and C. Dowling (Eds.), *Computers In Education* (pp. 1055-1060). Amsterdam: North-Holland.

Bork, A. (1980). Interactive Learning. In R. P. Taylor (Ed.) *The Computer in the School : Tutor, Tool, Tutee*. New York, Teachers College Press.

Boy, G. (1996). Learning Evolution and Software Agents Emergence. In C. Frasson, G. Gauthier and A. Lesgold (Eds.), *ITS'96 : Third International Conference on Intelligent Tutoring Systems* (pp. 10-25). Montreal, Canada: Springer.

Bratko, I. (1990). *Prolog : Programming for Artificial Intelligence* (Second ed.). Addison-Wesley Publishing Company.

Brown, J. S. and Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*(2), 155-198.

Brown, J. S., Burton, R. R. and Kleer, J. D. (1982). Pedagogical natural language and knowledge engineering techniques in SOPHIE I II and III. In D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems* (pp. 227-282). Academic Press.

Brusilovsky, P., Schwarz, E. and Weber, G. (1996). A Tool for developing Hypermedia-based ITS on WWW. In D. Suthers (Ed.), *ITS'96 : Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs*. Montreal, Canada.

Bryan, G., L. (1969). Computers and education. *Computing and Automation* (18)3, 1-4.

Bull, S., Pain, H. and Brna, P. (1995). Mr Collins : A Collaboratively constructed, inspectable student model for intelligent computer assisted language learning. *Instructional Science* (23), 65-87. Klawer Academic Publishers.

Burton, R. R. (1982). Diagnosing bugs in a simple procedural skill. In D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems* (pp. 157-183). London: Academic Press.

Burton, R. R. and Brown, J. S. (1982). An investigation of computer coaching for informal learning activities. In D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems* (pp. 79-98). London: Academic Press.

Carbonell, J. R. (1970). AI in CAI : An Artificial Intelligence Approach to Computer Assisted Instruction. *IEEE Transactions on Man-Machine Systems*, MMS-11(4), 190-217.

Carlsson, M. and Widen, J. (1993). SICStus Prolog User Manual. Kista, Sweden: Swedish Institute of Computer Science.

Carr, B. and Goldstein, I. (1977). Overlay : a theory of modelling for Computer-Assisted Instruction. *MIT AI Memo 406.*, February, 1977.

Carroll, J. M. and McKendree, J. (1987). Interface Design Issues for Advice-Giving Expert System. *Communications of the ACM*, (January, 1987), 14-31.

Cerri, S. A., Cheli, E. and McIntyre, A. (1992). Nobile : Object-Based User Model Acquisition for Second Language Learning. In M. L. Swartz and M. Yazdani (Eds.), *Intelligent Tutoring Systems for Foreign Language Learning* (pp. 171-190). Berlin, Germany: Springer-Verlag.

Chapman, D. (1987). Planning for Conjunctive Goals. *Artificial Intelligence*(32), 333-377.

Clancey, W. J. (1987a). *Knowledge Based Tutoring*. USA: MIT Press.

Clancey, W. J. (1987b). Methodology for Building an Intelligent Tutoring System. In G. P. Kearsley (Ed.), *Artificial Intelligence and Instruction : Applications and Methods* (pp. 193-227). USA: Addison-Wesley.

Computer Associates (1996). SuperCalc. USA: Computer Associates International.

Cox, R. and Cumming, G. (1990). The Role Of Exploration-Based Learning In The Development Of Expertise. In A. McDougall and C. Dowling (Eds.), *Computers In Education* (pp. 359-364). Elsevier Science Publishers.

Dillenbourg, P., Schneider, D., Mendelsohn, P. and Borcic, B. (1995). Design Alternatives for Domain-Independence. In *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems*, (pp. 24-27). Washington DC, USA.

Direne, A. I. (1993). Methodology and Tools for Designing Concept Tutoring Systems. In P. Brna, S. Ohlsson and H. Pain (Eds.), *AI-ED 93 : World Conference on Artificial Intelligence in Education* (pp. 58-65). Edinburgh, Scotland: AACE : Association for the Advancement of Computing in Education.

Direne, A. I. (1995). Authoring intelligent systems : Short and long-term solutions. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 28-31). Washington DC, USA.

Dooley, S. A., Meiskey, L., Blumenthal, R. and Sparks, R. (1995). Developing Usable Intelligent Tutoring System Shells. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 32-35). Washington DC, USA.

Drummond, M. (1985). Refining and Extending the Procedural Net. In J. Allen, J. Hendler and A. Tate (Eds.), *Readings in Planning* (pp. 667-669). San Mateo: Morgan Kaufmann.

Drummond, M. (1989). Situated Control Rules. In R. Brachman, H. J. Levesque and R. Reiter (Eds.), *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (pp. 103-113). Morgan Kaufmann Publishers.

Eberts, R. E. and Brock, J. F. (1988). Computer-Based Instruction. In M. Helander (Ed.), *Handbook of Human-Computer Interaction* (pp. 599-627). Amsterdam: Elsevier Science Publishers.

Fairweather, A., Gibbons. S, Rogers, D., Waki, R. and O'Neal, A. (1992). A model for Computer-Based Training. *AI Expert* (12), Volume 7, 30-35.

Feyock, S. (1977). Transition diagram-based CAI/HELP systems. *International Journal of Man-Machine Studies*(9), 399-413.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS : a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*(2), 189-208.

Gaines, B., R. (1990). Integration issues in knowledge support systems. In J. Boose and B. Gaines (Eds.), *The Foundation of Knowledge Acquisition*, London: Academic Press Limited.

Garg-Janardan, C. and Salvendy, G. (1990). A structured knowledge elicitation methodology for building expert systems. In J. Boose and B. Gaines (Eds.), *The Foundation of Knowledge Acquisition*, London: Academic Press Limited.

Giarratano, J. and Riley, G. (1989). *Expert Systems : Principles and Programming*, Boston: PWS-Kent Publishing Company.

Goldstein, I. P. (1979). The genetic graph : a representation for the evolution of procedural knowledge. *International Journal of Man-Machine Studies*(11), 51-77.

Harel, D. (1988). On Visual Formalisms. *Communications of the ACM*, (May, 1988), 514-529.

Hall, W., Thorogood, P., Sprunt, B., Carr, L. and Hutchings, G. (1990). Is Hypermedia an effective tool for education? In A. McDougall and C. Dowling (Eds.), *Computer In Education* (pp. 1067-1073). Elsevier Science Publishers.

Hill Jr., R. W. and Johnson, W. L. (1993). Designing an Intelligent Tutoring System Based on a Reactive Model of Skill Acquisition. In P. Brna, S. Ohlsson and H. Pain (Eds.), *World Conference on Artificial Intelligence in Education : AI-ED 93* (pp. 273-281). Edinburgh, Scotland: AACE : Association for the Advancement of Computing in Education.

Hill Jr., R. W. and Johnson, W. L. (1994). Situated Plan Attribution for Intelligent Tutoring. In *Twelfth National Conference on Artificial Intelligence* (pp. 499-505). Seattle, USA.

Hilliard, R. L. (Ed.). (1985). *Radio Broadcasting : An Introduction to the Sound Medium* (Third ed.). New York: Longman Inc.

Hollan, J. D., Hutchins, E. L. and Weitzman, L. M. (1984). STEAMER : An Interactive Inspectable Simulation-Based Training System. In G. Kearsley (Ed.), *Artificial Intelligence and Instruction : Applications and Methods* (pp. 113-134). Reading, Massachusetts: Addison-Wesley Publishing Company.

Holt, P., Dubs, S., Jones, M. and Greer, J. (1994). The State of Student Modelling. In J. E. Greer and G. I. McCalla (Eds.), *Student Modelling : The Key to Individualised Knowledge-Based Instruction* (pp. 3-35). Berlin: Springer-Verlag.

Hsieh, P. Y. and Redfield, C. L. (1995). Workshop : Authoring Shells for Intelligent Tutoring Systems. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 43-46). Washington DC, USA.

Huffman, S. and Laird, J. (1994). Learning from Highly Flexible Tutorial Instruction. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Volume One, (pp. 506-512), Massachusetts: AAAI Press/MIT Press.

Hulst, A. (1993). Support for exploration : Reducing the exploration space. In P. Brna, S. Ohlsson and H. Pain (Eds.), *AI-ED 93 : World Conference on Artificial Intelligence in Education* (pp. 561). Edinburgh, Scotland: AACE : Association for the Advancement of Computing in Education.

Humphrey, W., S. (1995). *A Discipline for Software Engineering*. USA: Addison-Wesley Publishing Company.

Hutchings, G., Hall, W., Briggs, J., Hammond, N., Kibby, M., McKnight, C. and Riley, D. (1992). Authoring and evaluation of hypermedia for education. In M. Kibby and J. Hartley (Eds.), *Computer Assisted Learning* (pp. 171-177). Pergamon Press.

Integral Solutions. (1995). Poplog Prolog. USA: Integral Solutions Limited.

Jackson, P. (1990). *Introduction to Expert Systems* (Second ed.). Wokingham, England: Addison-Wesley Publishing Company.

Johns, N. (1994). *LPA MacProlog32 User Guide.* London, England: Logic Programming Associates Ltd.

Johnson, W. L. and Soloway, E. (1987). PROUST : An Automatic Debugger for Pascal Programs. In G. Kearsley (Ed.), *Artificial Intelligence and Instruction : Applications and Methods* (pp. 49-67). Reading, Massachusetts: Addison-Wesley Publishing Company.

Jona, M. K. (1995). Representing and Re-using Generic Teaching Strategies : A Knowledge Rich Approach to Building Authoring Tools for Tutoring Systems. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 47-51). Washington DC, USA.

Kaplan, R. and Rock, D. (1995). New Directions for Intelligent Tutoring. *AI Expert*, (February, 1995), 31-40.

Kass, R. (1989). Student Modelling in Intelligent Tutoring Systems - Implications for User Modelling. In A. Kobsa and W. Wahlster (Eds.), *User Models in Dialog Systems* (pp. 386-410). Springer-Verlag.

Kearsley G., Hunter B. and Furlong F. (1992). *We Teach with Technology : New Visions for Education.* Oregon, USA: Franklin, Beedle and Associates, Inc.

Kemp, R. H. (1992). Intelligent Computer Assisted Instruction : a Knowledge-based Perspective. *Australian Computer Journal*, 24(3), 121-129.

Kemp, R. H. (1995). *Designing Interactive Learning Environments.* Doctor of Philosophy Thesis, Massey University, New Zealand.

Kemp, R. H. and Burns, D. (1992). POPIT : an intelligent teaching package for police training. In A. Holzl and D. Robb (Eds.), *Proceedings of the Second International Conference on Information Technology for Training and Education* (pp. 359-370), University of Queensland.

Kemp, R. H. and Smith, S. P. (1994a). Domain and task representation for tutorial process models. *International Journal of Human Computer Studies*, 41(3), 363-383.

Kemp, R. H. and Smith, S. P. (1994b). Facilitating feedback in discovery learning systems. *Information and Mathematical Sciences Report* (No. 94/19). Massey University.

Kemp, R. H. and Smith, S. P. (1994c). Teaching procedural skills by computer simulation. In *Second Singapore International Conference on Intelligent Systems (SPICIS '94)* (pp. B73-B78). Singapore.

Kemp, R. H. and Smith, S. P. (1994d). Using Planning Techniques to Provide Feedback in Interactive Learning Environments. In P. T. Metaxas (Ed.), *Sixth International Conference on Tools with Artificial Intelligence* (pp. 700-703). New Orleans, USA: IEEE Computer Society Press.

Kemp, R. H. and Smith, S. P. (1996). A Visual Approach to Procedural Tutor Specification. In J. Grundy and M. Apperley (Eds.), *Sixth Australian Conference on Computer-Human Interaction* (pp. 190-196). Hamilton, New Zealand: IEEE Computer Society Press.

Kieras, D. E. and Bovair, S. (1990). The Role of a Mental Model in Learning to Operate a Device. In J. Preece and L. Keller (Eds.), *Human-Computer Interaction* (pp. 205-221). Cambridge: Prentice Hall.

Kitto, C. M. and Boose, J. H. (1989). Selecting knowledge acquisition tools and strategies based on application characteristics. *International Journal of Man-Machine Studies*(31), 149-160.

Lajoie, S., P. and Lesgold, A. (1992). Apprenticeship Training in the Workplace : Computer Coached Practice Environment as a New Form of Apprenticeship. In M. J. Farr and J. Psotka (Eds.), *Intelligent Instruction by Computer : Theory and Practice.* New York: Taylor and Francis.

Lansky, A. L. (1990). Localised Representation and Planning. In J. Allen, J. Hendler and A. Tate (Eds.), *Readings in Planning* (pp. 670-674). San Mateo, USA: Morgan Kaufmann.

Lee, M. (1990). Designing An Intelligent Prolog Tutor. In D. Norrie and H. Six (Eds.), *ICCAC '90 : Computer Assisted Learning* (pp. 420-431). Springer-Verlag.

Leiden, S. H., O'Donnell, J. T., Peterson, E. and Shenk, T. (1987). Building of Expert Systems Using Petri Nets as a Representation of Knowledge Flow. In O. Friesen and F. Golshani (Eds.), *Phoenix Conference on Computers and Communications* (pp. 561-565). Scottsdale, Arizona, USA: IEEE Computer Society Press.

Liddle, J., Brown, K., Slater, A. and MacDonnchadha, S. (1995). Utilising Multiple Training Strategies within Intelligent Industrial Training Systems. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 56-61). Washington DC, USA.

Lockard, J., Abrams, P. and Many, W. (1987). *Microcomputer for Education*. Boston, USA: Little, Brow and Company.

LPA. (1994). LPA Prolog for Windows. London, England: Logic Programming Associates Ltd.

Lu, R., Cao, C., Chen, Y. and Han, Z. (1995). On Automatic Generation of Intelligent Tutoring Systems. In J. Greer (Ed.), *AI-ED '95 : World Conference on Artificial Intelligence in Education* (pp. 67-74). Washington DC, USA: AACE : Association for the Advancement of Computing in Education.

Macromedia. (1995). Authorware 3.0. San Francisco, California: Macromedia Inc.

Major, N. (1993). Reconstructing Teaching Strategies with COCA. In P. Brna, S. Ohlsson and H. Pain (Eds.), *AI-ED 93 : World Conference on Artificial Intelligence in Education* (pp. 66-73). Edinburgh, Scotland: AACE : Association for the Advancement of Computing in Education.

Major, N. (1995). How Generic Can Authoring Shells Become? In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 69-72). Washington DC, USA.

Major, N., Murray, T. and Bloom, C. (1995). Authoring Shells for Intelligent Tutoring Systems. In J. Greer (Ed.), *AI-ED 95 : World Conference on Artificial Intelligence in Education* (pp. 608). Washington DC, USA: AACE : Association for the Advancement of Computing in Education.

Major, N. and Reichgelt, H. (1992). COCA : A Shell for Intelligent Tutoring Systems. In C. Frasson, G. Gauthier and G. I. McCalla (Eds.), *Intelligent Tutoring Systems : Second International Conference, ITS'92* (pp. 523-530). Montreal, Canada: Springer-Verlag.

Mark, M. (1991). The VCR Tutor : Design and Evaluation of an Intelligent Tutoring System, *Research Report No. 91-7*. Department of Computational Science, University of Saskatchewan.

Mark, M. A. and Greer, J. E. (1995). The VCR Tutor : Effective Instruction for Device Operation. *Journal of the Learning Sciences*, 4(2), 209-246.

McGraw, K., L. (1994). Performance Support Systems : Integrating AI, Hypermedia and CBT to Enhance User Performance. *Journal of Artificial Intelligence in Education*, 5(1), 3-26, USA: AACE : Association for the Advancement of Computing in Education.

McKendree, J. (1989). Effective feedback content for tutoring complex skills. *MCC Technical Report* (No. ACT-HI-259-89). Microelectronics and Computer Technology Corporation.

Merrill, M. D., Li, Z. and Jones, M. K. (1990). Limitations of first Generation Instructional Design. *Educational Technology*, (January, 1990), 7-11.

Microsoft. (1995). Microsoft Internet Explorer. Redmond, Washington: Microsoft Corporation.

Microsoft. (1997). Microsoft Excel. Redmond, Washington: Microsoft Corporation.

Miller, M. L. and Lucado, S. R. (1992). Integrating Intelligent Tutoring, Compute-Based Training and Interactive Video in a Prototype Maintenance Trainer. In M. J. Farr and J. Psotka (Eds.), *Intelligent Instruction by Computer* : Theory and Practice (pp. 127-150). Taylor and Francis.

Mosaic. (1994). NCSA Mosaic : Internet information browser. University of Illinois at Urbana-Champaign, USA: The National Centre for Supercomputing Applications.

Muhlhauser, M. (1990). Issues of integrated Authoring/Learning Environments. In A. McDougall and C. Dowling (Eds.), *Computers In Education* (pp. 419-424). Amsterdam: North-Holland.

Munro, A., Johnson, M. C., Surmon, D. S. and Wogulis, J. L. (1993). Attribute-Centred Simulation Authoring for Instruction. In P. Brna, S. Ohlsson and H. Pain (Eds.), *AI-ED 93 : World Conference on Artificial Intelligence in Education* (pp. 82-89). Edinburgh, Scotland: Association for the Advancement of Computing in Education.

Murray, T. (1996). KAFITS. At http://www.cs.umass.edu/~tmurray/: CKS : Centre for Knowledge Communication.

Murray, T. and Woolf, B. (1992). Tools for Teacher Participation in ITS Design. In C. Frasson, G. Gauthier and G. I. McCalla (Eds.), *Intelligent Tutoring Systems : Second International Conference, ITS'92.* Montreal, Canada: Springer-Verlag.

Murray, W. R. (1989). Control for Intelligent Tutoring Systems : A Blackboard-based Dynamic Instructional Planner. In D. Bierman, J. Breuker and J. Sandberg (Eds.), *Artificial Intelligence and Education : Proceedings of the 4th International Conference on AI and Education.* (pp. 150-168). Amsterdam, Netherlands: IOS.

Netscape. (1995). Netscape Navigator. Mountain View, California: Netscape Communications Corporation.

Newman, D., Grignetti, M., Gross, M. and Massey, L. D. (1992). Intelligent Conduct of Fire Trainer : Intelligent Technology Applied to Simulator-Based Training. In M. J. Farr and J. Psotka (Eds.), *Intelligent Instruction by Computer* (pp. 239- 249). New York: Taylor and Francis.

Ngan, P. M. (1992). *The Development of a Visual Language for Image Processing Applications*. Doctor of Philosophy Thesis, Massey University, New Zealand.

Ohlsson, S. (1987). Some principles of intelligent tutoring. In R. Lowler and M. Yazdani (Eds.) *AI and Education*, Volume one. USA: Ablex Publishing Corp.

Or-Bach, R. and Bar-On, E. (1993). "TALK"ing About Evaluation. *Journal of Artificial Intelligence in Education*, 4(2/3), 227-243, USA: AACE : Association for the Advancement of Computing in Education.

Orey, M., Trent, A. and Young, J. (1993). Development Efficiency and Effectiveness of Alternative Platforms for Intelligent Tutoring. In P. Brna, S. Ohlsson and H. Pain (Eds.) *Artificial Intelligence in Education* (pp. 42-49). Edinburgh, Scotland: AACE: Association for the Advancement of Computing in Education.

O'Shea, T. and Self, J. (1983). *Learning and Teaching with Computers : Artificial Intelligence in Education*. Great Britain : Harvester Press Publishing Group.

Oxford. (1990). *Dictionary of Computing*. (Third ed.). Oxford: Oxford University Press.

Panasonic. (1994). *Panafax Uf-V40 User's Guide*. Singapore: Matsushita Graphic Communication System.

Paquette, G., Aubin, C. and Crevier, F. (1994). An Intelligent Support System for Course Design. *Educational Technology* (November-December 1994), 50-57.

Peachey, D. R. and McCalla, G. I. (1986). Using planning techniques in intelligent tutoring systems. *International Journal of Man-Machine Studies* (24), 77-98.

Peters, R. S. (1974). *Ethics and education* (Second ed.). London: George Allen and Unwin Ltd.

Powell, J. A. (1992). The Use of Multi-Media To Train Intelligent Command and Control of Major Fire Incidents. In *ITTE 1992 : Proceedings of the 2nd International Conference on Information Technology for Training and Education* (pp. 91-98). University of Queensland.

Pressman, R., S. (1997). *Software Engineering* : A Practitioner's Approach (Fourth ed.). USA: McGraw-Hill Companies, Inc.

Quintus. (1996). Quintus Prolog. Fremont, USA: Quintus Corporation.

Ragnemalm, E. (1993). Simulator-based training using a learning companion. In M. Brouwer-Janse and T. Harrington (Eds.), *Human-Machine Communication for Educational Systems Design* (pp 207-212). Berlin: Springer-Verlag.

Ralston, A. and Reilly, E. D. (1993). *Encyclopedia of Computrer Science.* (Third ed.). New York: Van Nostrand Reinhold.

Reder, L. and Klatzky, R. L. (1994). The Effect of Context on Training : Is Learning Situated?, *Technical Report (CMU-CS-94-187).* Carnegie Mellon University.

Reinhardt, B. and Schewe, S. (1995). A Shell for Intelligent Tutoring Systems. In J. Greer (Ed.), *AI-ED 95 : World Conference on Artificial Intelligence in Education* (pp. 83-90). Washington DC, USA: AACE : Association for the Advancement of Computing in Education.

Reye, J. (1996). Reusability via Formal Modelling. In D. Suthers (Ed.), *ITS'96 : Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs,* Montreal, Canada.

Ritter, S. and Blessing, S. (1995). Controlling Content : Towards a Domain-General Authoring Framework for Intelligent Tutors. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 78-81). Washington DC, USA.

Ritter, S. and Koedinger, K. R. (1995). Towards Lightweight Tutoring Agents. In J. Greer (Ed.), *AI-ED'95 : World Conference on Artificial Intelligence in Education* (pp. 91-98). Washington DC, USA: AACE : Association for the Advancement of Computing in Education.

Roberts, B. (1993). Constrained Learning Environments for Intelligent Tutoring. In P. Brna, S. Ohlsson and H. Pain (Eds.), *World Conference on Artificial Intelligence in Education : AI-ED 93* (pp. 521-528). Edinburgh, Scotland: AACE : Association for the Advancement of Computing in Education.

Russell, D., Moran, T. and Jordan, D. (1988). The Instructional Design Environment. In Psotka, Massey and Mutter (Eds.), *Intelligent Tutoring Systems, Lessons Learned*. Hillsdale, NJ: Lawrence Erlbaum.

Sacerdoti, E. D. (1977). *A Structure for Plans and Behaviour*. New York: Elsevier Computer Science Library.

Sanborn, J. C. and Hendler, J. A. (1988). A model of reaction for planning in dynamic environments. *Artificial Intelligence in Engineering*, 3(2), 95-102.

Sarti, L. and Van Marcke, K. (1995). Reuse in Intelligent Courseware Authoring. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 82-87). Washington DC, USA.

Schaafstal, A. and Schraage, J. M. (1993). The Acquisition of Troubleshooting Skill Implications for Tools for Learning.. In M. D. Brouwer-Janse and T. L. Harrington (Eds.), *Human-Machine Communication for Educational Systems Design* (pp. 107-118). Berlin: Springer-Verlag.

Schach, S. R. (1993). *Software Engineering*. (Second ed.). USA: Irwin and Aksen Associates Incorporated Publishers.

Slagle, J. R. (1963). A heuristic program that solves symbolic integration problems in freshman calculus. In E. A. Feigenbaum and J. Feldman (Eds.), *Computer and Thought*. New York, USA: McGraw-Hill Book Company Inc.

Sleeman, D. (1987). PIXIE : A Shell for Developing Intelligent Tutoring Systems. In R. W. Lawler and M. Yazdani` (Eds.), *Artificial Intelligence and Education, Volume One : Learning Environments and Tutoring Systems* (pp. 239-265). Norwood: Ablex Publishing.

Sommerville, I. (1996). *Software Engineering* (Fifth ed.). Wokingham, England: Addison-Wesley Publishing Company.

Smith, S. P. (1994a). Computer program for generating situated control rules. Palmerston North, New Zealand: Computer Science Department, Massey University.

Smith, S. P. (1994b). Computer program for procedural simulation using Tenenberg. Palmerston North, New Zealand: Computer Science Department, Massey University.

Smith, S. P. (1994c). Computer program for procedural simulation using TWEAK. Palmerston North, New Zealand: Computer Science Department, Massey University.

Smith, S. P. (1994d). Computer program for projection graph production. Palmerston North, New Zealand: Computer Science Department, Massey University.

Smith, S. P. (1994e). Computer program for teaching procedural tasks. Palmerston North, New Zealand: Computer Science Department, Massey University.

Smith, S. P. (1994f). Computer program to partition sub-domains. Palmerston North, New Zealand: Computer Science Department, Massey University.

Smith, S. P. and Kemp, R. H. (1995). Efficient Modelling of Domains for Computer Tutoring Systems. In R. Kotagiri (Ed.), *ACSC 95 : Proceedings of the 18th Australasian Computer Science Conference*, Vol. 17, Number 1, (pp. 491-498). Adelaide, South Australia: Australian Computer Science Communications.

Sowa, J. F. (1984). *Conceptual Structures : Information Processing in Mind and Machine.* USA: Addison-Wesley Publishing Company.

Specht, D. (1989). Are Intelligent Tutoring Systems Useful for Learning in Technical Environments. In V. Marik, O. Stepankova and Z. Zdrahal (Eds.), *CEPES-UNESCO International Symposium* (pp. 173-178). Prague: Springer-Verlag.

Spensley, F., Elsom-Cook, M., Byerley, P., Brooks, P., Federici, M. and Scaroni, C. (1990). Using Multiple Teaching Strategies in an ITS. In C. Frasson and G. Gauthier (Eds.), *Intelligent Tutoring Systems : At the Crossroad of Artificial Intelligence and Education* (pp. 188-205). Norwood: Ablex Publishing Corporation.

Srisethanil, C. and Baker, N. C. (1996). ITS-Engineering : A Domain Independent ITS for Building Engineering Tutors. In C. Frasson, G. Gauthier and A. Lesgold (Eds.), *Intelligent Tutoring Systems Third International Conference , ITS'96* (pp. 677-685). Montreal, Canada: Springer.

Steinberg, E. R. (1991). *Computer-Assisted Instruction : A Synthesis of Theory, Practice and Technology*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Stevens, A., Collins, A. and Goldin, S. E. (1982). Misconceptions in students' understanding. In D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems* (pp. 13-24). London: Academic Press.

Suppes, P. (1980). The Teacher and Computer-assisted Instruction. In R. P. Taylor (Ed.) *The Computer in the School : Tutor, Tool, Tutee.* New York, Teachers College Press.

Suthers, D. and Lesgold, A. (1995). An Interlingua for Instructional Media. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 88-92). Washington DC, USA.

Syllabus. (1992). Advanced technologies lead the way to the future of educational computing. *Syllabus*, (November/December 1992), 2-8.

Tenenberg, J. (1991). Abstraction in Planning. In J. Allen, H. Kautz, R. Pelavin and J. Tenenberg (Eds.), *Reasoning About Plans*, Morgan Kaufmann.

Thibeault, T. (1994). SmarTText 1.1. Dept. of Foreign Languages, Mailcode 4521, Southern Illinois University, Carbondale, IL 62901-4521.

Turban, E. (1992). *Expert Systems and Applied Artificial Intelligence.* New York, USA: Macmillan Publishing Company.

van Joolingen, W. R. and de Jong, T. (1992). Modelling Domain Knowledge for Intelligent Simulation Learning Environment. In M. R. Kibby and J. R. Hartley (Eds.), *Computer Assisted Learning : Selected Contributions from the CAL91 Symposium* (pp. 29-38). Oxford: Pergamon Press.

Van Marcke, K. (1992). Instructional Expertise. In C. Frasson, G. Gauthier and G. I. McCalla (Eds.), *Intelligent Tutoring Systems : Second International Conference, ITS'92* (pp. 234-243). Montreal, Canada: Springer-Verlag.

Van Marcke, K. (1993). Representing Domain Knowledge. In P. Brna, S. Ohlsson and H. Pain (Eds.), *World Conference on Artificial Intelligence in Education* (pp. 570). Edinburgh, Scotland: AACE : Association for the Advancement of Computing in Education.

van Vliet, H. (1993). *Software Engineering : Principles and Practise.* Chichester, England: John Wiley and Sons.

VanLehn, K. (1996). Conceptual and Meta Learning During Coached Problem Solving. In C. Frasson, G. Gauthier and A. Lesgold (Eds.), *ITS'96 : Third International Conference on Intelligent Tutoring Systems* (pp. 29-47). Montreal, Canada: Springer.

Vassileva, J. (1995a). Dynamic Courseware Generation : At The Cross Point of CAI, ITS and Authoring. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 99-101). Washington DC, USA.

Vassileva, J. (1995b). Reactive Instructional Planning to Support Interacting Teaching Strategies. In J. Greer (Ed.), *7th World Conference on Artificial Intelligence in Education : AI-ED 95* (pp. 334- 342). Washington DC, USA: AACE : Association for the Advancement of Computing in Education.

Waterman, D. A. and Hayes-Roth, F. (1983). An Investigation of Tools for Building Expert Systems. In F. Hayes-Roth, D. A. Waterman and D. B. Lenat (Eds.), *Building Expert Systems* (pp. 169-215). London: Addison-Wesley Publishing Company.

White, B. Y. and Frederiksen, J. R. (1987). Qualitative Models and Intelligent Learning Environments. In R. W. Lawler and M. Yazdani (Eds.), *Artificial Intelligence and Education (Volume 1) : Learning Environements and Tutoring Systems* (pp. 281-305). Norwood, USA: Ablex Publishing.

Wigetman, R., Patacchini, J.-J. and Evrard, F. (1992). Sieel : An Intelligent Tutoring System for Air Traffic Control. In *ITTE 1992 : Proceedings of the 2nd International Conference on Information Technology for Training and Education* (pp. 632-641). University of Queensland.

Woods, P. J. and Warren, J. R. (1995). Generating Tutoring Systems with Versatile Teaching Strategies. In N. Major, T. Murray and C. Bloom (Eds.), *AI-ED '95 : Workshop on Authoring Shells for Intelligent Tutoring Systems* (pp. 104-110). Washington DC, USA.

Woolf, B. (1987). Teaching a Complex Industrial Process. In R. Lawler (Ed.), *Artificial Intelligence and Education* (pp. 414-427). Ablex Publishing Corp.

Woolf, B. (1992). Hypermedia in Education and Training. In D. Kopec and R. B. Thompson (Eds.), *Artificial Intelligence and Intelligent Tutoring Systems* (pp. 97-109). Ellis Horwood.

Woolf, B. P. and Hall, W. (1995). Multimedia Pedagogues. *Computer*, (May, 1995), 74-80.

Yazdani, M. (1987). Intelligent Tutoring Systems : An Overview. In R. Lawler (Ed.), *Artificial Intelligence and Education* (pp. 183-201). Ablex Publishing Corp.

Yob, G. (1975). Hunt the Wumpus. *Creative Computing*,, (September / October, 1975), 51-54.

Glossary[1]

Advice guidance a teaching strategy where SCRs are used to warn users about critical choice points.

Adaptive CAI CAI systems which make use of information obtained from the student to guide its use.

Authoring language a specialised high-level language that allows the user to write limited types of computer-assisted instruction systems without extensive programming knowledge.

Authoring shell see *authoring tool.*

Authoring tool software designed to allow non-programmers to create computer-based learning material and systems.

Authoritarian guidance strategy students start with freedom of movement but are quickly forced onto the path of an ideal student.

Authorware computer programs and related materials specifically designed for authoring courseware.

CAI see *computer assisted instruction.*

CAL see *computer assisted instruction.*

Computer assisted instruction the use of the computer as an instructional tool.

---

[1] Selected entries from Lockard, Abrams and Many (1987), Oxford (1990), Ralston and Reilly (1993) and Turban (1992).

CBI see *computer assisted instruction.*

CBT see *computer assisted instruction.*

Computer based training see *computer assisted instruction.*

Correct action an action which is valid in a domain and appropriate in the current situation.

Courseware computer programs and related materials specifically designed for instruction.

Critical choice point a state in a projection graph that has an associated action which can lead to an undesirable state.

Current situation a list of conditions which are currently enabled in a domain environment.

DCE see *domain construction environment.*

Declarative knowledge a collection of stored knowledge about a domain.

Delayed feedback some system dependent criteria is used to delay user feedback. This gives the user a chance to correct their own mistakes.

Destructive action an action which causes an effect on the current environment which is not reversible.

Device-based guidance strategy a guidance approach based on the simulation of real world devices.

Diagnostic models models used to provide a mechanism for explaining why a student is making a mistake.

Differential modelling a student modelling technique in which the student model is compared to a model describing what an expert would do in the same situation.

Discovery learning a teacher sets up an environment for learning and the student explores it. The teacher can provide guidance when requested or if the user is in difficulty.

Domain area of knowledge or expertise.

Domain construction environment a graphical domain definition environment which provides tools for domain and task model definition, model testing, code validation and generation, and the production of domain interface templates.

Domain knowledge a representation of the subject matter.

Domain layer an interface communication layer separating domain dependent and domain independent components.

Domain model a subset of the total domain knowledge. A description of the domain environment.

End-user the target user of a system.

Enumerative bug library a list of all the bugs in some domain, or in some cases all the bugs based on some targeted misconceptions.

ESS see *expert system shell.*

Expert knowledge knowledge from some domain that an expert in that domain would know.

Expert model a model of expert knowledge.

Expert system computer system that applies reasoning methodologies on knowledge in a specific domain in order to render advice, much like a human expert. Consists minimally of a knowledge base, an inference engine and a user interface.

Expert system shell a complete expert system stripped of its specific knowledge consisting of a user interface, an inference engine and a structure for handling a knowledge base, but which is empty.

Exploration based learning a learning approach where learners are encouraged to navigate their own route through an environment.

Extreme guidance a teaching strategy where SCRs and projection graphs are used to force the user to an optimal path.

Format standardisation layer a layer between the domain and task models and the GDI which formalises their communication.

GDI see *general domain interpreter.*

General domain interpreter a domain and platform independent tutoring engine that can be used to provide a discovery learning based tutoring environment for teaching procedural skills.

General knowledge the domain independent knowledge for a system.

Generative bug library library based on generating and explaining bugs based on some cognitive model.

Generative CAI CAI systems designed for situations requiring repetitive practise of a skill.

Generic a generic package provides a template from which a particular instatiation can be produced by providing the appropriate parameters.

Generic tutoring engine a general tutoring engine which can be reused over several domains or ITS developments.

Goal a collection of conditions that are satisfied in the current situation of an environment.

High order operations guidance strategy a guidance approach where high order operations are allowed after primitive operations have been mastered by the user.

Hypercard software for the Apple Macintosh which implements hypertext and hypermedia concepts.

Hypermedia software which links differing data formats in a non-linear fashion. Text, graphics, sound and animation can be interrelated in a network of information.

Hypertext text with cross-reference links among words which allow for non-sequential reading.

ICAIS see *intelligent computer aided instruction system.*

Ill-defined domains domains which are incompletely specified either in terms of initial state, transitions or final state.

Inessential condition a condition which can be derived from other conditions.

Instructional knowledge see *teaching strategy.*

Intelligent computer aided instruction system see *intelligent tutoring system.*

Intelligent tutoring system tutoring system which combines CAI with techniques from artificial intelligence.

Interface layer communication layer between an interface and a system.

Interface link the connection between an interface and the processing component of a system.

Impossible action an action which is valid in a domain but due to some circumstance is no longer possible.

Iterative deepening a search technique where a depth-limited search is executed iteratively, increasing the depth limit, until a solution is found.

ITS see *intelligent tutoring system.*

ITSAT see *ITS authoring tool.*

ITS authoring tool an ITS development tool typically consisting of a user interface, an tutoring engine, several teaching strategies and a structure for handling domain knowledge.

ITS shell a complete ITS stripped of its domain knowledge. Typically consisting of a user interface, a tutoring engine and a structure for handling domain knowledge.

Knowledge acquisition see *knowledge elicitation.*

Knowledge base collection of facts, rules and procedures organised into schemas. The assembly of all of the information and knowledge of a specific field of interest.

Knowledge-based system a system which performs a task by applying heuristics to a symbolic representation of knowledge.

Knowledge elicitation the process of extracting and storing knowledge from some domain area. Usually from a domain expert.

Knowledge engineer AI specialist responsible for the technical side of developing an expert system. The knowledge engineer works closely with the domain expert to capture the expert's knowledge in a knowledge base.

Knowledge representation a formalism for representing, in the computer, facts and rules about a subject or a speciality.

Knowledge representation layer communication layer between the domain knowledge and domain independent components of a system, eg. the tutoring engine and the teaching strategies.

Knowledge representation techniques techniques which can be used to formally represent information.

Learning by doing an advisory design approach where the user can freely initiate actions which are compared to an expert's moves. Feedback is provided to help the user move closer to the expert's approach.

Limited guidance a teaching strategy where SCRs are used to force users to take correct paths at critical choice points.

Maintenance static axiom axioms which are used to provide consistency in a current situation after an operator has been executed.

Mental model the model of a system or procedure that the user/student currently has.

Multimedia several human-machine communication media (eg. voice, text).

Natural language processing computer systems that analyse, attempt to understand or produce one or more human languages.

NLP see *natural language processing*.

Optimal path guidance strategy a guidance strategy where the user is forced to stay on an optimal path when completing a task.

Overlay modelling a student modelling technique which assumes that all differences between the student's behaviour and the behaviour of an expert model can be explained by the lack of skill or skills on the part of the student. Thus the student model is considered a subset of the domain model.

Perturbation modelling a student modelling technique which allows the student model to be extended beyond the range of the expert model.

Plan net a bipartite directed graph built from two types of nodes: condition nodes (circles) which are facts about an environment and operator nodes (squares) which denote events. Arcs between nodes describe relations of causation and enablement (Drummond, 1989).

Planning sub-field of artificial intelligence which involves reasoning about the effects of actions and the sequencing of available actions to achieve a given cumulative effect.

Platform dependent components of a system which are dependent on the platform of their implementation. For example, a user interface.

Platform independent components of a system which are independent of their implementation platform. For example, knowledge representation technique.

Platform layer an interface communication layer separating platform dependent and platform independent system components.

POP table see *precondition/operator/postcondition table.*

Postcondition a condition that is added to the current situation after the use of an operator.

Precondition a condition that is required in the current situation before an operator can be enabled.

Precondition/operator/postcondition table a tabular representation of domain information based on operator enablement and causation.

Premature action an action whose preconditions are currently not all enabled.

Procedural knowledge a collection of actions or procedures that an intelligent system can carry out.

Procedural net a graph structure whose nodes represent actions, organised into a hierarchy of partially ordered time sequences (Sacerdoti, 1977).

Projection graph graph structure displaying a temporal projection over a state space.

Prolog PROgramming in LOGic. High-level computer language designed around the concepts of predicate calculus.

Prototyping strategy in system development in which a scaled down system or portion of a system is constructed, tested and improved in several iterations.

Rapid prototyping a technique for quick development of an initial version of a system to test the effectiveness of the overall design.

Reconstructive bug library a bug library which involves reconstructing errors from observed behaviour and the process of trying to identify the misconception which led to the error.

Redundant action an action which produces an effect that is already in place in the current situation.

Reusable components components which are reusable across several implementations or developments.

SCR see *situated control rule.*

Semantic network knowledge representation formalism consisting of a network of nodes, standing for concepts or objects, connected by arcs describing the relations between the nodes.

Sequential processing traditional computer processing technique of performing actions one at a time in a sequence.

Sharable knowledge bases knowledge bases which can be shared across several implementations or developments.

Simulation a model that behaves or operates like a given system when provided a set of controlled inputs. An implementation of a special kind of model that represents at least some key internal elements of a system and describes how those elements interact over time.

Situated control rule tutor defined rules for providing local guidance in a simulated environment.

Specific knowledge domain or implementational dependent knowledge that is required by a system.

Standard static axiom an axiom that describes the conditions that are needed to derive an inessential condition.

Static axioms axioms that define relations in a domain that are assumed to be always satisfied (Tenenberg, 1991).

Static world assumptions several observations defined by Sanborn and Hendler (1998) to limit the modelling of dynamic real world models.

STM see *student task model.*

Student model representation of the current state of the student's knowledge of a subject matter.

Student task model a tree hierarchy of goals and sub-goals for a particular task.

Sub-domain a subset of domain knowledge.

Sub-domain decomposition the process of splitting the domain knowledge into sub-domains.

SWAs see *static world assumptions*

System interface an interface for user/system interaction.

TANDEM see *Task ANd Domain Environment Model.*

Task an activity or sequence of activities that may be carried out to achieve some goal.

Task ANd Domain Environment Model an authoring environment for the construction of procedural task computer based tutoring systems.

Task model a model of tasks for a domain. Typically represented with a series of goals and sub-goals for tasks.

Teaching strategy the instructional strategy used to teach the current subject matter. The way the tutoring system controls the overall interaction with the user.

Toolkit library of reusable components.

Tutoring engine control component of an ITS. Typically used to enforce the teaching strategy of a system.

Tutoring strategy see *teaching strategy*.

UNIX operating system originally for minicomputers, now on microcomputers, noted for its capacity to execute multiple tasks simultaneously.

User-friendly term used to describe a facility designed to make interaction with a computer system easy and comfortable for the user.

User interface component of a computer system that allows bi-directional communication between the system and its user.

User models representations which model the users of a system. Typically student and expert models.

Appendix A

Projection Graphs In TANDEM

A.1 Introduction

The generation of large projection graphs is computationally expensive and very time consuming. This problem is documented in Smith and Kemp (1995). Another problem is that when presenting projection graphs to a user so that they can select paths through the graph, displaying the graph clearly is difficult.

There are two main problems that need to be considered before projection graphs can be easily presented to users; node placement on screen and complete graph presentation. Solutions to these problems have been developed and incorporated into the current implementation of TANDEM.

A.2 Node Placement

The placement of graph nodes is important when it comes to displaying their transitions. If an unfortunate choice of node placement is taken then there may be many transitions crossing nodes. This is undesirable visually and can make node labelling very ugly.

The problem reduces to how nodes can be placed so that there is a minimum of transitions crossing nodes. In the current work, a simple approach is taken. This is to plot the nodes around a circle. If nodes are

plotted around of circle, then there are virtually no transitions crossing nodes (Figure A.1).

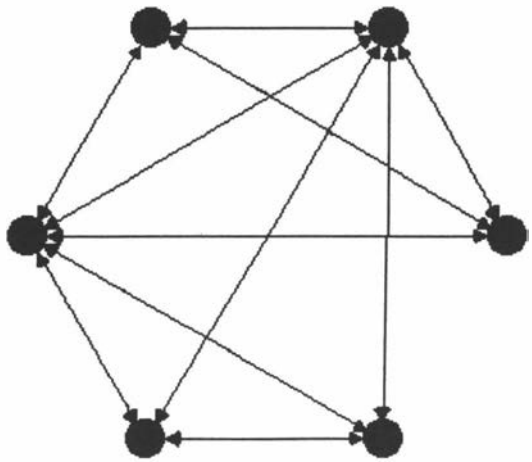


Figure A.1 : Plotting nodes around a circle.

This approach also has the advantage that node labels can be placed around the outside of the circle and are not covered by transitions (Figure A.2).

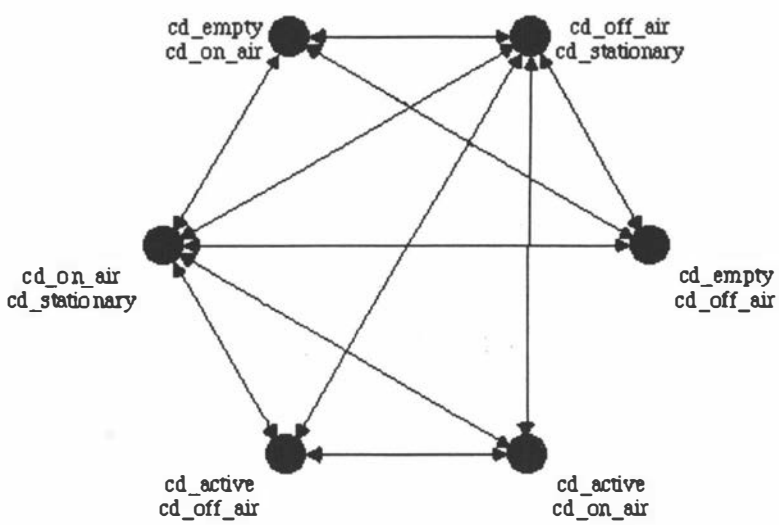


Figure A.2 : Labelling nodes around a circle.

A.3 Iterative Deepening

The second problem is that if a projection graph is generated with 20 nodes and 60+ transitions, displaying this graph on a standard size

monitor is not possible. For domains with a large number of independent operators, projection graphs in excess of this are not unusual. Reducing the size of the graph and supplying a zoom-in feature is one possible solution but this can make it difficult to manipulate the whole graph when marking paths for SCR generation.

The original projection graph algorithm (Drummond, 1989) presents a depth-first type algorithm for projection graph generation. This is acceptable if the whole projection graph is to be generated but is not optimal if only part of a projection graph is required. Typical projection graphs are generated from a start situation and then projected out from there. In many cases, only one or two levels of projection are required before critical choice points in the state space can be identified. In these cases, generating the complete projection graph is unnecessary.

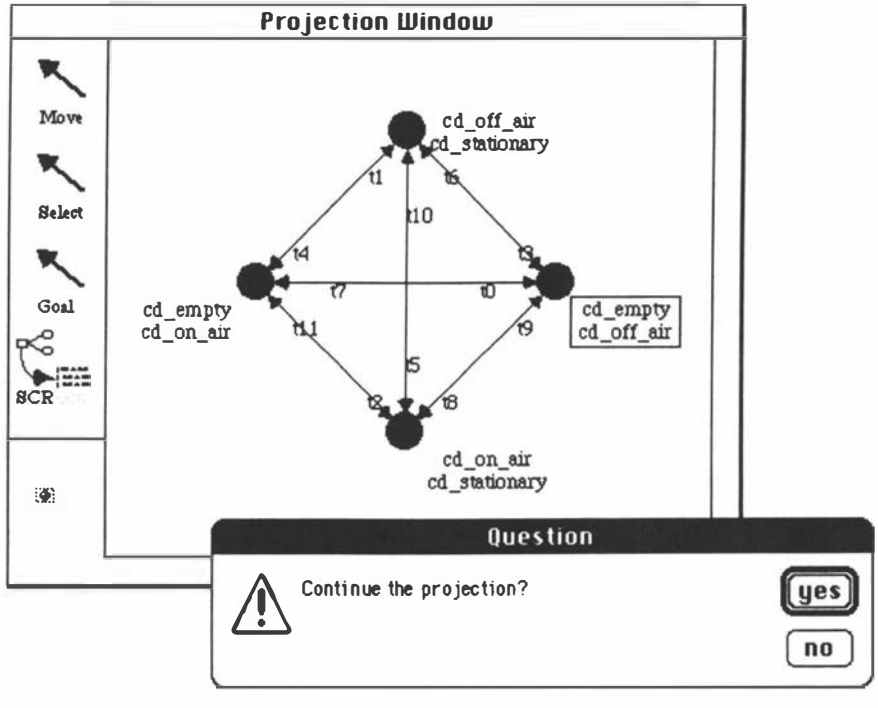


Figure A.3 : First level projection graph.

In the current work, an iterative deepening approach (Bratko, 1990) has been adopted. Increasingly deepening levels of projection, from a start situation, are generated and presented to the user. If the required depth has been reached, then the projection process can be terminated and the definition of SCRs begun. If more levels of projection are required, the process can continue to the next level. Figures A.3 and A.4 show a first

level projection graph and the next, and final, projection graph respectively.

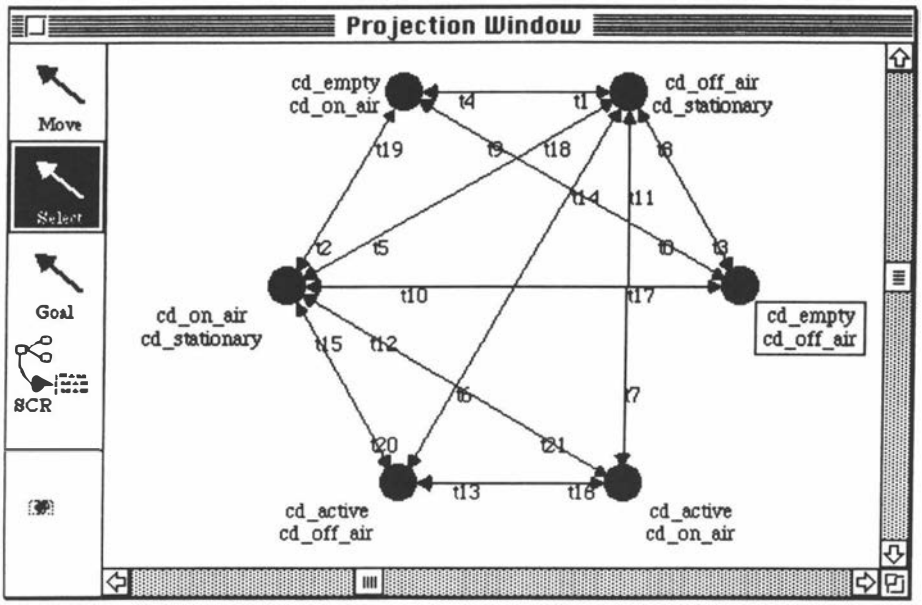


Figure A.4 : Final projection graph.

A.4 Transition Selection

Even with the nodes placed around a circle and the use of iterative deepening to reduce the size and complexity of the displayed graphs, as seen in Figure A.4, they are still complex for even small examples. One question that can be asked of these displayed graph is whether or not all the presented information is necessary?

During the process of SCR definition, the user is initially interested in indicating critical choice points and then choosing appropriate paths from these points. Therefore, at the first stage, only the nodes are required. Displaying all the transitions on the graph only complicates the display.

In the current implementation of TANDEM, initially, only the nodes of a projection graph are presented to the user (Figure A.5).

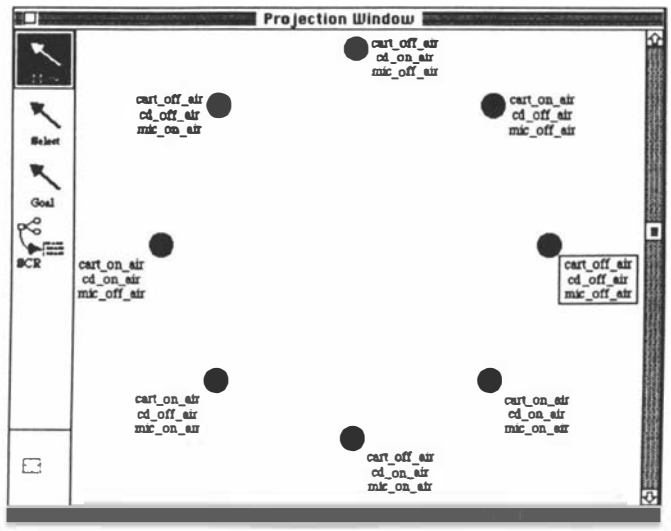


Figure A.5 : Nodes of a projection graph.

The user can examine the nodes presented and determine which are important and which should be avoided. The *select* tool can be used to select the nodes that would be appropriate for path finding in SCR generation. Once a node is selected, all the transitions that end on that node are added to the projection graph (Figure A.6).

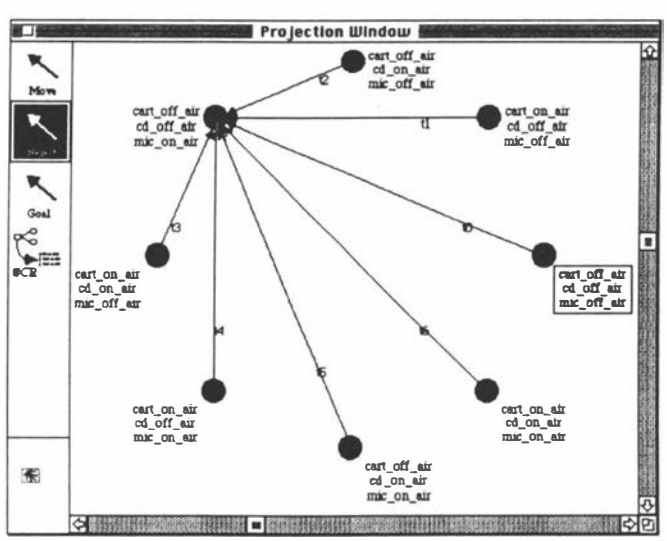


Figure A.6 : Selecting an appropriate node.

Thus, the user only selects the nodes that are required in the current example. As only the visible transitions are used for path finding in the generation of SCRs, this approach allows the user to define paths that avoid undesirable nodes. In the example in Figure A.7, from the radio

studio domain, it is undesirable to have two devices on-air at the same time, so only nodes without two devices on are selected. This method helps reduce the amount of redundant information displayed on projection graphs.

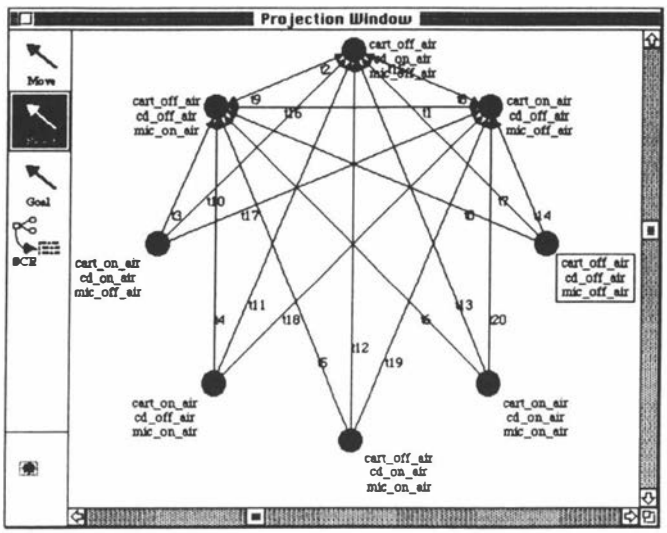


Figure A.7 : The final projection graph.

A.5 Generating Projection Graphs

The generation of projection graphs is based on the work of Drummond (1989). His original algorithm can be seen in Figure A.8.

```
project(N,S,X,G) {
   if X > 0 the {
      A = free_from_interference(S,N);
      for P in A {
         S1 = apply(P,S);
         if S1 is not in G then {
            G = add_case(S1,G);
            G = add_labelled_arc(S,S1,G);
            project(N,S1,X-1,G);
         }
         else G = add_labelled_arc(S,S1,P,G);
      }
   }
   return(G);
}

Key
===

N = plan net
S = initial case
X = depth cutoff indicator
G = initial projection graph (which contains S)
```

Figure A.8 : The plan net projection algorithm (Drummond, 1989).

There are three main steps in this algorithm. Firstly, all the operators that apply in the initial case situation (S) are found that are free from interference. A set of operators are free from interference if these operators can be applied in any order without changing the final outcome of the situation. Thus, the operators are totally independent of the other operators in the set.

Secondly, the free sets (A) are applied, set by set, to the current situation (S), generating a new situation (S1). Finally, if the new situation (S1) is not present in the current projection graph, it is added as a node and transition pair and a new projection is begun in a depth-first manner. If the new situation is present in the current graph, only the relevant transition is added and the next free set is applied.

This process terminates when no new situations are added to the projection or if a predefined cut-off level (X) is reached. As noted by Drummond (1989), the projection algorithm performs a chronologically organised search through a space of world state descriptions.

In the current work, the basic structure of the projection algorithm remains, although it has been modified to allow the iterative deepening approach described early.

At (1) in Figure A.9, the second parameter, the new situation, is empty, therefore the projection is complete. At (2), the depth (parameter three) has reached zero, so this level of projection is over. The next two routines differ slightly in that (3) is used to initialise the iterative deepening of a level by generating the current cut-off point by counting the nodes at the current level. (4) is the general projection routine which places new situations at the tail of a list and processes the head situations until the current levels cut-off point is reached. The predicate that calls *project* can then recall it with the new situation set (parameter five in (1) and (2)) to (3) which can initialise a new, deeper projection.

```
   project(T, C, X, O, N) {
(1)  if C = [] {
        return(N)
        }
(2)  else if X = 0 {
        return([C|N])
        }
(3)  else if T = first {
        C = [S|L];
        A = free_from_interference(S,O);
        New_C = get_new_nodes(A,C,O,N);
        New_X = count_sits(New_C);
        project(many,New_C,New_X,N,New_N)
        }
(4)  else if T = many {
        C = [S|L];
        A = free_from_interference(S,O);
        New_C = get_new_nodes(A,C,O,N);
        New_X = X - 1;
        project(many,New_C,New_X,N,New_N)
        }
     return(New_N);
     }

Key
===

T = type of projection
C = current list of situations
S = current situation
L = list of situations
X = depth cutoff indicator
O = old plan net
N = next plan net
```

Figure A.9 : TANDEMs projection algorithm.

A.6 Summary

This appendix has considered the approaches that have been used in the current implementation of TANDEM to manipulate projection graphs. This includes techniques to aid in the viewing of these graphs and in their generation.

Appendix B

DCE Generated Files

B.1 Introduction

In TANDEM, domain and task models are defined by users in the Domain Construction Environment (DCE). The result of this process are data files that are used as input to the tutoring component of TANDEM, the General Domain Interpreter (GDI). The GDI is kept domain independent by isolating it from the domains that it teaches about. This is done by standardising the format of the DCE generated files and providing an domain/platform layer between the GDI and the student interface (Figure B.1).

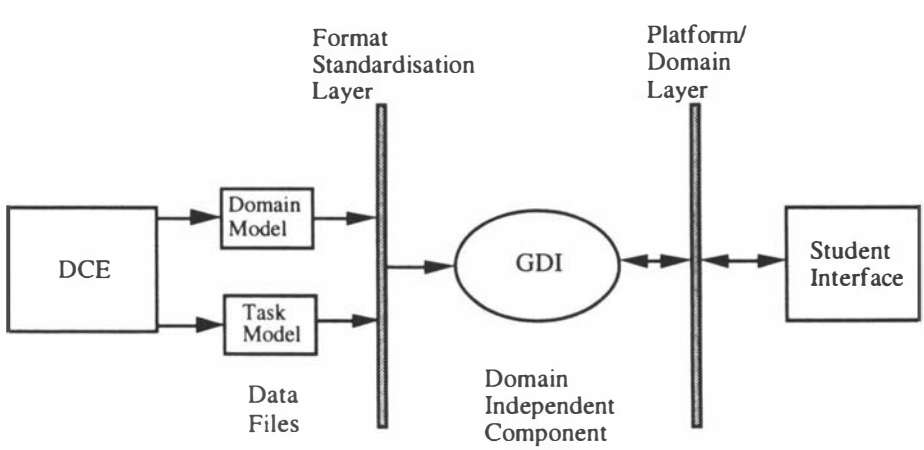


Figure B.1 : Providing GDI domain independence.

The DCE generates four data files for every domain that is developed in TANDEM. These are the domain model (the POP table and static axiom definitions), the task model (the task hierarchy and SCRs), a layout file (a definition of the graphical objects from the DCE layout) and an interface file (code stubs for GDI and student interface interaction).

This appendix will present the format of these files with an example from the car maintenance domain. Both the DCE and GDI use a dummy (empty) file for identifying a domain. The other files can then be accessed but their manipulation is transparent to the user. These files can be seen in Figure B.2.

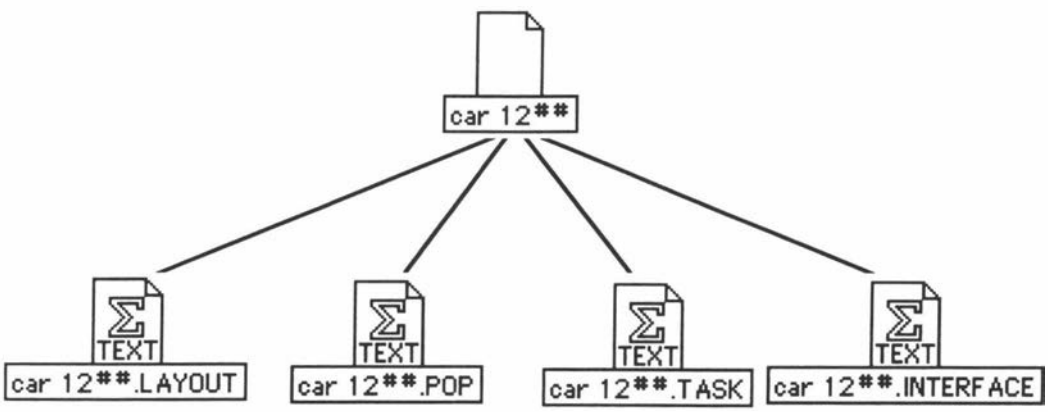


Figure B.2 : DCE generated files.

B.2 The Layout File

The DCE is a graphical intensive environment. The layout file is used to save the relations between graphical objects, the windows they are drawn on and any other domain related information to needs to be saved between sessions.

Figure B.3 shows a portion of the domain definition for the car maintenance domain. Figure B.4 is the corresponding entry from the layout file.

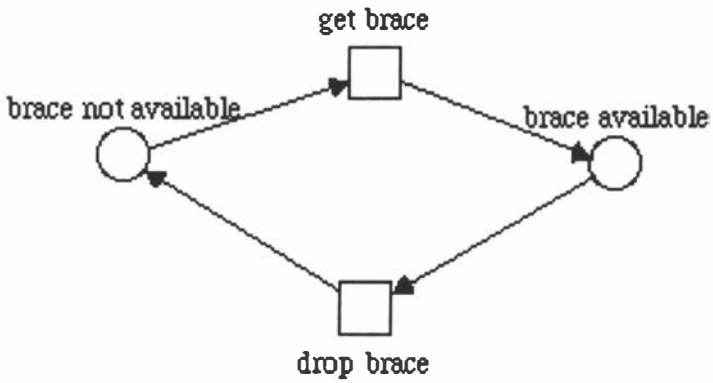


Figure B.3 : Car domain layout example.

```
(1) ['wheel brace', 65, 50, 385, 553, 400, 500, 0].
(2) ['wheel brace', action_object6, [square(86, 140, 20),
       textline('Times', 12, 0, 61, 119, 'get brace')], (-3, 0), 0].
    ['wheel brace', action_object7, [square(158, 137, 20),
       textline('Times', 12, 0, 173, 112, 'drop brace')], (11, 0), 0].
(3) ['wheel brace', condition_object10, [circle(113, 47, 10),
       textline('Times', 12, 0, 91, 4, 'brace not available')], (0, 0), 0].
    ['wheel brace', condition_object11, [circle(116, 229, 10),
       textline('Times', 12, 0, 94, 195, 'brace available')], (0, 0), 0].
    ['wheel brace', transition_object14, pointer((86, 150), (113, 220),
       right), (0, 0), 0].
    ['wheel brace', transition_object13, pointer((110, 56), (86, 130),
       right), (0, 0), 0].
    ['wheel brace', transition_object16, pointer((162, 127), (118, 55),
       right), (0, 0), 0].
    ['wheel brace', transition_object15, pointer((120, 221), (163, 147),
       right), (0, 0), 0].
    end_of_window.
```

Figure B.4 : Layout file code example.

In Figure B.4, (1) is the window definition and (2) and (3) are the action and condition object descriptions respectively.

Also, when a large domain is being defined over several definition sessions, it is typical that the author will not require that the full model be generated, with all its associated files, each time the domain is saved. Thus, TANDEM provides a 'Quicksave' option that only saves the layout information for the current domain (Figure B.5). (NB. the custom save option will be discussed in section B.5).

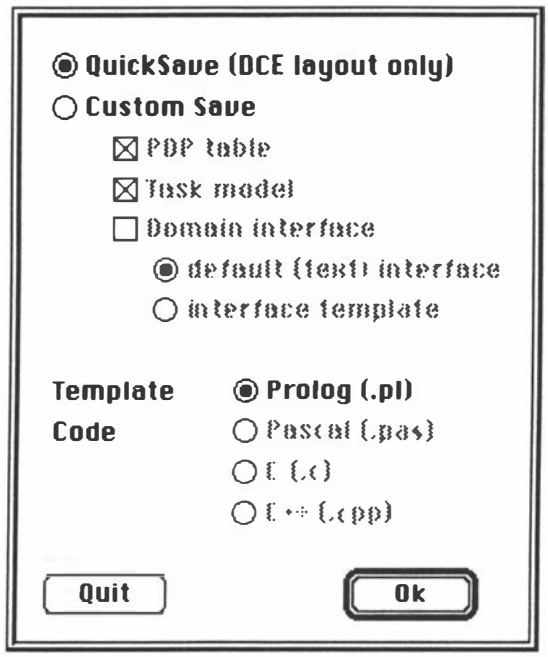


Figure B.5 : Selecting the QuickSave option.

B.3 The POP File

The POP file contains the Prolog version of the POP table that is generated from the graphical display from the DCE. This is made up of five main predicates. Three which are generated from the action/condition objects in the DCE are *operator, precondition* and *postcondition*. The *initial situation* predicates are defined in their own window in the DCE, as are the relevant static axiom definitions. A portion of the POP corresponding to objects in Figure B.3 can be seen in Figure B.6.

```
operator(get_brace).
operator(drop_brace).

precondition(get_brace,[brace_not_available]).
precondition(drop_brace,[brace_available]).

postcondition(get_brace,[brace_available]).
postcondition(drop_brace,[brace_not_available]).

initial_situation(brace_not_available).

static_axiom([brace_not_available then ~(brace_available)]).
static_axiom([brace_on_wheel then ~(brace_available)]).
static_axiom([brace_available then ~(brace_not_available) and
   ~(brace_on_wheel)]).
```

Figure B.6 : POP table code example.

B.4 The Task File

Similar to the POP table definitions, the task rules and SCRs are generated from any rules that have been defined by the author. Figures B.7 and B.8 show examples of tasks and SCRs that have been defined for the current example.

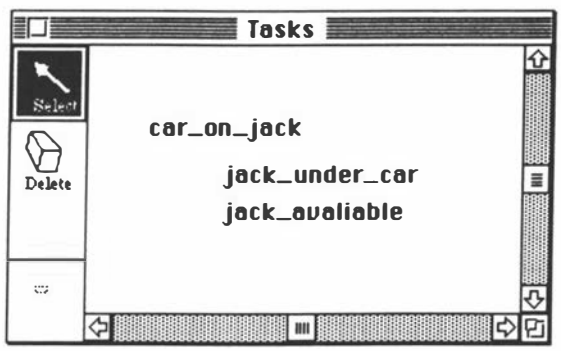


Figure B.7 : Displayed tasks example.

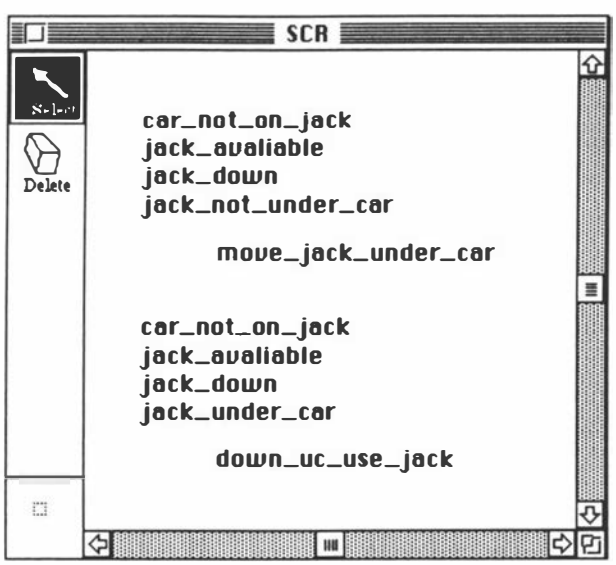


Figure B.8 : Displayed SCRs example.

These definitions can then be used to generate the *task* and *scr* predicates that are stored in the task file (see Figure B.9).

```
task([car_on_jack], [jack_under_car, jack_avaliable]).

scr([car_not_on_jack, jack_avaliable, jack_down,
   jack_not_under_car], [move_jack_under_car]).
scr([car_not_on_jack, jack_avaliable, jack_down,
   jack_under_car], [down_uc_use_jack]).
```

Figure B.9 : Example task file.

B.5 The Interface File

The interface file allows the GDI to determine whether the default text interface is to be used or whether a custom interface is going to be used. When a custom save is chosen by an author (Figure B.10), either code for a default text interface can be generated or code stubs for a custom interface template can be saved.

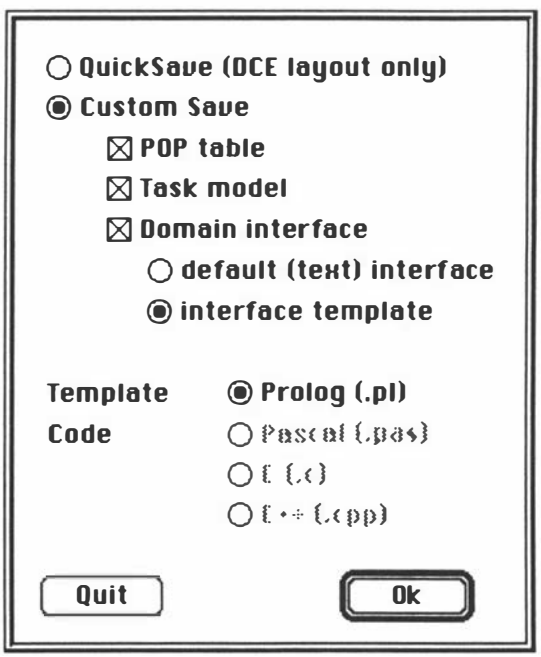


Figure B.10 : Saving custom interface code.

If the default interface option is chosen, then the following code is generated for the domain interface (Figure B.11).

```
domain_interface('car 12d##', default).

domain_event_loop(default).
initialise_interface(default).
deinitialise_interface(default).
```

Figure B.11 : Default interface code.

This code basically allows the GDI to retain control of the GDI session and utilise its own built in text interface to interact with the user.

```
domain_interface('car 12##', 'car 12##').

domain_event_loop('car 12##').
initialise_interface('car 12##').
deinitialise_interface('car 12##').

interface_event(Event,Object,Return) :-
   call_gdi(d_use_jack,Return).
interface_event(Event,Object,Return) :-
   call_gdi(u_use_jack,Return).

process_gdi_return(Event,Parameter1,[valid_op,Op],
   Parameter2).
```

Figure B.12 : DCE generated interface code.

If a custom interface is required, then code stubs for every operator in that domain are generated for calls to and from the GDI. These are handled by the *call_gdi* and *process_gdi_return* predicates respectively. Figure B.12 shows a portion of the raw code that is generated by the DCE for the car maintenance domain and Figure B.13 is the current graphical interface's code.

When the *initialise_interface* predicate at (1) in Figure B.13 is called from the GDI, control of the session is passed to the domain interface. It draws the interface window and all the relevant objects in this domain. The session is now event driven, with interface events then activating GDI calls; see (4). At (5) return values from the current action are returned from the GDI. When a session is complete, control must be passed back to the GDI. In this interface that is done at (3). When the main window is closed, control is passed back to the GDI. The *deinitialise_interface* predicate at (2) is not needed as this particular interface deinitialises itself when its window is closed at (3).

```
(1) initialise_interface('car 9##') :-
    del_all_props,
    car_window(Win,Top,Left,Depth,Width,Split,Maxx,Maxy,Vis,Go),
    create_car_window(Win,Top,Left,Depth,Width,Split,Maxx,Maxy,Vis,Go),
    draw_road(Win,start),
    draw_car(Win,start),
    draw_jack(Win,start),
    draw_spare_tire(Win,start),
    draw_wheel_brace(Win,start).

(2) deinitialise_interface('car 9##').

(3) car_win(close,Win) :- wkill(Win),call_gdi(quit_gdi,[ok]).

(4) interface_event(use,(Win,[jack_pic]),Return) :-
       get_pic(Win,jack_pic,jack(Y,X,down)),
       call_gdi(d_use_jack,Return).
    interface_event(use,(Win,[jack_pic]),Return) :-
       get_pic(Win,jack_pic,jack(Y,X,up)),
       call_gdi(u_use_jack,Return).

    interface_event(at_location,(Win,jack_pic),Return) :-
       call_gdi(place_jack_under_car,Return).

(5) process_gdi_return(use,[jack_pic],[valid_op|Rest],[Win]) :-
       draw_items(Win,Rest),
       check_location(Win,Name).
```

Figure B.13 : Car domain custom interface code.

B.6 Summary

This appendix has discussed the four files that are generated in TANDEM by the DCE for use by the GDI and the development of user interfaces. Examples of these files from the car maintenance domain have been presented.

Appendix C

TANDEM Domain and Platform Drivers

C.1 Introduction

The tutoring engine of TANDEM, the General Domain Interpreter (GDI), has been kept domain independent from the domains it teaches about. One way this is achieved is by the use of platform and domain drivers to keep the GDI separate from the student user interface (Figure C.1).

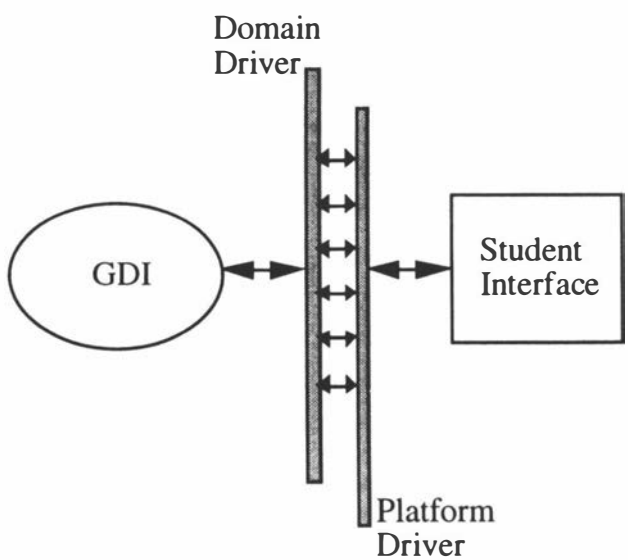


Figure C.1 : Platform and domain drivers.

This way, GDI code can remain platform and domain independent. This is done for two main reasons: firstly, to ease moving TANDEM to an alternative platform if required and secondly, domain independence allows more flexibility in the choice of user interfaces that can be developed.

This appendix presents the code from the current implementation of TANDEM using Prolog on a Power Macintosh. Firstly, the GDI control unit, which controls a GDI session will be described. This will be followed by portions of code from the platform and domain drivers and example Macintosh drivers that are provided with the current version of TANDEM.

C.2 GDI Control Unit

```
(1) gdi :-
      get_gdi_info(User_Name,[Domain,Interface,Task],Strategy),
      begin_gdi_session(User_Name,[Domain,Interface,Task],Strategy).
(2) gdi :-
       deinitialise_domain_interface.

(3) begin_gdi_session(User_Name,[quit,_,_],Strategy).
(4) begin_gdi_session(User_Name,[Domain,Interface,Task],Strategy) :-
      \+(Domain = quit),
      initialise_new_session(User_Name,Strategy),
      initialise_domain([Domain,Interface,Task]),
      start_domain_interface.

(5) quit_gdi_session :-
      deinitialise_domain_interface,
      gdi.
```

Figure C.2 : GDI control unit code.

(1) Start of a GDI session requires user name, domain name, interface type, task hierarchy and session strategy.

(2) If the GDI session fails, the domain interface may need to be cleaned up and terminated.

(3) If quitting a GDI session, there is no need to begin a new one.

(4) Start a new session, initialise the domain in the GDI and then pass session control to the domain interface.

(5) At a formal exit from a session, terminate the domain interface.

C.3 Platform Driver

```
(6) platform(powermac).

/* gdi start information for Power Macintosh default interface*/

(7) get_gdi_info(User_Name,[Domain,Interface,Task],Strategy) :-
      get_gdi_info_platform(User_Name,[Domain,Interface,Task],Strategy).

(8) get_gdi_info(User_Name,[quit,_,_],Strategy).
```

Figure C.3 : Platform driver code.

(6) Current platform identification.

(7) Request GDI information from the current platform driver.

(8) If quitting, do nothing.

C.4 Power Macintosh Platform Driver

```
(9) strategies(['Free Roam','Task - Hints','Task - Advice',
        'Task - Limited', 'Task - Extreme']).

(10) get_gdi_info_platform(User,[Domain,Interface,Task],Strategy) :-
       remember(domain_file,'No Domain'),
       strategies(S),
       dialog('GDI Information',50,150,175,230,
         [button(142,157,26,66,'Ok'),
          button(145,10,20,60,'Quit'),
          text(40,75,20,100,'User Name'),
          edit(15,20,20,190,'',User),
          popup(110,5,20,215,'Strategy :',S,
            'Free Roam',Strategy),
          text(80,80,20,100,'No Domain'),
          button(80,10,20,60,'Domain')
         ],Btn,check_gdi_info_button(Domain,Interface,Task)).

(11) check_gdi_info_button(D,7,Domain,Interface,Task) :-
       old('DCEF',File),
       close(File),
       fname(File,Path,Name,Ex),
       fname(POP_File,Path,Name,'.POP'),
       fname(Interface_File,Path,Name,'.INTERFACE'),
       fname(Task_File,Path,Name,'.TASK'),
       setditem(D,6,Name),
       remember(domain_file,POP_File),
       remember(interface_file,Interface_File),
       remember(task_file,Task_File),
       fail.
     check_gdi_info_button(D,1,Domain,Interface,Task) :-
       recall(domain_file,Domain),
       recall(interface_file,Interface),
       recall(task_file,Task).
```

Figure C.4 : Power Macintosh platform driver.

(9) Menu list of current teaching strategies.

(10) Dialog box code requesting GDI user information.

(11) Check that the required GDI interface files are present for the defined domain.

C.5 Domain Driver

```
(12) start_domain_interface :-
       initialise_domain_interface.

(13) initialise_domain_interface :-
       domain_interface(Name,Interface),
       initialise_interface(Interface),
       domain_event_loop(Interface).


(14) deinitialise_domain_interface :-
       domain_interface(Name,Interface),
       deinitialise_interface(Interface).


/* process commands from the event driven engine */

(15) call_gdi(quit_gdi,[ok]) :-
       quit_gdi_session.

(16) call_gdi(Command,Return) :-
       gdi_strategy(Strategy),
       strategy(Strategy,Command,Return).
```

Figure C.5 : Domain driver code.

(12) To start a GDI session, call the appropriate domain interface.

(13) Initialise the domain and start the domain event loop for processing domain events.

(14) Deinitialise the domain interface.

(15) Inform the GDI engine when a session is complete.

(16) Standard call to the GDI with the current strategy from a domain event.

C.6 Power Macintosh Default Text Interface

```
(17) domain_event_loop(default) :-
       get_command(Input),
       call_gdi(Input,Return),
       write_interface('Command : '),
       write_interface(Input),
       write_nl(1),
       return_from_gdi(Return),
       domain_event_loop(default).
     domain_event_loop(default) :-
       call_gdi(quit_gdi,[ok]).

(18) output_window('Default Text Output', 1,250,30,200,400,0).

(19) initialise_interface(default) :-
       output_window(Name, Vis, Top,Left,Depth,Width,Go),
       wcreate(Name, Vis, Top,Left,Depth,Width,Go).

(20) deinitialise_interface(default) :-
       output_window(Name, Vis, Top,Left,Depth,Width,Go),
       wkill(Name).

(21) get_command(Input) :-
       see(user),
       read(Input),
       seen.
```

Figure C.6 : Power Macintosh text interface A.

(17) Event loop for the textual interface. Gets input from the user, sends the command to the GDI engine and then returns any feedback to the user.

(18) Data for user feedback window.

(19) Creating the user feedback window.

(20) Deleting the user feedback window.

(21) Requesting user GDI commands via a Prolog read dialog box.

```
return_from_gdi([scr_advice,Scr_Ops,Return]) :-
   write_interface('At this point a good command would have been :'),
   write_list(Scr_Ops),
   return_from_gdi(Return).

return_from_gdi([scr_limited,Scr_Ops,Return]) :-
   write_interface('At this point a better command would be :'),
   write_list(Scr_Ops).

return_from_gdi([invalid_op,Op]) :-
   write_interface('You can not do that').

return_from_gdi([premature_op,Op]) :-
   write_interface('That can not be done at the moment').

return_from_gdi([valid_op,Op]) :-
   write_interface('Ok').

return_from_gdi([help, Help_List]) :-
   write_interface('The following commands are currently valid : '),
   write_list(Help_List).
```

Figure C.7 : Power Macintosh text interface B.

In Figure C.7, several examples of the *return_from_gdi* predicate are presented. These predicates are used to process the parameters that are returned from the GDI engine after a GDI processed command.

C.7 Summary

This appendix has presented portions of GDI code that are used in the current version of TANDEM. More specifically, the code that is used to maintain platform and domain independence for the GDI.

Appendix D

Domain Complexity

D.1 Introduction

In this appendix the transition network representation of domains is compared with the POP table representation (Smith and Kemp, 1995).

D.2 Domain Complexity

Before the two domain representation techniques can be compared, measures of complexity must be defined for the transition network so that accurate results can be generated about their construction.

The following notations will be used for describing the transition networks (or graphs, as they will be called in this appendix). G(n:t) is a graph with n nodes and t transitions between nodes, where graphs can be one-way or two-way directed (see Figure D.1). Also, the word 'transition' indicates an entry in a POP table, eg. a graph of size (2:1).

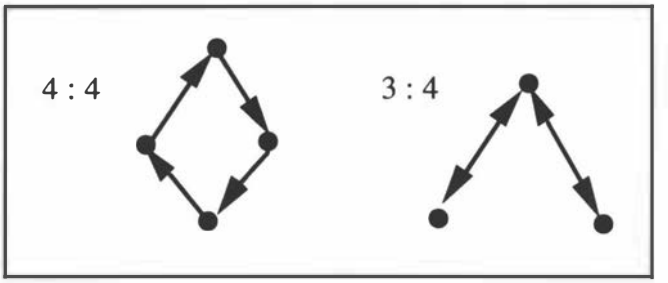


Figure D.1 : Initial graph examples.

Taking an existing graph and adding a new transition that is independent of the current graph will be the first measure to be developed. This would be the case if a graph was generated from a POP table and then a new independent entry[1] was added. It is interesting to see how the new entry effects the size/complexity of the existing graph. An example of this can be seen in Figure D.2.

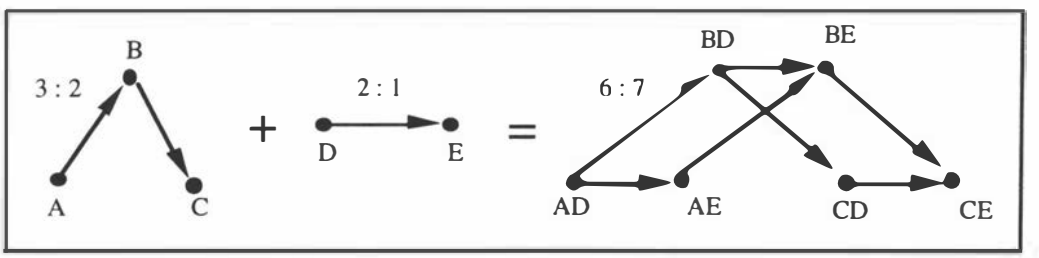


Figure D.2 : Adding an independent transition.

When adding an independent transition to an existing directed graph, it is hypothesised that the size of the combined graph can be generated by the following formulae.

D.3 Complexity Formula 1a

For a one-way directed graph $G_1(n_1{:}t_1)$ where n_1 is the number of nodes in G_1 and t_1 is the number of transitions in G_1, after the addition of an independent transition of size (2:1), the new graph $G_2(n_2{:}t_2)$ can be generated with the following formula.

---

[1] A new entry is independent of an existing POP table if none of its preconditions can be determined by the postconditions of the other entries in the table.

n_2 nodes = $2n_1$ nodes.

t_2 transitions = $2t_1$ transitions + n_1 nodes.

or

$N_2 = 2N_1$

$T_2 = 2T_1 + N_1$

D.3.1 Proof by Example

Figure D.3 shows the original graph G_1 (graph ABC) and the independent transition DE.

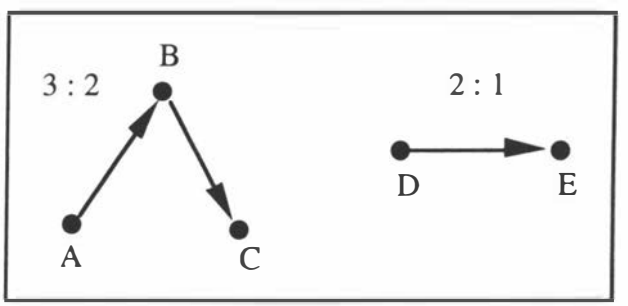


Figure D.3 : The original graphs.

By adding transition DE to G_1, a duplicate of DE is made in every node of G_1 (Figure D.4).

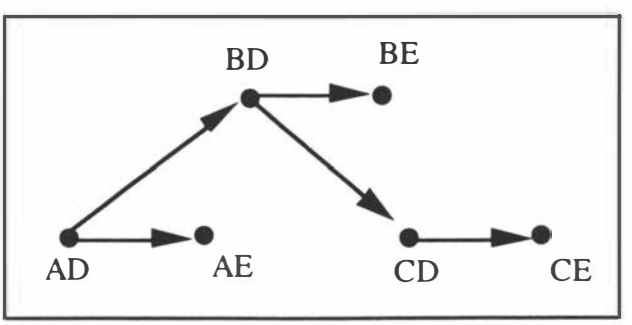


Figure D.4 : Duplication of DE.

As this formula is only concerned with a single transition graph, there is a constant factor of 2 nodes in DE, thus the new graph contains twice the number of nodes as G_1, i.e. $N_2 = 2N_1$.

Then the transitions of G_1 must be preserved in the new graph (Figure D.5).

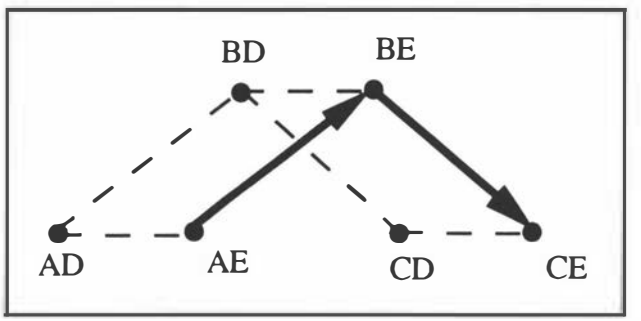


Figure D.5 : The preserved transitions from G_1.

There is also a constant factor of 1 transition in DE for this formula, thus the number of new transitions from the copy of DE into G_1 is the number of nodes in G_1, eg. N_1.

The transitions for the new graph are the transitions from the original graph G_1, plus the transitions added by the preservation of transitions in G_1, eg. $T_1 + T_1 = 2T_1$. Therefore the formula for the transitions of the new graph is $T_2 = 2T_1 + N_1$.

The previous formulae only considers the result of adding one independent transition to an existing graph. Of more interest is the result of adding any number of independent transitions. Thus a general form of the above specific formulae is needed.

A few examples of the specific formula are iterated and a general pattern is searched for.

$$N_{k+1} = 2N_k$$
$$N_{k+2} = 2N_{k+1}$$
$$= 2(2N_k)$$
$$= 4N_k$$
$$N_{k+3} = 8N_k$$
$$N_{k+4} = 16N_k$$
$$\vdots$$
$$N_{k+n} = 2^nN_k \qquad \text{(conjecture general form for nodes)}$$

$T_{k+1} = 2T_k + N_k$

$T_{k+2} = 2T_{k+1} + N_{k+1}$

$= 2(2T_k + N_k) + 2N_k$

$= 4T_k + 2N_k + 2N_k$

$= 4T_k + 4N_k$

$T_{k+3} = 8T_k + 12N_k$

$T_{k+4} = 16T_k + 32N_k$

$T_{k+5} = 32T_k + 80N_k$

:

$T_{k+n} = 2^nT_k + n2^{n-1}N_k$ (conjecture general form for transitions)

Now that the general formulas have been identified, a proof is required so that it can be confirmed that they correctly generate a sequence of new graphs from a given graph.

D.3.2 Proof by Induction for General Node Formula

Specific form : $N_{n+1} = 2N_n$

General form : $N_{n+m} = 2^mN_n$

Base case (Show P_1 is true) :

$m = 1$

$N_{n+1} = 2^1N_n$

$= 2N_n$

Induction hypothesis (Assume P_k is true) :

$N_{n+k} = 2^kN_n$

Induction Step (Verify P_{k+1} is true using assumption P_k is true) :

$N_{n+(k+1)} = N_{(n+k)+1}$

$= 2N_{n+k}$

$= 2(2^kN_n)$

$= 2^{k+1}N_n$

$\Rightarrow N_{n+m} = 2^mN_n$

D.3.3 Proof by Induction for General Transition Formula

Specific form : $T_{n+1} = 2T_n + N_n$

General form : $T_{n+m} = 2^mT_n + m2^{m-1}N_n$

Base case (Show P_1 is true) :

$m=1$

$T_{n+1} = 2^1T_n + 1.2^0N_n$

$= 2T_n + N_n$

Induction hypothesis (Assume P_k is true) :

$T_{n+k} = 2^kT_n + k2^{k-1}N_n$

Induction Step (Verify P_{k+1} is true using assumption P_k is true) :

$$
\begin{aligned}
T_{n+(k+1)} &= T_{(n+k)+1} \\
&= 2T_{n+k} + N_{n+k} \\
&= 2(2^kT_n + k2^{k-1}N_n) + N_{n+k} \\
&= 2^{k+1}T_n + k2^kN_n + N_{n+k} \\
&= 2^{k+1}T_n + k2^kN_n + 2^kN_n \\
&= 2^{k+1}T_n + (k+1)2^kN_n
\end{aligned}
$$

$$
\Rightarrow T_{n+m} = 2^mT_n + m2^{m-1}N_n
$$

A similar approach can be used for two-way directed graphs as can be seen in Complexity Formula 1b.

D.4 Complexity Formula 1b

For a two-way directed graph $G_1(n_1:t_1)$ where n_1 is the number of nodes in G_1 and t_1 is the number of transitions in G_1, after the addition of an independent transition of size (2:1), the new graph $G_2(n_2:t_2)$ can be generated with the following formula :

n_2 nodes = $2n_1$ nodes.

t_2 transitions = $2(t_1$ transitions + n_1 nodes).

or

$N_2 = 2N_1$

$T_2 = 2T_1 + 2N_1$

$= 2(T_1 + N_1)$

General Forms :

$$N_{n+m} = 2^m N_n$$
$$T_{n+m} = 2^m T_n + m2^m N_n$$

D.4.1 Proof

Similar to Complexity Formula 1a proof.

D.4.2 Example

An example from a video cassette recorder (VCR) domain can now be used to show the differing requirements of the two representation techniques.

Part of a VCR domain transition network is shown in Figure D.6 as a graph. The transition labels have been removed for clarity.

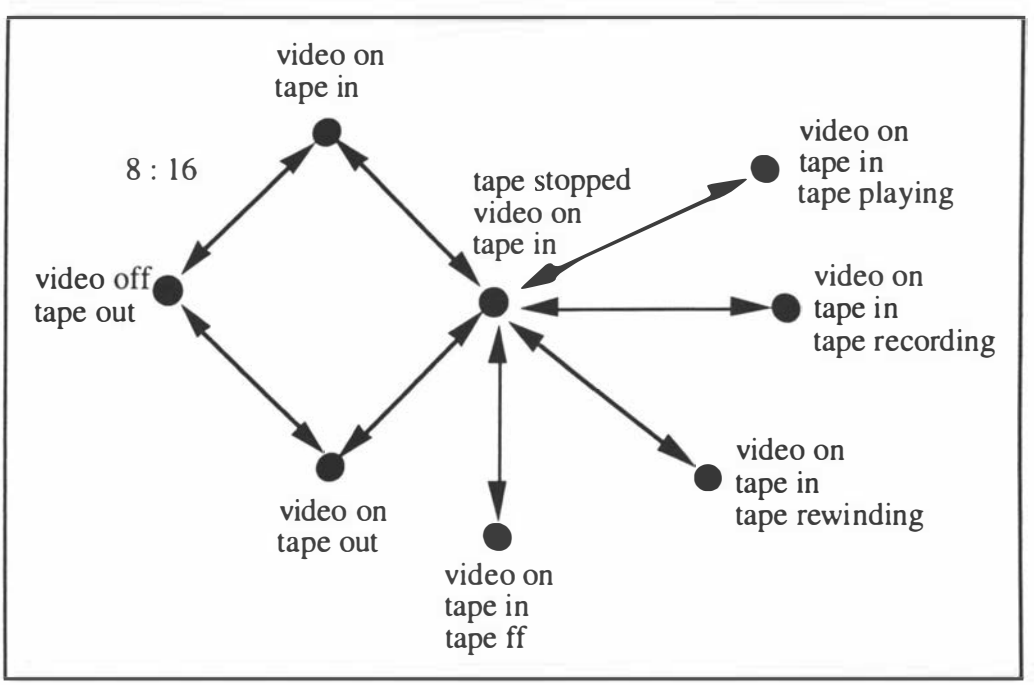


Figure D.6 : Graph of part of a VCR domain.

This domain can also be represented by a POP table with 9 entries. If an independent transition was to be added to the domain, Complexity Formula 1b can be used to generate the size of the new graph:

$N_1 = 8, T_1 = 16.$

$N_2 = 2 * 8 = 16.$
$T_2 = 2 (8 + 16) = 48.$

Thus, the graph goes from a (8:16) graph to a (16:48) graph while the POP table is only extended by one entry. Now suppose another 5 independent transitions are to be added to the new graph, $G_2(16 : 48)$. The size of the new graph can be calculated using the general form of Complexity Formula 1b.

$N_6 = N_{2+4} = 24N_2 = 16 * 16 = 256.$
$T_6 = N_{2+4}$
$= 2^4T_2 + 4.2^4N_2$
$= (16 * 48) + (64 * 16) = 1792.$

Again the POP table can be increased by only 5 more entries as the transitions are independent and do not effect the existing POP table entries. Thus, while the complexity of the transition networks are increasing exponentially, the POP tables are only increasing in a linear fashion.

D.5 Summary

Figure D.7 shows how the representations change when independent transitions are added, with best case transition networks being one-way graphs and worst case transition networks being two-way graphs.

Representation	**Initial size**	**# of added independent transitions**			
		1	3	5	7
POP table	9	10	12	14	16
One-way net (Best case)	8:8	16:24	64:160	256:896	1024:4608
Two-way net (Worst case)	8:16	16:48	64:320	256:1792	1024:9216

Figure D.7 : Comparison of representations.

This shows that the POP representation is a more economical technique for domain representation than transition networks and, as has been shown elsewhere, it is a representation that more closely reflects the semantics of the operations that are being carried out (Kemp and Smith, 1994a).