x-21-159113-x
F6988

Developing the Didactic Operations for Intelligent Tutoring Systems: A Synthesis of Artificial Intelligence and Hypertext

Marios Angelides, B.Sc. (Hons)

London School of Economics and Political Science

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

September 1992

ABSTRACT

This thesis is concerned with Intelligent Tutoring Systems. It investigates the architecture of an Intelligent Knowledge Based Tutoring System in terms of three knowledge models: that of the domain, the student and the tutor, and examines the interrelatedness and interconnectedness of these three knowledge models.

Existing Knowledge Based Tutoring Systems are reviewed, and the relationship between their behaviour and architecture is analysed by evaluating them against Wenger's model of a didactic operation. Two such systems, PROUST, a tutoring system for Pascal program debugging skills, and micro-SEARCH, a tutoring system for mathematical transformations skills, are used in the study. This evaluation serves two purposes: to unravel the requirements for interrelatedness and interconnectedness between the three knowledge models in order to develop a true Knowledge Based Tutoring System with a full-scale didactic operation, and to uncover the limitations of the current generation of Knowledge Based Tutoring Systems and how they fail to fully encompass these requirements.

On this basis the thesis goes on to propose a hybrid model made up of Artificial Intelligence and Hypertext concepts which helps to overcome the limitations of existing Knowledge Based Tutoring Systems. This model in particular addresses the requirements for the development of an Intelligent Tutoring Systems with a full-scale didactic operation. The model integrates Hypertext's explicit information nodes and linking properties with Artificial Intelligence's logical inferencing on knowledge representation schemes. The thesis finally shows how to use this model to design a generic Intelligent Tutoring System that supports a full-scale didactic operation.

To my parents,

Costas and Christalla

ACKNOWLEDGEMENTS

This thesis would not have been possible without the encouragement, friendship, support, and patience of Dr. Georgios I. Doukidis, Dr. Edgar A. Whitley, and Dr. Ray J. Paul. I am very grateful to Georgios who accepted me in 1988, when he was still a lecturer in the Information Systems Department at the London School of Economics, to pursue this Ph.D. degree under his supervision. Georgios has given me so many opportunities (and he still does) in the years that I have known him that I cannot hope to ever repay him for what he has done for me. Special thanks are also due to Georgios' wife Lina who had to put up with me (both in Athens and while they were living in London) every time I had to consult with George about the Ph.D.

I am also indebted to Edgar who agreed to read for Georgios the drafts of my thesis when these started coming through. Without his support and encouragement this thesis would still be no more than hazy ideas rather than a properly presented argument. I have benefitted greatly from the endless debates about the thesis, his views on my work, and his friendship.

I am also deeply indebted to Ray whose experience with Ph.D. supervision I called up on at the finishing stages of my thesis. With his continuous support, patience, and encouragement during the three difficult months leading to the viva, the draft matured to a Ph.D. thesis. I have also benefitted greatly from his views on my thesis and his true friendship. Ray agreed to be my 'supervisor' for the purposes of the graduate office records when Georgios was required to go to Greece in 1989. During this time, I called up on him endless times for signatures, references, advice, supervision. Ray has helped so much in all my years at the LSE (since my early undergraduate years) that I cannot hope to ever repay him for what he has done for me.

To my parents, who have given me so much love, I am dedicating this thesis. They, my godparents and my grandmother Maria, have taught me the most important lessons about life, love, and friendship. My parents have made many sacrifices to enable me to do the work I enjoy so much. Special thanks are due to my sister Avgoula who sacrificed several days of her holidays in London to help me submit the thesis for the viva. Thank you.

The Information Systems department at the London School of Economics and Political Science has been a wonderful place to work in - both as a student and, more recently, as a member of staff. The friendship, encouragement, and discussions I have enjoyed with the lecturers, secretaries and students have shaped many of the ideas expressed in this thesis to an extent that I could never mention all the individuals concerned. However, I would not be doing justice if I did not thank Miss Alison O'Neil, my efficient secretary, who typed parts of the thesis, and Miss Geraldine Gibson, a student on the M.Sc. course on Analysis, Design and Management of Information Systems who practically became my assistant during the three days leading to the viva.

Last, but by no means least, I am grateful for the love and companionship of all my friends who kept me happy and 'sane' while I was researching my thesis. Their patience was particularly appreciated when I became 'anti-social' in the final stages of preparing the thesis. Special thanks go to my friend Kurt Klappholz who was the first university academic I met when I arrived at the LSE and who became a dear friend. I much enjoyed the endless discussions I had with Kurt in Rosebery Hall. I am very grateful to him for all his support and advice and the many opportunities he has given me in Rosebery Hall, when they were most needed. Special thanks also go to my very good friend Harry Ioannides who gave me (free) accommodation in his house for the first three months as a Ph.D. student and whenever, I wanted a break from work. I enjoyed very much living with him and his family. Special thanks are also due to Michael Lambrou, Lakis Theodotou, Lakis Tsiakkas and his family, and Panayiotis Panayis for all the help they gave during the time that I have known them.

The first two years of this research were supported by an ORS Award no. 8855029. In addition, the first year of the research was supported by an LSE Library Bursary and a bursary from the LSE 1980's fund and the second year was supported by a Wingate Scholarship and the LSE Jackson Lewis Scholarship.

Marios Angelides

September 1992.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1: INTELLIGENT TUTORING SYSTEMS

After many years of research and development the educational approach is still very much the same as it was centuries ago. As mentioned by Anderson and Reiser [1985], on the one hand, we have conventional classroom instruction and learning that involves listening to lectures, reading texts, and working alone on assigned homework problems. On the other hand, we have individual tutoring that provides the student with an experienced person, the tutor, whose major role is to guide the student's reading of texts and problem solving, and to turn problem solving episodes into more effective learning experiences. Studies of students learning experience have revealed that individual tutoring appears to be more effective. The studies such as that by Anderson and Reiser [1985] revealed that with private tutors it only takes approximately 25% of the time spent to learn as much as classroom students learned. In such studies, the tutored students are seen to perform better on tests than the average classroom student. The major benefit occurred with the poorer tutored students rather than with the best students for whom there was relatively little advantage.

Teaching of students has recently entered a new era. For a number of years, we have been studying how students learn logic, mathematics, programming, and similar skills. We have reached a stage where we can develop computer-based tutors for these kind of domains. From its early beginnings, the computer has been considered as capable of instructing, thereby improving the quality of education. Computer systems for student tutoring that could offer to the student the same instructional and pedagogical potential as a human tutor, are being developed [Swartz and Yazdani, 1992].

The major objective of these systems has been to explore and understand the student, the student's special needs and interests, and respond to these needs as a human tutor would. These systems are considered as very important, especially in university teaching [Duchastel and Imbeau, 1988], in the promise which they offer as learning resources which the instructors can place at the disposal of their students to foster their learning. It is also believed that they will contribute to shaping the future direction of educational technology and influence positively the way we conceive of learning and how it can take place. However, Dede [1986] stresses that it is still the case that without the continual presence of human tutors, the new educational and pedagogical technologies will weaken the quality and efficiency that computer-based instructional programs can attain.

Computer-Assisted tutorials or Computer-Assisted Instruction as they are better known, have been developed in many areas such as Mathematics, Geography, Nuclear Physics, Grammar, Electronics, Medicine, Meteorology, Statistics and Programming Languages, especially Lisp and Pascal.

Work in Computer Assisted Instruction has been taken over and gradually dominated by Artificial Intelligence to become another Artificial Intelligence application, that of Intelligent Tutoring Systems [Swartz and Yazdani, 1992], [Clancey and Soloway, 1990], [Clancey, 1987], [Harmon, 1987], [Anderson and Reiser, 1985], [Sleeman and Brown, 1982].

In this Chapter, the thesis gives an account of the state-of-the-art in the Intelligent Tutoring System architecture and in particular the components that make up this architecture. The Chapter first presents the most popular Intelligent Tutoring System

Architectures. It then gives a detailed analysis of the individual components of an Intelligent Tutoring System according to these architectures. The thesis then proposes to examine the nature of interaction of these components, during the period of use of an Intelligent Tutoring System. As is shown in this Chapter, most of the recent Intelligent Tutoring Systems have been developed as Knowledge Based Systems, as the result of the field being an Artificial Intelligence Application area. Consequently, the thesis will focus on Intelligent Knowledge Based Tutoring Systems, although in Chapter 1 the discussion includes non-Artificial Intelligence Systems. The reason for this is simply to provide a better understanding of the field.

1.1 POPULAR INTELLIGENT TUTORING SYSTEMS ARCHITECTURES

Brecht and Jones [1988] suggest that, although there has recently been some agreement regarding the components of an Intelligent Tutoring System, nevertheless, there is still little consensus regarding the exact nature and interaction of these components. What would really be the ideal Intelligent Tutoring System?

A computer system is classified as an Intelligent Tutoring System if it passes three tests of intelligence [Burns and Capps, 1988], [Clancey and Soloway, 1990], [Burns and Parlett, 1991]. First, the subject matter, or domain, must be "known" to the computer system well enough for the embedded domain expert to draw inferences or solve problems in the domain of application. Second, the system must be able to deduce a learner's approximation of that knowledge. Third, the tutorial strategy or pedagogy must be intelligent in that the "instructor in the box" can implement strategies to reduce the difference between the expert and the student performance. These ideas are found in the three most common architectures for tutoring systems: Anderson's architecture, the

Hartley and Sleeman architecture and O'Shea's five ring model.

According to Anderson's architecture [Anderson et al, 1985] [Anderson, 1989] there are four sources of knowledge in an Intelligent Tutoring System. First is the Domain Expert module which is capable of actually solving problems in the domain. This is sometimes also referred to as the 'ideal student' model. Second is the Bugs catalogue, an extensive library of common misconceptions and errors in the domain. These are the possible deviations a student can make from the ideal expert behaviour. Third, is the Tutoring knowledge module, which contains the strategy to teach the domain knowledge. It is based on three sets of principles: one for determining from student's behaviour what they know and what misconceptions or bugs they have, another for deciding when to interrupt them in the problem solving process and what advice to offer at this point, and last, to figure out what exercises a student should do and when the student should advance to new material. In Anderson's fourth source of knowledge, internal models of the students are built by the tutor where the students' knowledge, difficulties, and misconceptions are represented. This will enable the tutor to tailor its instructions to each individual student. Finally, there is a user interface module to administer the interaction between the tutor and the students.

The Intelligent Tutoring System architecture proposed by Hartley and Sleeman [1973] has four similar primary components represented as four distinct knowledge bases. These contain, firstly, a representation of the teaching task, that is knowledge of the task domain, which will not only include specific educational objectives, but also the task analyses which indicate the structure of material and the components of operations. Secondly, a model of the student through their performance data or other achievement measures, in other words, a model/history of their behaviour. Thirdly, a set of teaching operations, that will include curriculum management, teaching style and type of feedback, and use of remedial material, in other words teaching strategies. Finally, a set of means-ends guidance rules, which are decision rules that state the conditions under which the teaching operations should be used with particular students during their learning.

O'Shea's five ring model, [O'Shea et al, 1984], has five primary components, one for every ring of the model: the student history, which is a record of material presented to the learner and the learner's responses; the student model, which makes predictions about the learner's future performance and current state of knowledge and ability; the teaching strategy, which relates the systems view of the learner to the general types of teaching action that are possible, and decides the type of the next action; the teaching generator, which is a mechanism that yields a specific teaching action for use by the teaching administrator; and finally, the teaching administrator, which presents material to the learner and processes learner responses.

Yazdani [1986a] classifies the three different schools of thought in terms of two extremes (see Figure 1.1). One is the typical learning environment, where the Anderson proposal is closer, and the other is the traditional Computer Assisted Instruction with distinctive emphasis on teaching, where the O'Shea proposal is closer. He argues that the choice of strategy is dependent on the nature of expertise to be taught. He points that exceptionally abstract and general concepts, e.g. Model building can be better taught within a learning environment by constructing a computer-based microworld. Also the teaching of skills which are basically problem solving in a specific domain can be best achieved via problem-solving monitors such as Anderson's. As the tasks become less abstract and more concrete the traditional Computer Assisted Instruction approach becomes more suitable.

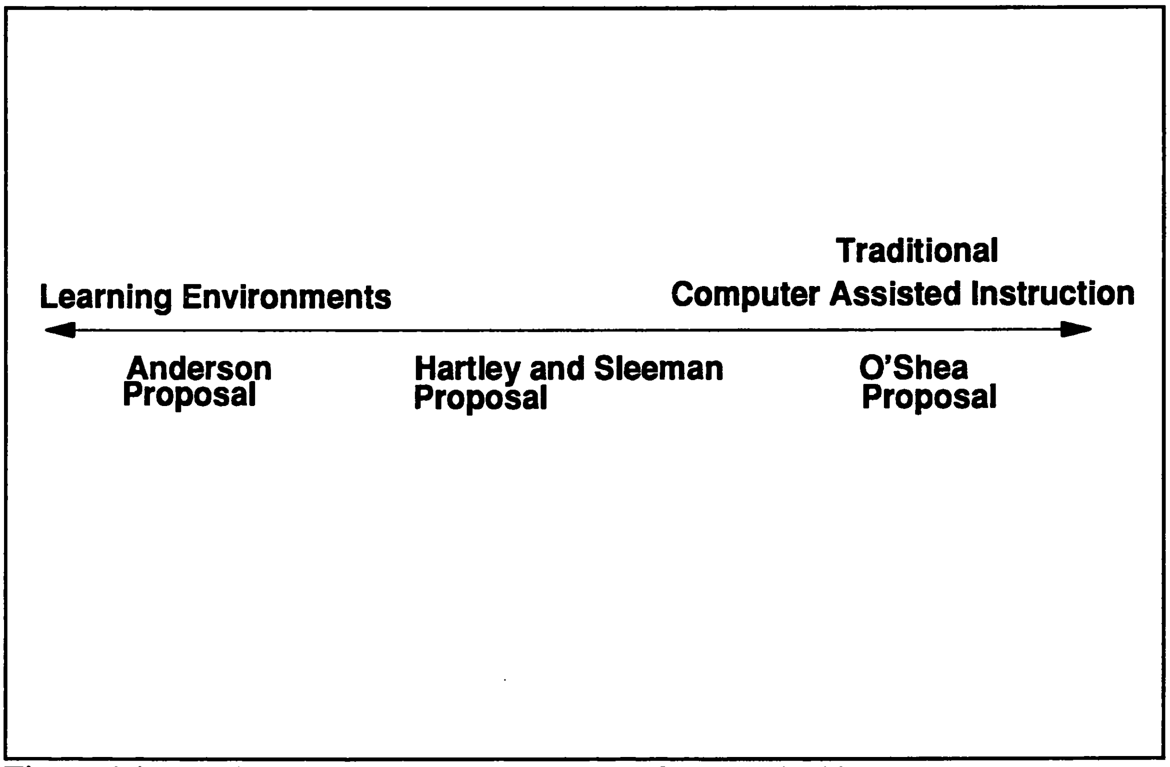


Figure 1.1: Classification of Intelligent Tutoring Systems Architectures

These three architectures suggest that at the foundation of Intelligent Tutoring Systems one finds three special kinds of knowledge and problem-solving expertise programmed in a sophisticated instructional environment: Domain Expert Knowledge, Student Diagnostic Knowledge, and the Instructional or Curricular Knowledge.

The Instructional Environment refers to the 'Environment' part of the system which is responsible for specifying or supporting the activities that the student does and the methods available to the student to do those activities [Burton, 1988]. It defines the kind of problems the student is to solve and the tools available for solving them. In the SOPHIE I troubleshooting environment [Brown, Burton and deKleer, 1982], the activity is finding a fault in a broken piece of equipment, and the primary tool available to solve the problem is the ability to ask in English for the values of measurements made on the equipment. The environment part of SOPHIE I supports these activities by providing a circuit simulation, a program to understand a subset of natural language, and the routines to set up contexts, keep history lists, etc. The Environment includes some aspects of help that the system provides to the student while he is solving problems, but does not include those forms of help that one would classify as requiring intelligence; these are dealt with by the tutor in the box. The Instructional Environment in many ways defines the way the student looks at the subject matter [Burton, 1988].

A major consideration in developing Intelligent Tutoring Systems is how the three kinds of knowledge are embedded in such systems. Other considerations include how these systems accrue the advantages of advanced computer interface technologies, how Intelligent Tutoring Systems will emerge in the real world of complex problem solving, how both researchers and developers must learn to evaluate the efficiency and effectiveness and overall quality of these systems, when, where, and how is an Intelligent Tutoring System quality understood, and what is the exact nature of the interaction between the "real" components and the system in its actual environment of operation?

For the sole purpose of convenient classifications of Intelligent Tutoring Systems research and development dimensions, it is assumed that the anatomy of an Intelligent Tutoring System is as is shown in Figure 1.2. The components highlighted are discussed in the following sections. Appendix A classifies existing Intelligent Tutoring Systems under five generic categories. As explained before, non-Artificial Intelligence systems are also classified under these categories.

1.2 THE DOMAIN EXPERT MODEL

The first key place for intelligence in an Intelligent Tutoring System is in the knowledge

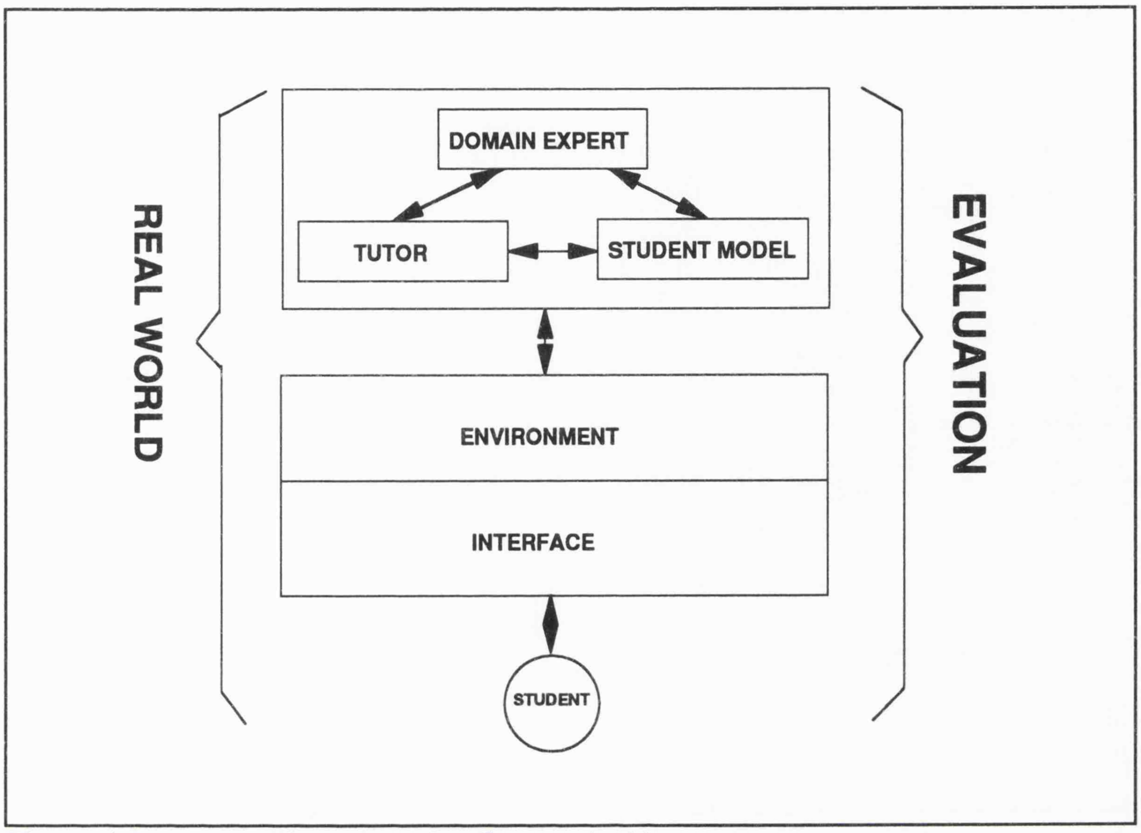


Figure 1.2: The Intelligent Tutoring System Architecture

that system has of its subject domain [Woolf, 1987], [Woolf, 1988]. Anderson [1988] claims that this is the backbone of an Intelligent Tutoring System. A domain expert model must have an abundance of domain knowledge. A great deal of the effort in developing a complete and efficient domain expert model would be expended on knowledge elicitation and analysis and knowledge codification for the domain expert model. The sheer amount of knowledge required in most complex domains ensures that the development of the domain expert will be labour-intensive. As Intelligent Tutoring Systems techniques evolve, tutoring system shells may be expected to assume much of the work involved in tutoring systems development. However, such authoring systems cannot do any part of the work of knowledge elicitation and analysis and can only provide an easier means to codify the knowledge into the domain expert.

There have been three approaches in encoding knowledge into the domain expert which give rise to the three different types of models for domains experts. The first approach, which gives rise to the ***Black Box Model*** of a domain expert, involves finding a method of reasoning about the domain that does not actually require codification of the knowledge that underlies human intelligence. In other words, the reasoning is implemented using conventional data processing rather than symbolic processing methods. The second approach, which gives rise to the ***Glass Box Model*** of a domain expert, involves reasoning about the domain by applying codified knowledge. This is the standard Expert Systems approach to reasoning with knowledge [Kopec et al, 1992]. The third approach, which gives rise to the ***Cognitive Model*** of a domain expert, involves making the domain expert a simulation, at some level of abstraction, of the way humans use the knowledge. As argued by Anderson [1988], the third approach, although the most demanding of the three, produces the best high-performance tutoring systems. Anderson [1988] argues that the pedagogical effectiveness of the three domain expert models increases along with the implementation effort in the order in which they have been presented.

First generation expert systems were developed using the knowledge engineering methodology which involved deploying humanlike knowledge that was codified using one or more knowledge representation schemes, mainly production rules, stored in a separate knowledge base [Doukidis, Angelides and Harlow, 1988], [Doukidis, Rogers and Angelides, 1989]. Expert performance was achieved through reasoning with the contents of such a knowledge base. Such products were named knowledge based expert systems an equate to the Glass Box models of a domain expert. First generation expert systems were also developed following the criterion-based approach: any system that achieves high quality performance could be classified as an expert system. Therefore, any kind of domain expert in an Intelligent Tutoring System capable of undertaking a complex task proficiently would be regarded as an expert system by the criterion-based approach. Such were the Black Box models of a domain expert although they were originally of limited scope.

Second generation expert systems which promise a more fundamental understanding of their domain of discourse and are not so narrow or brittle as their predecessors, are currently under test and development. They have not yet achieved the same levels of performance as the first generation of expert systems but they are regarded as the hope for the future [Anderson, 1988]. Expert systems of this kind are developed as ***Qualitative Process models*** which are a special kind of Cognitive model [Clancey, 1988]. They are concerned with reasoning about the causal structure of the world and in particular the domain of discourse. Both generations of expert systems are currently in use in the development of domain expert models [Fink, 1991].

1.2.1 Black Box Models

A Black Box model of a domain expert is one that generates the correct input-output behaviour over a range of tasks in the domain, and so can be used as a judge of correctness. However, the internal computations by which it provides this behaviour are either not available or are of no use in delivering instruction.

A classical example of the use of a Black Box model for a domain expert is WEST [Burton and Brown, 1982], a tutor for a mathematical game. In this system the domain expert performs an exhaustive search of the possible moves and determines the optimal move given a particular game strategy. Such a domain expert model can be used in a reactive tutor that tells the students whether they are right or wrong, and possibly what the right move would be. This suggests a cheap and easy way of converting off-the-shelf expert systems into tutoring systems that could be quite pedagogically effective.

The intelligent tutoring paradigm, however, is based on the belief that what a tutor says is critical and that is helpful to say more than just "right", "wrong" and "do this". The problem with Black Box models of domain experts is how to build a more articulate tutor around such an expert system when the knowledge of that system is not accessible. Burton and Brown [1982] suggested a methodology, called ***issue-based tutoring*** which involves recognising patterns, in both the student's and the expert's surface behaviour, so called issues, and generating instruction for those patterns. This is the approach which Burton and Brown [1982] followed in WEST. Nevertheless, this surface-level issue-based tutoring does not solve the problem of providing explanations about the actual reasoning process, if the Intelligent Tutoring System does not have access to the internal structure of the domain expert.

1.2.2 Glass Box Models

A Glass Box model of a domain expert is similar to that of a conventional Expert System. As King and McAulay [1992] argue, the basic methodology of building these expert systems involves a knowledge engineer and a domain expert who can identify a problem area and its scope, who can enumerate and formalise the key concepts in the domain, formulate a system to implement the knowledge, and then iteratively test and refine that system. These systems are characterised by the great quantity and humanlike nature of knowledge that is articulated. Because of its nature, the emerging expert system should be more amenable to tutoring than a Black Box model because a major component of this expert system is an articulate, humanlike internal representation of the knowledge underlying expertise in the domain.

The expert system methodology in its variations has been successfully used to tackle a wide range of tasks: interpretation, prediction, diagnosis, design planning, monitoring, debugging, repair, control, and in certain cases [Stevens, Collins and Goldin, 1982] tutoring expertise.

GUIDON [Clancey, 1982 and 1987] uses MYCIN [Shortliffe, 1976] as a Glass Box model in a tutoring system. MYCIN consisted of 450 if-then rules which encoded a model of the probabilistic reasoning that underlies medical diagnosis. With GUIDON, tutorial interaction is driven by t-rules (i.e. tutorial rules), an extension of issue-based tutoring. T-rules are compiled to be a combination of the difference between the expert behaviour and the student behaviour and an expert's reasoning process. In Figure 1.3 below, the t-rule refers to the internal structure of the domain expert, such as rules and goals, and not on the surface behaviour.

The GUIDON project has highlighted one further important issue: for tutoring systems to be effective, it is not enough to understand the knowledge in the domain expert but also the way in which this knowledge is deployed and the humans restrictions levied on it.

1.2.3 Cognitive Models

A Cognitive model of a domain expert is a simulation of human problem solving in a domain in which the knowledge is decomposed into meaningful, humanlike components and deployed in a humanlike manner [Anderson et al, 1990]. Cognitive Models are best understood in the context of the three types of knowledge that a tutoring system may be

IF
The infection which requires therapy is meningitis
Organisms were not seen in the stain of the cuiure
The type of infection is bacterial
The patient does not have a head injury defect
The age of the patient is between 15 and 55 years
THEN
The organisms that might have been causing the infection are dipiococus-pneumoniae (.75) and neisseria-meningitidis (.74)

A typical MYCIN rule

IF
The number of factors appearing in the domain which need to be asked by the student is zero
The number of subgoais remaining to be determined before the domain rule can be appiied is equal to 1
THEN
Say: subgoai suggestion
Discuss the (sub)goal with the student in a goal-directed mode
Wrap up the discussion of the domain being considered

A Guidon tutorial rule

Figure 1.3: MYCIN and GUIDON Rules [Clancey, 1987]

developed to tutor: ***procedural*** that entails knowledge about how to perform a task such as calculus problem solving, ***declarative*** that entails a set of facts appropriately organised so that an Intelligent Tutoring System can be implemented to reason with them, and causal knowledge in the form of qualitative models that entails knowledge about a device that allows one to reason about the behaviour of that device, for example, electronic circuit troubleshooting.

Procedural Knowledge

Procedural knowledge represented as a Cognitive Model can take the form of a rule-based production system. This provides some model of human problem solving behaviour matched to a working memory of facts which is regarded as some form of human short-term memory. This is the approach followed in the Lisp Tutor [Anderson, Boyle and Reiser, 1985], the Geometry Tutor [Anderson, Boyle and Yost, 1985], AlgebraLand

[Brown, 1983], BUGGY [Brown and VanLehn, 1980], and many other systems.

One of the major advantages of production rules for the purposes of instruction is their modularity: each production rule is an independent piece of knowledge. This means that a rule can be communicated to the student independently of communicating the total problem structure in which it appears. However, this does not mean that they are context-free. They specify explicitly that part of the context which is relevant. The current generation of goal-factored production systems make explicit reference in their conditions to goals that each and every one of their production rules are relevant to. This allows communication to the student of only the relevant information. Another major advantage of the modularity of production rules is that we can use the rules to represent the student's knowledge state. Given in Figure 1.4 below is a production rule representation of the subtraction skill as procedural knowledge from the BUGGY system.

```
Sub () Satisfaction Condition: TRUE
   L1: {} --->                 (ColSequence RightmostTOPCeRightmostBottomCell RightmostAnswerCell)
COLSEQUENCE (TC BC AC) Satisfaction Condition: (Blank? (Next TC))
   L2: {} --->                 (SubCol TC BC AC)
   L3: {} --->                 (ColSequence (Next TC) (Next BC) Next AC))
SubCol (TC BC AC) Satisfaction Condition: (NOT (Blank? AC))
   L4: {(Blank? BC)} --->      (WriteAns TC AC)
   L5: {(Less? TC BC)} --->    (Borrow TC)
   L6: {} --->                 (Diff TC BC AC)
Borrow (TC) Satisfaction Condition: FALSE
   L7: {} --->                 (BorrowFrom (Next TC))
   L8: {} --->                 (Add10 TC)
BorrowFrom (TC) Satisfaction Condition: TRUE
   L9: {(Zero? TC)} --->       (borrowFromZero TC)
   L10: {} --->                (Decr TC)
BorrowFromZero (TC) Satisfaction Condition: FALSE
   L11: {} --->                (Write9 TC)
   L12: {} --->                (BorrowFrom (Next TC))
```

Figure 1.4: Production rule on subtraction skills in BUGGY [Brown and VanLehn, 1980]

With this rule-based approach it is possible to implement a tutoring methodology called ***model tracing*** in which a student's surface behaviour in solving a problem is contrasted with a sequence of production rules that are firing in the domain model. This correspondence can then be used to produce an interpretation of the student's surface behaviour. The major function of such a student model trace is to provide feedback or student errors as soon as possible. With this rule-based approach is possible to ***compile the expert out*** and perform in advance all possible computations of the expert for a particular problem and to store them in some efficiently indexed scheme.

Declarative Knowledge

In many situations, there is a need for students to understand the principles and facts of the domain and how to reason with them generally and not just become competent at any one application of such knowledge. This calls for the use of declarative knowledge representations. However, this does not suggest that procedural tutoring and declarative tutoring are incompatible. On the contrary, the Artificial Intelligence community has labelled them *dual semantics*, because they mutually complement each other [Doukidis and Whitley, 1988]. As a result, the goal of instruction of an Intelligent Tutoring System may be to make a student competent with the procedures of a domain and articulate about the justifications of those procedures and other factual information. With declarative knowledge one wants hierarchical representations of knowledge, structured so that flexible inference procedures on the knowledge base can be defined.

With declarative representations, the knowledge base is separate from the inference mechanisms, unlike most procedural representations. With some declarative mechanisms, such as frames and schema systems, *procedural attachments* are also embedded in the declarative knowledge structure. Procedural attachments are rules that actively wait for their conditions to become true and fire in dynamic systems.

Figure 1.5 shows a schema representation for evaporation as declarative knowledge from WHY [Stevens and Collins, 1977]. There are slots for the actors of evaporation, for the factors that influence the amount of evaporation, for the functional relationships among these factors, and the result of evaporation. Bugs are created by erroneous entries in the slots.

```
Evaporation
    Actors
        Source: Large-body-of-water
        Destination: Air-mass

    Factors
        Temperature (Source)
        Temperature (Destination)
        Proximity (Source, Destination)
    Functional-relationship
        Positive (Temperature (Source))
        Positive (Temperature (Destination))
        Positive (Proximity (Source, Destination))
    Result
        Increase (Humidity (Destination))
```

Figure 1.5: A schema representation for evaporation in WHY [Stevens and Collins, 1977]

In tutoring with declarative knowledge bases, it is assumed that the student has already in hand the inference procedures that can be used for reasoning with the knowledge. Therefore, the task becomes one of representing the knowledge in a such form that these inference procedures can be invoked. This suggests the making of a very simple tutorial agenda, namely, to determine what the student has filled in for each node, fill in any missing conceptions and debug any misconceptions. With declarative knowledge, as opposed to procedural knowledge, the Intelligent Tutoring System must be able to understand how students draw inferences on their declarative knowledge base. Figure 1.6 illustrates a rule from the set of tutoring rules formulated by Collins for use in WHY.

```
IF
   The student gives an explanation of one or more factors that are not sufficient
THEN
   Formulate a general rule for asserting that the factors given are sufficient
   Ask the student if the rule is true
```

Figure 1.6: An example of a tutoring rule for WHY [Stevens and Collins, 1977]

These rules have a resemblance to the issue-based recognition rules used for black and glass box models. However, here the conditions for such rules refer to the underlying knowledge rather than to any surface behaviour and incorporate a mixture of knowledge assessment and instruction.

Qualitative Process Models

A qualitative model of a domain expert is concerned with the knowledge that underlies our ability to mentally simulate and reason about dynamic processes [White and Frederiksen, 1990]. This is important when engaging in troubleshooting behaviour, which involves reasoning through the causal structure of a device to find potential faults. de Kleer and Brown's work on envisionment with SOPHIE is an example in which the causal structure of an electronic circuit is communicated to the user. They divide the process of envisionment into constructing a causal model and then simulating the process in this model [White and Frederiksen, 1990]. Figure 1.7 shows their conception of the process.

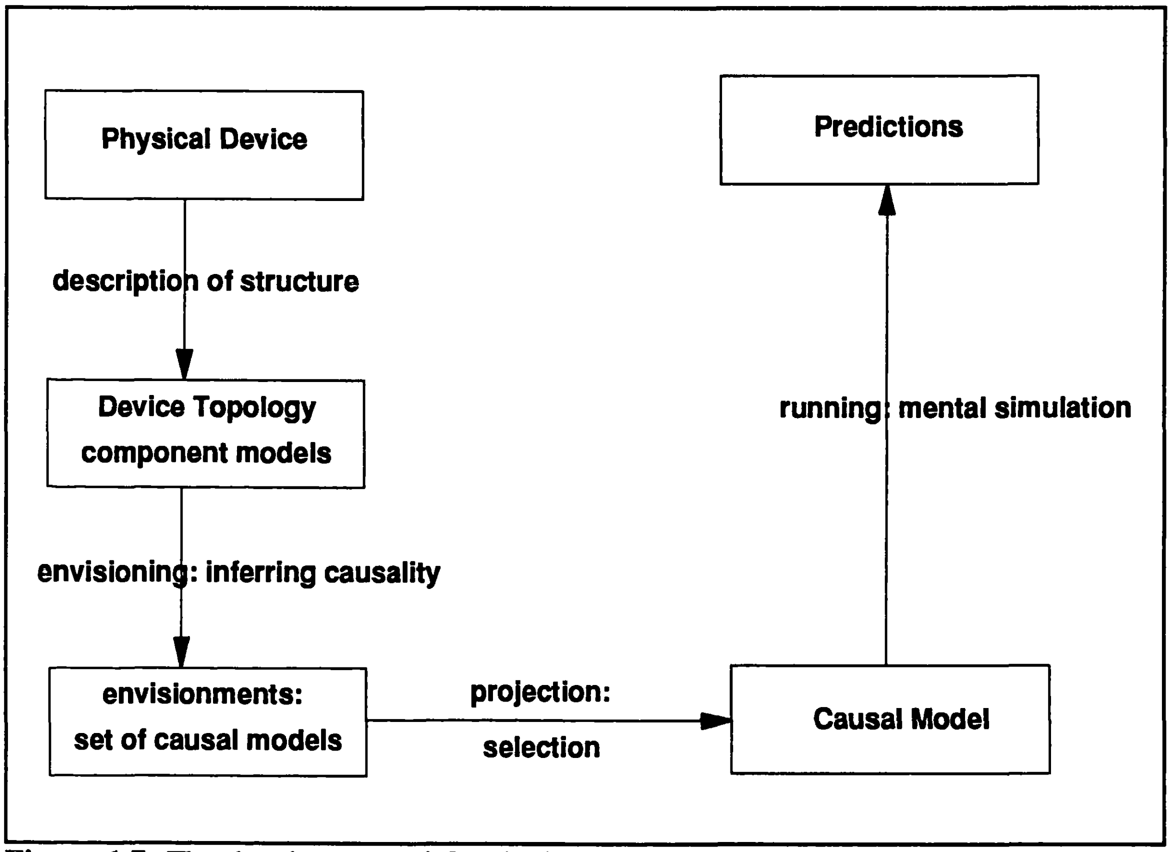

Physical Device
Predictions
description of structure
Device Topology component models
running: mental simulation
envisioning: inferring causality
envisionments: set of causal models
projection: selection
Causal Model

Figure 1.7: The development of Qualitative Simulation

The causal structure of the device is inferred from its topology by examination of the local interactions among components and not of their function in the device. As a result, it is called the *no function in structure* principle. Having this causal model, de Kleer and Brown then use a calculus to propagate the behaviour of the device through these components.

1.3 THE STUDENT MODEL

The second key place for intelligence in an Intelligent Tutoring System is in the knowledge that the system infers of its user-learner- the student. An Intelligent Tutoring System infers a model of a student-user's current understanding of the subject matter and uses this individualised model to adapt the instruction to the student's needs [Corbett, Anderson and Patterson, 1990]. The component of an Intelligent Tutoring System that represents the student's current state of knowledge is called the Student Model. Inferring a student model is called Student Diagnosis. An Intelligent Tutoring System's diagnostic system attempts to uncover a hidden cognitive state (the student's knowledge of the subject matter) from observable behaviour. The student model and the diagnostic module are tightly linked together. Designing these two components is known as the Student Modelling Problem.

The input for diagnosis is garnered through the interaction with the student. The particular kinds of information available to the diagnosis module depend on the overall Intelligent Tutoring System application. This information could be answers to questions posed by the Intelligent Tutoring System, moves taken in a game, or commands issued to an editor. This information is sometimes complemented by the student's educational history. The output of the diagnostic module depends on the use of the student model. Nevertheless, it should reflect the student's current knowledge state. Some of the most common uses for the student model include, advancement of the user to the next curriculum topic, offering unsolicited advice when the student needs it, dynamic problem generation, and adapting explanations by using concepts that the student understands. All these assume consultation with the student model before any kind of action is taken.

Because there are so many ways to use the student model, one cannot talk sensibly about the output of the diagnosis module, nor can one classify student modelling problems by the desired input-output relationship. What makes sense is to classify these problems partially according to the structural properties of the student model and partially on properties of the input available to the diagnosis module [VanLehn, 1988]. This classification has three dimensions. The first relates to the input, and the other two are structural properties of the student model.

1.3.1 Bandwidth

The input to the diagnosis unit consists of various kinds of information about what the student is doing or saying. From this, the diagnosis unit must infer what the student is thinking and believing. The Bandwidth dimension is a rough categorisation of the amount and quality of the input information. In attempting a task posed by an Intelligent Tutoring System, students go through a series of *mental states* [Payne, 1988]. The highest bandwidth an Intelligent Tutoring System could attain would be a list of the mental states that the students traverse as they solve problems. Since an Intelligent Tutoring System can only approximate a student's mental state via some indirect information, the highest bandwidth category is *Approximate Mental States.*

In more complicated forms of problem solving the students may make observable changes that carry the problem from its initial unsolved state to its final solved state, for instance, while playing chess. This results in a series of observable intermediate states, for example, midgame board positions. Sometimes an Intelligent Tutoring System has access to these intermediate states, and sometimes it can see only the final state. This suggests two other categories of bandwidth: *intermediate mental states* and *final mental states*. Each category is intended to include the information in the category beneath it.

1.3.2 Target knowledge type

Student models might be asked to solve the same problems that students do and therefore be used to predict the student's answers. Solving problems requires some kind of interpretation process that applies knowledge present in the student model to solve the problem. There are two types of interpretation, one for declarative knowledge and one for procedural. The interpreter for procedural knowledge does not perform any form of knowledge base search but makes decisions based on local knowledge about which strand of knowledge to turn onto and follow. A declarative interpreter searches through all the strands of knowledge, assembles facts and deduces answers from them.

These considerations underlie the second dimension in the space of student modelling problems, the ***type*** of knowledge in the student model. There are, therefore, two types of knowledge: *procedural* and *declarative*. Procedural knowledge is further subdivided into two subcategories: *flat* and *hierarchical*. Hierarchical representations allow subgoaling, flat ones do not. Therefore, procedural knowledge may be represented as a hierarchical tree where as flat cannot.

1.3.3 Differences between Student and Expert

The Domain Expert Model is used for providing explanations of the correct way to solve a problem. The student model must keep track of the degree to which the student-user has equivalent knowledge. Because students will move gradually from their initial state of knowledge towards mastery, student models must be able to change from representing novices to representing experts. Most Intelligent Tutoring Systems use the same knowledge representation language for both the expert model and the student model. Conceptually, an Intelligent Tutoring System has one knowledge base to represent the expert and one to represent the student. Nevertheless, the student model is generally represented as the expert model plus a collection of differences [Wilkins, Clancey and Buchanan, 1988]. There are basically two kinds of differences: *missing conceptions* and *misconceptions*. A missing conception is an item of knowledge that the expert has and the student does not. A misconception is an item of erroneous knowledge that the student has.

Conceptually, the student model is a proper subset of the expert model along with a list of student misconceptions. Such student models are called *Overlay models*. With overlay models, a student model consists of the expert model plus a list of items that are missing. Weights on each element in the expert knowledge base are also introduced [VanLehn, 1988].

To model misconceptions an Intelligent Tutoring System employs a library of predefined misconceptions and missing conceptions known as the *Bugs Library*. In this case, the student model consists of an overlay model plus a list of bugs. Such a system performs student diagnosis by finding bugs from the library that, when added to the overlay model, yield a student model that fits the student performance. An alternative to the bug library approach is to construct bugs from a *library of bug parts*. Bugs are constructed during diagnosis rather than being predefined [VanLehn, 1988].

1.3.4 Diagnostic Techniques

Figure 1.8 shows how the nine diagnostic techniques that have been used in existing Intelligent Tutoring Systems align with the student model knowledge and, in specific, with the three kinds of information described above, namely, bandwidth, target knowledge type and difference between student and expert.

Knowledge Type Bandwidth	Procedural-Flat	Procedural-Hierarchical	Declarative
Mental States		Model Tracing	
Intermediate States	Issue Tracing	Plan Recognition	Expert System
Final States	Path Finding Condition Induction	Decision Tree Generate and Test Interactive Diagnosis	Generate and Test

Figure 1.8: Student Diagnostic techniques

Model Tracing

The model tracing technique assumes that all of the student's mental states are available to the diagnostic program. The basic idea is to use an undetermined interpreter for modelling problem solving [Corbett, Anderson and Patterson, 1990]. At each step in problem solving, the undetermined interpreter may suggest a whole set of rules to be applied next, whereas a deterministic interpreter can only suggest a single rule. The diagnostic program fires all these suggested rules, obtaining a set of possible next states. One of these should correspond to the state generated by the student. If so then is reasonably certain that the student used the corresponding rule to generate the next mental state and so must know that rule. The student model is updated accordingly. The term *Model Tracing* comes from the fact that the diagnostic program merely traces the execution of the model and compares it to the student's activity.

Path Finding

If the bandwidth is not high enough to warrant the assumption that the student has applied just one mental rule, then model tracing is inapplicable. However, it is feasible to put a path-finding algorithm in front of the model-tracing algorithm. Given two consecutive states, like the current state and the goal state, it finds a path, or a chain of rule applications, that takes the first state into the second state. The path is then given to a model tracing algorithm, which treats it as a faithful rendition of the student's mental state sequence.

Condition Induction

In contrast, model tracing assumes that two consecutive states in the student's problem solving can be connected by a rule in its model. This puts strong demands on the completeness of the model. Bug part libraries are used as the basis for student modelling with Condition Induction. Given two consecutive states, the system constructs a rule that converts one state to the other. This technique requires two complete libraries: a library of operators that converts one state to the other, and a library of predicates. An operator would be the action part of a rule and a logical combination of predicates the condition part of a rule. A student's record of past problem solving is consulted in building the rule. This technique requires a lot more data than the techniques covered so far would require in order to help the diagnosis program update the student model. For this reason a bug parts library which includes a larger number of hypotheses serves as the basis for this diagnostic technique.

Plan Recognition

Path finding followed by model tracing with or without rule induction, can theoretically diagnose everything. However, when the paths between two states are long, reliable diagnosis may become infeasible. Plan recognition is a diagnostic technique that, like path finding, may be used as a front end to model tracing. However, plan recognition requires that the target knowledge type is procedural hierarchical and all of the student's mental states in the student's problem solving are available to the diagnostic program. For example, in the case of the goal being to win a game of chess, the tree is the *plan* and plan recognition is the process of inferring a plan tree when only its leaves are available. Assuming that Plan recognition can find a unique plan tree that spans the student's actions, this can be regarded as the student's mental state which can then serve as input to a model tracer, which updates the student model accordingly.

Issue tracing

This model tracing technique assumes that the knowledge in the student model is a fairly accurate psychological model of the knowledge employed by a student. If any coarse-grained student model is available then issue tracing, a variant of model tracing, may be employed. Issue tracing works by analyzing a short episode of problem solving into a set of microskills or issues that have been employed during the episode. The analysis does not produce an account of what was the role of these issues in problem solving or how they interacted with each other but simply a list of these. Therefore a student model diagnosed by such a technique is not a detailed one but merely a list of issues that the student has used. The first step in issue tracing is to analyze the student's move and the expert's would-be move into issues. Each issue is then allocated two counters: used and missed. Used counters are incremented for all the issues in the student's move. Missed counters are incremented for all the issues in the expert's would-be move that are not in the student's move. If the used counter is high and the missed counter is low, the student probably understands the current issue. If the missed counter is high and the used counter is low, then the student probably does not understand the issue. If both counters are zero, the issue has not come up yet. If both counters are high, the model is inadequate in some way. This situation in issue tracing is called tear (as in 'rip').

Expert Systems

The basic idea of the expert systems approach to student diagnosis is to provide diagnostic rules for all the situations that arise. Guidon [Clancey, 1987], for example, uses inference rules to diagnose a student model. This approach is much more complicated than issue tracing. Usually students have more than one missing conception or misconception. The techniques covered so far assume that at most one rule fires between consecutive mental states, so each missing conception or misconception will show up in isolation as a buggy rule application. Because bugs appear in isolation, each bug can be accurately diagnosed even when there are several of them. Systems with less bandwidth use a less accurate description of missing conceptions and misconceptions, which allows them to model combinations of missing conceptions and misconceptions much more simply.

The next three techniques aim for more accurate diagnoses with bandwidths of mainly final mental states. The student models diagnosed with these techniques are based on bug libraries which contain not necessarily accurate bug descriptions. These techniques predict the sequence of intermediate states, and perhaps even the sequence of approximate mental states. Furthermore, the following three techniques aim at diagnosing multiple bugs by generating the symptoms of co-occurring bugs from the union of the symptoms these bugs display when in isolation.

Decision Trees

The decision tree technique is an approach to bug compounding and analysis of the interaction between bugs. It first appeared in the BUGGY [Brown and VanLehn, 1980] diagnostic system, a system teaching arithmetic, where the bugs library was enlarged by forming all possible pairs of bugs. With BUGGY there were 55 bugs and bugs compounding generated 55^2 (= 3025) bug pairs. BUGGY preanalysed an arithmetic problem that students were given and formed a decision tree that indexed the bugs by the students' answers to the problems. The top node of the tree corresponds to the first problem. BUGGY always generates problems in pairs. Answers from all possible diagnoses, either a bug or a bug pair, are collected. For each generated answer, a daughter node is created in the tree, labelled by an answer and linked to the root node. Associated with each node are the diagnoses that gave that answer. This tree-building operation is repeated for each of the daughter nodes, using the second problem. When BUGGY is finished a large tree has been built. Each diagnosis corresponds to a path from the root to some leaf of this decision tree where each leaf corresponds to exactly one diagnosis, provided the test problems are appropriately chosen. The decision tree is constructed before any interaction has taken place. Assuming the student makes no unintentional errors or slips, then his answers are used to steer BUGGY on a diagnosis path from the top node to a leaf node, and hence a diagnosis that is appropriate.

Generate and Test

The generate and test diagnostic technique generates a set of diagnoses, finds the answers that each diagnosis predicts, tests those answers against the student's answers, and keeps the ones that match best. Generate and test is coupled with domain specific heuristics. This technique has been used in DEBUGGY [Brown and VanLehn, 1980] that was designed to diagnose up to four or five multiple co-occurring bugs. Unlike BUGGY, it does not calculate the answers of co-occurring bugs in advance, but dynamically. It finds a small set of bugs that match some, but not necessarily all, of the student's answers. It then forms pairs of these bugs and selects the ones that match the student's answers. It then forms triples of these and selects those that match the student's answers. The bug-compounding process occurs again and again until no further improvement in the match is found. The resulting tuple of bugs is the output of DEBUGGY's diagnosis of the student.

Interactive Diagnosis

BUGGY and DEBUGGY work with a set of predefined problems, the student's answers to it, and the corresponding remedial action. IDEBUGGY [Brown and VanLehn, 1980] is an Intelligent Tutoring System which can generate a problem whose answer will help the diagnosis most. Given a set of diagnoses consistent with the student's answers, it tries to construct a problem, an arithmetic operation, that will cause each diagnosis to generate a different answer. Interactive diagnosis, where the diagnosis algorithm drives the tutorial interaction, puts heavy demands on the speed of the diagnostic program. However, it yields highly accurate diagnoses with much less predefined problems.

1.4 THE TUTOR MODEL

The third key place for intelligence in an Intelligent Tutoring System is in the principles by which it tutors students and in the methods by which its applies these principles. Clearly, human tutors are instructionally effective only when they possess both kinds of intelligence: Domain knowledge, and effective Tutoring ability. Human tutors cannot tutor effectively in a domain in which they are not expert, and there are also inarticulate experts who make terrible instructors. Automated tutors can use many different instructional techniques, but tutorial interactions, however they are conducted, must exhibit three characteristics [Halff, 1988], [Woolf, 1991]:

[1]. A tutor must exercise some control over curriculum, that is, the selection and sequencing of material to be presented to the student and some control over instruction, that is the process of the actual presentation of that material to the student.

[2]. A tutor must be able to respond to student's questions about the subject matter.

[3]. A tutor must be able to determine when students need help in the course of practising a skill and what sort of help is needed.

There are three inter-related cental issues that underline the development of any kind of Domain Tutor for an Intelligent Tutoring System: the *nature of learning*, the *nature of teaching*, and the *nature of the subject matter* [Lesgold, 1988] [Brown, Collins and Duguid, 1991]. Some Domain Tutors are primarily concerned with teaching factual (declarative) knowledge and inferential skills. These are the *expository tutors*. They teach students a body of factual knowledge and the skills needed to draw first-order inferences from that knowledge.

Other Domain Tutors are primarily concerned with teaching skills and procedures that have application outside the tutorial situation. These are the *procedural tutors*. Tutors of this kind are concerned with teaching the procedures that manipulate factual knowledge. As a result, procedure tutors function much more like coaches. They present examples to exhibit problem-solving skills, and they pose exercises for purposes of testing and practice.

1.4.1 Curriculum: Propaedeutics, Selection and Sequencing

Curricula in Intelligent Tutoring Systems serve several functions [Lesgold, 1988] [O'Neil, Slawson and Baker, 1991]:

[1]. A curriculum should divide the material to be learned into manageable units. These units should address at most a small number of instructional goals and should present material that will allow students to master them.

[2]. A curriculum should sequence the material in a way that conveys its structure to students.

[3]. A curriculum should ensure that the instructional goals presented in each unit are achievable.

[4]. Tutors should have mechanisms for evaluating the student reaction to instruction on a moment-to-moment basis and for reformulating the curriculum.

The problem of curriculum can be broken down into two problems [Halff, 1988]: formulating a representation of the material in the Domain Expert, and selecting and sequencing concepts from that representation. In addition, Halff [1988] argues that a Domain Tutor must also incorporate some form of propaedeutics, that is knowledge which is needed to enable learning but not for achieving proficient performance.

Propaedeutics

There have been tutoring systems where a knowledge representation was suited for instruction but not to a skilled performance [Leinhardt and Greeno, 1991]. With such propaedeutic representations which serve to support performance up to an intermediate level, there is an underlying assumption that skilled performance will be achieved only with practice. Propaedeutic representations have two characteristics: first, they make explicit the functional basis of the procedures used in exercising the skills, and second, they are manageable with the limited cognitive resources that are made available to students. As a result, they serve, firstly, to relate theory to practice, secondly, to justify, explain, and test possible problem solutions, thirdly, as a stepping-stone to more efficient problem- solving strategies and, fourthly, as strategies for management of the working memory during intermediate stages of learning.

Selection and sequencing

The differences between expository and procedure tutors are evident in the problems associated with selecting and sequencing material to the student. Procedural tutors need to establish the ordering of the subskills of the target skill and the selection of exercises and examples to reflect that order. With expository tutors, the problems are, in addition, those of maintaining focus and coherence and of covering the subject matter in an order that supports later retrieval of the concepts being taught [Halff, 1988].

Curricula and topic selection in expository tutors must deal with two sources of constraints. One set of constraints arises from the subject matter: topics must be selected to maintain coherence and to convey the structure of the material being taught. A second set of constraints comes from the tutoring context. Selection of some topic or fact for discussion must reflect the student's reaction to previous tutoring events.

In either case, the methods used to construct curricula must reflect the structure of the material [Halff, 1988]. Procedural skills are nearly always taught by exercise and example. In these cases, the major curricular issue is that of choosing the correct sequence of exercises and examples. Ideally, the choice of exercises and examples should be dictated by a model of learning but in practice the lack of a precise computational learning theory makes this impossible.

1.4.2 Instruction: Presentation Methods, Answering Questions, Tutorial Intervention

The tutor may use different methods to deliver a curriculum. These methods cover initial presentation of the material, ways of responding to students' questions and the conditions and content of tutorial intervention [O'Neil, Slawson and Baker, 1991].

Presentation Methods

The methods used to present material depend on the subject matter and the instructional objectives of the Intelligent Tutoring System. Expository tutors mainly use dialogue as the form of communication whereas procedural tutors use examples and coached exercises to develop those skills. With tutorial dialogues, teaching facts and concepts is accomplished by asking for or explaining the material. The decision to ask or tell is made on the basis of the importance of the material and the student's knowledge thereof. Teaching of rules in tutorial sessions usually involves inducing the student to consider the relevant data and to formulate the rule. This can be achieved by presenting case data that makes the rule clear or by entrapment strategies that enable the student to eliminate incorrect versions of the rule.

Skills for deriving rules are taught as procedures. These procedures are broken down into their components (e.g. listing factors or generating cases to specification) and exercises and examples are provided that address each subskill. Instructional modelling, the use of worked examples or guided practice, is a prime vehicle for introducing students to procedures that they must learn. Essential to the success of modelling in Intelligent Tutoring Systems is the formulation and presentation of procedures for working the examples. These procedures must be based on the representations (including propaedeutic ones) that students need to acquire the target skills, and they must be presented to the student in a manner that shows how each step applies to the case being modelled.

Answering Questions

Effective answering of questions is related to the difficulty of natural language comprehension and generation which has been described as the Achilles' Heel of any effort on Intelligent Tutoring Systems development.

Tutorial Intervention

Tutorial intervention is needed in order to maintain control of the tutorial situation to protect the student from inappropriate or incorrect learning, to keep the student from exploring paths that are not instructionally useful, and to speed the course of instruction. This involves devising rules for deciding when or when not to intervene and formulating the content of the intervention. There are two major approaches to decisions about tutorial intervention.

First, model tracing which calls the tutor to intervene whenever a student strays from a known solution path. A tutor using this technique maintains a model of the student's cognitive processing as the student moves through an instructional unit. This model aims to reflect the cognitive processes of a competent performer in the instructional setting. As the student progresses, the model traces that behaviour, attempting to match it to one of the paths that could be taken by the ideal student. When the matching process fails, the tutor intervenes with advice that will return to a successful path.

Second, issue-based tutoring calls the tutor to intervene when the tutor can make a positive identification of a particular occasion for intervention. It does not restrict its intervention just to remedial instruction. Furthermore, issue-based tutors can be more informative in the content of their intervention, since they can speak about the issue that caused the intervention. Issue-based tutors do not require perfect expert models to run with. While model-tracing tutors will intervene even when the student finds a 'better' or alternative approach than the expert model, issue-based tutors will remain silent in these circumstances. When a tutor decides to intervene it must also formulate the content of the intervention. There has been no uniform approach to the content of intervention among the few existing computer coaches. The obvious technique, to directly correct the problem that caused the intervention, is not in general used because informing the student of the low level actions needed to recover from a bad situation does not generally constitute a viable context for instruction. Goldstein [1982] suggests that naive users making an error must receive suggestions of a coarse nature whereas advanced students making the same error must receive more detailed advice.

1.5 USER INTERFACE

The interaction between students and Intelligent Tutoring Systems is inherently complex because the users of these systems are, by definition, working with concepts they do not understand well [Bonar, 1991]. Consequently, a well-designed interface can add considerably to the way in which the student will conceptualise the problem domain, as well as over the vocabulary the student will use to talk about the domain [O'Malley, 1990]. Human interface techniques affect two aspects of Intelligent Tutoring Systems [Miller, 1988]. First, they determine how students interact with the Intelligent Tutoring System. A well-designed human interface allows the Intelligent Tutoring System to present instruction and feedback to students in a clear and direct way [Baker, 1990]. Similarly, it can provide students with a set of expressive techniques for stating problems and hypotheses to the Intelligent Tutoring System. Second, they determine how students interact with the domain that is being tutored, through either a simulation of the domain or direct connection to the domain itself. This interaction is generally tied closely to the tutorial component of the system so that actions in the domain are analyzed and acted upon.

A tutorial interface defines the way that students think about the concepts in which they are being taught. Human-computer interaction in such terms is not a mechanical exchange of actions, but a communication of concepts, a semantic process [Miller, 1988] in which the interface reflects the semantic nature of this interaction [Streitz, 1988]. The interface needs to embody an understanding of, and appreciation for, the goals and concepts that are important to users and in the domain being tutored. Consequently, it needs to embody an understanding of the user's cognitive abilities and limitations, and the domain to which the interface serves as a portal. Therefore, the important issue is not the application area of the interface but the definition of the ways in which good interfaces can support people as they gradually acquire an understanding of a complex semantic domain.

Based on the overall structure and orientation of the interface to the user, that is the perceived relationship between the user and the domain addressed by the computer system, interfaces can be divided into two groups [Bonar, 1991]: interfaces that allow users to become direct participants in the domain, or interfaces where the users control the domain by instructing an intermediary to carry out actions in the domain.

1.5.1 First-person interfaces

In first-person interfaces or direct manipulation interfaces, the user has a feeling of working directly with the domain. These interfaces allow users to carry out desired actions by manipulating objects. Such interfaces are designed so that the actions and objects relevant to the task and domain map directly to actions and objects in the interface. The underlying mechanism behind such systems are almost always *icons*, which are small pictures on the screen which when selected by the user trigger some action. Icons represent data structures and procedures, and links between these objects specify how the procedures are to be applied on the data.

Although first-person interfaces appear to offer significant advantages to users, some aspects of the system's functionality may not be self-evident to the inexperienced user. In such cases, the Intelligent Tutoring System may have to explain the different capabilities of the system to a user. Furthermore, the link between the semantics of the domain and the semantics of the interface may be fuzzy. The problem here is how much of the underlying application is conveyed through the model for the users to understand which parts of the system they can directly manipulate.

1.5.2 Second-person interfaces

With second-person interfaces, users interact with the domain by giving commands to a computerised intermediary, which then carries out the desired actions.

Command languages, typical second-person interfaces, are keyword-oriented interfaces in which a command consists of a string of words and sometimes special characters that, when processed by the system's command interpreter, specify the action the user wants to carry out. With ***Menus***, a list of options is shown to the user, who selects the desired option by striking a key. Menu-based systems stand between first-person and second-person interfaces: being presented with information and selecting some of this information is a characteristic of second-person interfaces, whereas the direct way in which the user can specify the information is characteristic of first-person interfaces.

With a ***natural language interface***, the most popular user interface to an Intelligent Tutoring System, users communicate in a language they already know with an agent that can interpret their requests for action to be triggered. Human computer interaction in natural language is normally restricted to some form of stylised English. Full coverage is difficult because natural language interfaces are second-person interfaces in which the style of interaction is that of speaking to an intermediary who will carry out the requested actions. There have been many approaches in developing natural language interfaces: symbolic pattern matching, sub-languages, semantic grammars, context-free grammars, generative grammars, etc. [Miller, 1988].

1.5.3 Alternative interface technologies

Developmental changes in the hardware platform on which these interfaces are presented are relevant to the communication needs of tutoring systems and allow information to flow much more directly between the Intelligent Tutoring System and the student. These developments can contribute to the primary design goal of a good interface: to make the semantics of the domain evident and manipulable. *Graphics technology, large and small* *displays, videodisks, CD-ROMs, touch screens and digitising tablets, speech recognition and understanding, speech coding and synthesis* have been used to bring together multiple colours, multiple windows, menus, icons, animations, two-dimensional and three-dimensional images, digitised information and images, pointing devices, finger sensing, voice processing, digitised and synthetic speech as part of the same Intelligent Tutoring System interface.

1.6 RESEARCH OBJECTIVES AND METHODS

This Chapter and most of Appendix A show that the vast majority of existing Intelligent Tutoring Systems have been developed as knowledge based systems. There have been many reasons why Knowledge Expert Based Systems seem to offer an ideal basis on which to build Intelligent Tutoring Systems, other than the obvious fact that they embody large amounts of expert knowledge! One advantage of these systems is the separation of the (usually) production rules in the knowledge base from the procedural interpreter that uses them. This allows access to modular pieces of knowledge, which are expressed declaratively and can often be understood independently. In addition, explanation facilities have been developed to justify the behaviour of some systems. These can trace the chains of inferences, thus offering explanations of both how the reasoning has led to the conclusions the system proposes and why the system needs certain pieces of information when it requests data from the user. A Knowledge Based Expert System with good explanation capabilities can none the less only justify its actions passively. To be able to present knowledge actively, it is acknowledged that an Intelligent Tutoring System must be endowed with the ability to select instructional material, to be sensitive to the student and to conduct an effective interaction.

The research objective of the thesis is to investigate the nature of interaction of the three knowledge models (domain, student and tutoring Knowledge models) within a general architecture of an Intelligent Knowledge Based Tutoring System. This is achieved through an investigation of their interrelatedness and interconnectedness during the course of interaction.

The research method that has been followed is the traditional Empirical Information Systems research approach, the *Laboratory Experiment.* The purpose of a *Laboratory Experiment* is to improve the efficiency and effectiveness of the Intelligent Tutoring System in Practice and to examine the impact that system behaviour has on the individual in terms of its Architecture. The key feature of this research method is the identification of the precise relationships between variables in a designed laboratory situation, using quantitative analytical techniques in the hope of making generalisable statements applicable to real-life situations. The strength of this research is the isolation and control of a small number of variables which may then be studied intensively. The weaknesses of this research approach is the limited extent to which identified relationships exist in the real world due to over-simplification of the experimental situation and the isolation of such situations from most of the variables which are found in the real world.

1.7 THESIS OUTLINE

Chapter 1 has given an overview of the three knowledge models that make up the Intelligent Tutoring System and showed that the vast majority of existing Intelligent Tutoring Systems have been developed as Knowledge Based Systems. Consequently, the thesis will pursue its objective (i.e. the investigation into the interrelatedness and interconnectedness between the three knowledge models), by concentrating on existing

Intelligent Knowledge Based Tutoring Systems.

Chapter 2 presents Wenger's model of a didactic operation which provides the framework within which the interrelatedness and interconnectedness of the three knowledge models presented in Chapter 1 will be examined. This model does not explicitly state what the nature of interaction between the three components should be, but it does serve to explain the behaviour of an Intelligent Tutoring System that follows a full-scale didactic operation.

At this stage the thesis will suggest that to continue with the investigation, an evaluation that examines the relationship between such a system behaviour and the architecture for existing Knowledge Based Tutoring Systems is required. This would help uncover what the requirements for interrelatedness and interconnectedness between the three knowledge models should be in the context of the didactic operation. Chapter 3 introduces two existing Knowledge Based Tutoring Systems, PROUST and micro-SEARCH, that are used in the evaluation exercise that is discussed in Chapter 4.

The evaluation of PROUST and micro-SEARCH in Chapter 4 against Wenger's Model of a didactic operation yields several requirements with respect to interrelatedness and interconnectedness between the three knowledge models. The evaluation also highlights a number of limitations of the knowledge based systems approach to developing a tutoring system with a full-scale didactic operation.

Chapter 5 proposes a hybrid model made up of Artificial Intelligence and Hypertext that seeks to overcome the limitations of existing Knowledge Based Tutoring Systems with respect to the requirements for the development of an Intelligent Tutoring Systems with a full-scale didactic operation.

Chapter 6 explains how to use the model derived in Chapter 5 to design a generic model of an Intelligent Tutoring System with a full-scale didactic operation. The model caters for the design of an open and scalable system that allows for a variety of system components, such as domain, student and tutoring knowledge, to be combined into a single model while allowing for additional knowledge models to be included at a later stage.

CHAPTER 2: WENGER'S MODEL OF A DIDACTIC OPERATION

This Chapter presents Wenger's [1988] model of a didactic operation which is used to provide a framework in which the architecture of an Intelligent Tutoring System and the interrelatedness and interconnectedness of the three knowledge models presented in Chapter 1 will be examined. The underlying idea behind this model is that all three forms of knowledge are organised around the model of the domain knowledge. The model of a didactic operation also assumes the existence of a pedagogical process model that provides some global coordination of the system's didactic operation. According to Wenger's model, the architecture of an Intelligent Tutoring System is as shown in Figure 2.1 below. The architecture is similar to that presented in Chapter 1 in Figure 1.2.

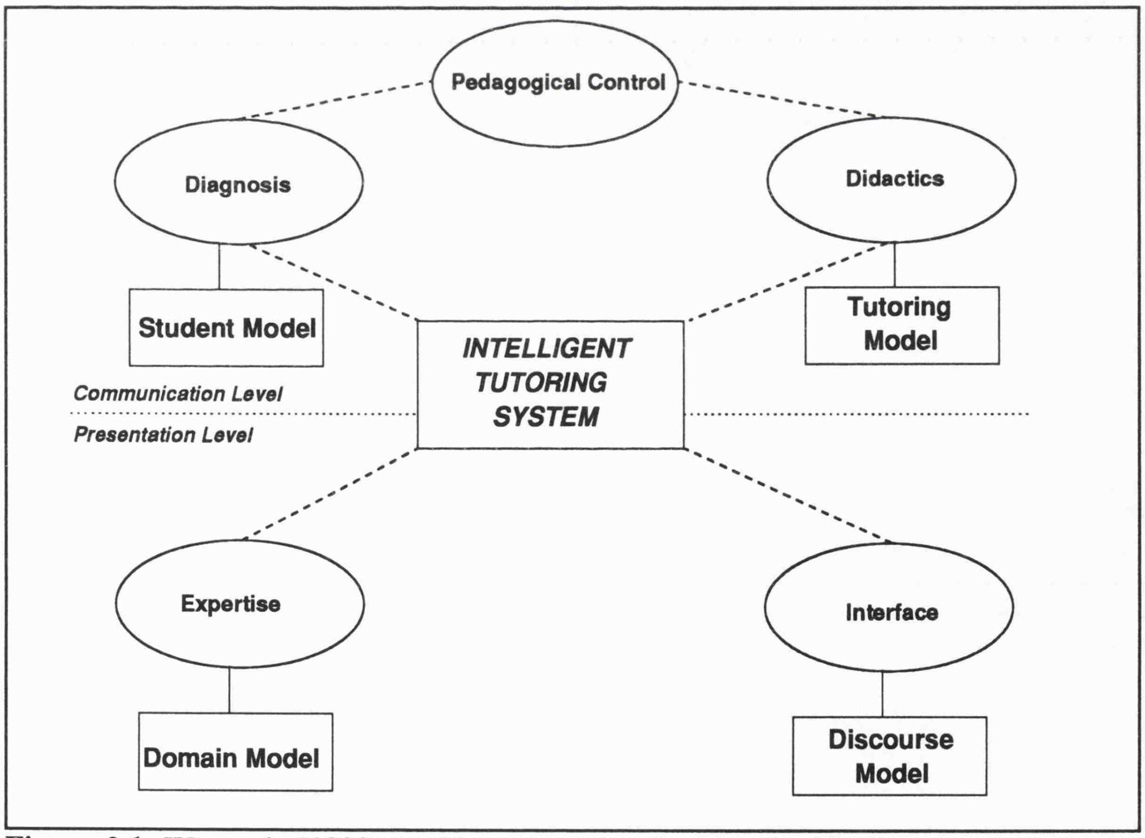


Figure 2.1: Wenger's [1988] Intelligent Tutoring Systems Architecture

The model assumes that the two process models in the lower half of Figure 2.1 do not directly take part in major pedagogical decisions. Instead, they support the tutoring process by making the representation of domain knowledge available. The domain expert knowledge process model can directly answer student questions or provide information to other models about the domain if and when they need it. To extract useful information from the domain knowledge representation, the domain expert knowledge process model applies reasoning processes. The interface process model translates the flow of information to and from the student-user.

The activities of the process models in the upper half of Figure 2.1 result in decisions that shape the course of instruction. Didactics refers to pedagogical activities intended to have a direct effect on the student, as opposed to diagnostic activities. The task of these activities is to create a pedagogical bridge between the tutoring model, the domain expert model and the student model. Strategies for dealing with this take the form of pedagogical plans that incorporate fixed sets of diagnostic expectations along with mechanisms for dealing with common student problems, like misconceptions. There are three classes of circumstances each with different implications for the respective roles of student diagnosis and the didactic operation. In *opportunistic* pedagogical strategies, the monitored activities provide a focus for both diagnostic and didactic activities but diagnosis is the driving force because it reveals opportunities for tutorial interventions. Pedagogical goals are associated with diagnostic units and their attainment is monitored by differential modelling. In *plan-based* pedagogical strategies, the main task of diagnosis is to monitor the implementation of teaching plans that embody pedagogical goals. These plans provide a focus for diagnostic activities with the consequence that differential modelling is performed in terms of plan failures so that revisions can be made.

The pedagogical module is responsible for optimising the interplay of diagnosis and didactic operations thus providing for a coherent pedagogical strategy. According to Wenger, it is central to an Intelligent Tutoring System because it includes decision making about the degree of control exercised by the system, the choice of teaching strategy to apply, the selection of strategic contexts (i.e. opportunistic versus plan-based contexts) the interleaving of pedagogical episodes, the allocation of computational resources required by competing functions and the target level of operations (i.e. behavioural, epistemic or individual). All these decision making aspects conjecture the didactic operation which is the mechanism by which the pedagogical process model drives the interaction in order to attain its pedagogical goals.

2.1 THE MODEL OF A DIDACTIC OPERATION

Wenger [1988] defines a didactic operation to be a unit of decision in the tutoring process. It is more general than a didactic intervention, in that it does not necessarily correspond to actions visible to the student. A *didactic operation* has four characteristic aspects as shown in Figure 2.2: the *plan of action* that enacts a didactic operation, the *strategic context* in which the operation is triggered, the *decision base*, that provides *constraints* and *resources* for the construction of the operation, and the *target level* of the student model at which the operation is aimed.

2.1.1 Didactic Plan of Action

A framework for defining didactic operations is the concept of a *plan*, because it can be made general enough to encompass all situations: from the simple prestored interventions of current systems to the dynamic knowledge communication capabilities. A curriculum is a plan, nevertheless, the concept actually applies to most didactic activities. Even a local explanation, for instance, can be considered a plan since an explanation rarely

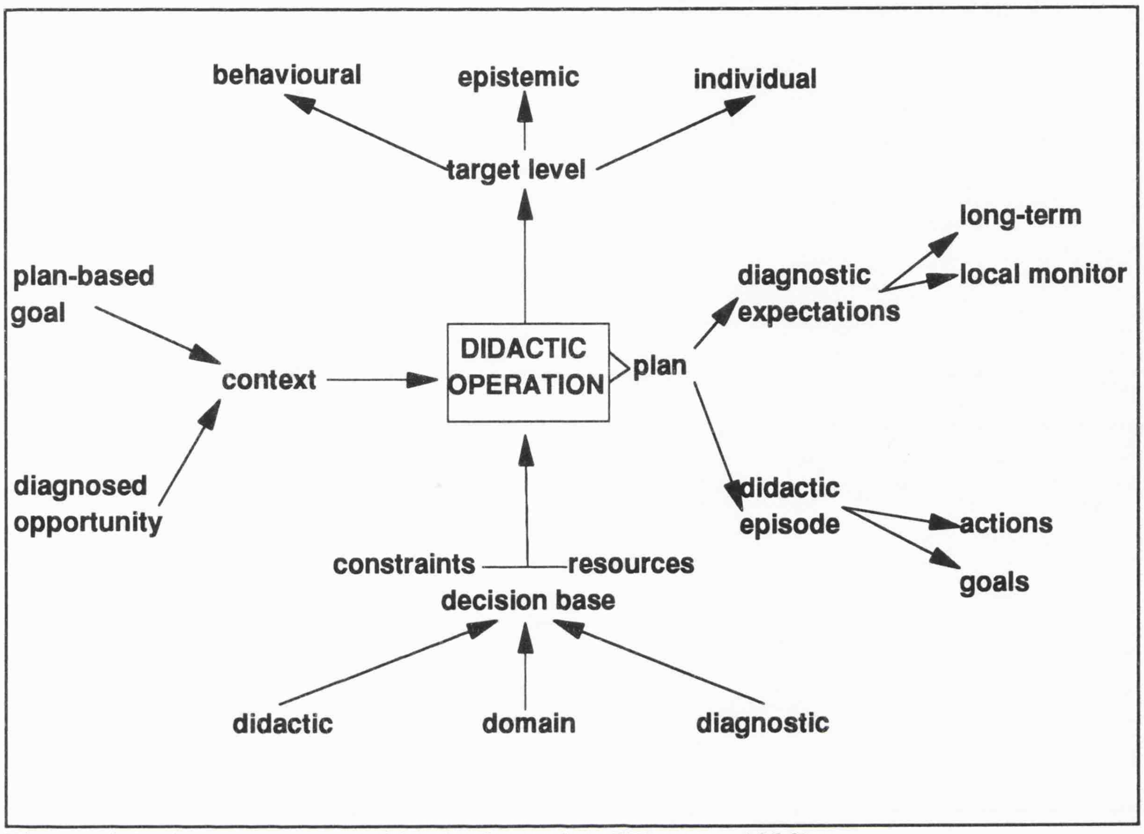


Figure 2.2: Aspects of the Didactic Operation [Wenger, 1988]

consists of a single conceptual element. A non-trivial explanation is a plan, a kind of mini-curriculum, for leading the student along a local learning path.

In addition to generating an *episode* of actions or subgoals, a didactic operation can also generate explicit *diagnostic expectations*. Planning distinguishes situations in which the actual effects of operations in an episode can be predicted accurately, from those in which they cannot, and in which the execution of a plan requires some monitoring. Diagnostic expectations that articulate goals and possible outcomes can be both local, monitoring the unfolding of a plan, and global, building up a long-term context and creating continuity throughout the tutorial session. Most didactic operations in today's Intelligent Tutoring Systems consist of a single basic action, such as presenting a piece of prestored text or submitting a selected problem.

2.1.2 Pedagogical Contexts: Opportunistic versus Plan-based

Instruction has goals which can be achieved either through intentional planning activities, by which one gains control over the environment, or through the recognition of opportunities presented by the environment's resources. Goals are best achieved by an appropriate combination of both styles. Hence, in attaining teaching goals and in generating subgoals, *opportunistic* and *plan-based* approaches define a range of pedagogical styles that vary in the Intelligent Tutoring System's control over the shape of the tutorial sequence. The degree of this control determines different *triggering contexts* for didactic operations and suggests different roles for diagnosis and didactics.

Opportunistic strategies take advantage of teaching opportunities that arise in the context of some activity or dialogue in which the student is engaged. If the environment is rich and structured enough, instructional goals may be eventually achieved, and the student's activities or statements can provide a focus for diagnosis and hence for the content of tutorial interventions. If the strategic context is loose, teaching opportunities may be revealed by diagnostic information, and planning can be locally focused on these opportunities. As a consequence, the presentation of the material is driven by the student's interaction with the environment. Nevertheless, the adoption of an opportunistic strategy does not necessarily imply that the student is given a greater amount of freedom.

Although the pedagogical expertise required for intelligent opportunistic interventions can be quite sophisticated, these strategies are suited to teaching *in situ*, for problem-solving guidance or coaching in learning environments, especially if the tutored activities complement other kinds of teaching such as formal instruction. In such cases problem-solving environment often contains an implicit plan or curriculum, in the form of a pool of activities or topics ordered or otherwise, that the interaction is expected to cover. However, opportunistic Intelligent Tutoring Systems have little control over how the organisation of instructional sessions communicates the subject matter to the student. This limits their adaptive monitoring of the student's learning and their usefulness as a primary source of instruction.

In plan-based contexts, pedagogical goals predominate and their attainment is dynamically controlled by the Intelligent Tutoring System, which organises the activity and the interaction around them. Therefore, planning tends to be hierarchical. Although the structure of the environment and the student's behaviour play a less central role, the student does not necessarily have less freedom.

A plan based context simply means that the tutor manipulates the sequences of experiences through which the student is expected to acquire the target expertise. Thus the Intelligent Tutoring System plans learning events, globally or locally, even when the student enjoys a great deal of freedom within this context, and this plan provides the focus for didactic and diagnostic activities. This also changes the function of diagnosis in a subtle way from triggering interventions to monitoring an unfolding plan. In those Intelligent Tutoring Systems where the order of topics is not predefined, diagnostic information is combined with local optimisation criteria to determine good exercises or issues to attack next.

Within a globally opportunistic strategy, the tutor can take control with local interventions that are strongly plan-based in order to get a focused point across to the student. Within a plan-based strategy, new goals can emerge in a completely opportunistic fashion, taking advantage of unexpected events. A complex interleaving of such embedded contexts with alternating opportunistic and plan-based strategies can result in a 'Socratic' dialogue similar to that which might be led by a human tutor. This suggests the need for an *internal agenda*, which can keep track of active subgoals and emergent goals and provide a complex triggering context for didactic operations.

2.1.3 Decision Base: constraints and resources

Didactic operations must comply with a number of *constraints*, which ensure their effectiveness but which often imply the resolution of conflicts between various competing factors affecting decisions. Also didactic operations require *resources* as building materials and whose limitations are also an implicit source of constraints. In particular, the triggering context provides important constraints and resources by focusing didactic operations on recognised opportunities or prevailing plans.

There are three major sources of both constraints and limitations for Intelligent Tutoring Systems: *didactic*, *domain*, and *diagnostic* information, as shown in Figure 2.3, each corresponding to a knowledge model.

What Wenger calls the 'Didactic Base', that is the tutoring knowledge and process model, is a source of local tutoring tactics and global teaching strategies. Local tactics refers to situation-specific or domain-specific goals and plans that will be applied in the context of a didactic episode. Global strategies refers to domain-independent teaching strategies that are suitable for tutoring with the domain knowledge. An Intelligent Tutoring System's teaching strategies, especially for material sequencing, provide good examples of didactic constraints in the form of pedagogical principles.

	constraints	resources	
		local	global
diagnostic	relative strengths and weaknesses	integration remediation	means-ends analysis
domain	relative importance of topics	content of interventions	sequencing
didactic	pedagogical principles	tactics	strategies

Figure 2.3: Aspects of the Decision Base of the Didactic Operation [Wenger, 1988]

These constraints often interact with other requirements, competing and cooperating with them. Wenger's model indicates that, at the global decision making level, domain-independent teaching strategies interact with the domain knowledge for material sequencing, with the representational syntax of the domain knowledge being a determining factor. The means-ends analysis of the student, that is the classification of the user as a learner (e.g. as a novice, advanced beginner, etc.) also influences the choice of strategy that is to be used and the level of detail. In the context of a didactic episode, when a teaching strategy is used to attain a teaching goal, it should also have access to the corresponding knowledge in the domain knowledge model. In addition, it should have access to the student overlay model in order to check for attained goals and missing concepts from goals that have been attempted.

What Wenger calls the 'Domain Base', that is the domain knowledge and process model, is a source to the tutoring knowledge process model for global domain material sequencing and for providing material for the content of local tutorial interventions, for example, explanations in problem-solving episodes. The organisation, structure and functionality of the domain knowledge is a source of constraints about the relative importance of topics and the functionality of the tutoring strategies.

Wenger's model indicates that the domain knowledge model, when augmented with the two other knowledge and process models, is a source of domain information. At the global decision making level, the tutoring knowledge process model requires access to domain material, and the student knowledge process model performs means-ends analysis by differential modelling which involves comparing the student model with the domain expert model, perhaps with an overlay technique.

In the context of a didactic episode, the goals that the student-user has to attain, and the knowledge that the student already has acquired, relate to specific domain knowledge. The domain knowledge model is a source of information for any missing conceptions in the corresponding local student overlay model, for diagnosis of any student errors and for providing relevant remedial information. The domain knowledge will serve to provide the (correct) knowledge for diagnosing the student's perception of the concept that a didactic episode is dedicated to, and of the domain as a whole. This perception needs to be determined and incorporated into the student model.

The Diagnostic Base, that is student knowledge and process knowledge, is a source for means-ends analysis of the student-user (classifying the student-user as a learner and thus infer/assume additional information about him) and for integration of domain knowledge and remediation of misconceptions in the student model. In the student model, domain expertise acquired by the student is represented in an overlay model which represents an orthogonal variability between knowledge of the student that has been inferred and represented in the student model and knowledge in the domain knowledge model.

At the global decision making level, the student knowledge process model performs a means-ends analysis by differential modelling which involves comparing the entire student knowledge model to the domain knowledge model. The relative strengths or weaknesses of the student as a result of differential modelling can then influence the flow of the tutorial interaction.

In a didactic episode, the student model is superimposed on the corresponding part of the domain knowledge model to obtain an indication of the level of mastery of the concept that the didactic episode is dedicated to. In addition, this overlay model provides a way of determining potential candidate areas for further pedagogical action. The student knowledge process model, having access to libraries of commonly observable deviations from the correct knowledge, and access to the correct knowledge in the corresponding part of the domain base, is able to diagnose such deviations in the student behaviour.

Once a didactic operation has been triggered, diagnosis can provide further constraints by revealing weak areas of the student's knowledge by considering underlying misconceptions or missing conceptions. The first task of diagnosis is then to determine from the user input both which knowledge, correct or incorrect has been used by the student, and which relevant domain knowledge has been overlooked. This requires student diagnosis with access to missing or buggy rules about the knowledge domain, and also the knowledge used by the student to be compared to the relevant knowledge in the domain knowledge model.

With respect to tailored interventions, for instance explanations called by the student knowledge process model, the didactic operation must not only satisfy didactic principles but must also take diagnostic information into account so as to tailor their content and detail levels to individual students. This requires access to those teaching strategies that proved to be effective for the student. The value of diagnostic information as a resource is clearest in remedial situations, for instance, in an opportunistic context.

Therefore, in the context of a didactic episode the goals that the student-user has attained, and the best teaching strategy for the user, relate to specific tutoring knowledge. In addition, the knowledge which the student has already acquired, that which is missing from the student knowledge model or that which is a source of misconceptions also relates to specific domain knowledge.

2.1.4 Target Levels: Behavioural, Epistemic and Individual

The *target level* of a didactic operation is the level of the student model at which an operation seeks immediate modifications. The target level my be behavioural, epistemic or individual. Thus different target levels define different classes of instructional capabilities and strategies. Selecting the target level or levels to which an operation should be addressed is an important didactic decision.

At the behavioural level didactic interventions guide the performance of a task without addressing domain knowledge in any direct or organised fashion. Thus, they can be constructed dynamically even when only compiled knowledge (black box knowledge) is available. From this target level perspective, hints or pieces of advice are different from explanations, and corrections are different from remediation since they only address behaviour. Also simple demonstrations and traces of reasoning are restricted to exposing faulty behaviour in the domain without providing other forms of support to learning. Finally, situations only expose the behaviour of objects in the form of manipulations of the simulated environment. In such a case the didactic operation requires interpretation by the student in order to be converted into useful knowledge. The pedagogical assumption with this target level is that students will be able to acquire the correct expertise by being repeatedly exposed to problems. This involves conceptual understanding beyond that of the Intelligent Tutoring System.

Didactic operations targeted at the behavioural level capitalise on the fact that performing a task and being exposed to an environment constitute a valid learning context which provides students with raw material for actively forming their own conceptualisation of the domain. Student interpretation of difficulties and errors can be turned into a learning experience, if these difficulties and errors are properly resolved. Thus didactic operations targeted at the behavioural level support the acquisition of knowledge *in situ*, provided the Intelligent Tutoring System is conducive to the types of interpretation that can warrant beliefs.

At the epistemic level, didactic operations explicitly seek to modify the student's knowledge state, either via direct communication of domain knowledge or via practice, by organising specific experiences to expose the student to. At this level, explanations are central to dealing with the articulation of knowledge. Unlike behaviourally oriented interventions, explanations explicitly supply some of the interpretations of phenomena that serve as warranting processes for the student. At this level direct modifications in a knowledge state and within the different dimensions of variability between knowledge states are sought.

The dimensions of a knowledge state provide for a framework for a taxonomy of didactic primitives. For instance, a statement with some examples may be enough for presenting a new assimilable fact, whereas the correction of a misconception may require confrontation, corrective suggestions and explanations. Wenger argues that a student needs to be actively engaged in problem-solving in order to perceive problems. Consequently, his viewpoint can be determined by an iterative process of uncovering current limitations and discovering new problem-solving capabilities that demonstrate a new viewpoint's conceptual superiority.

At the individual level, didactic operations deal with the management of the learning process. These management tasks deal with dimensions of the individual model that have an impact on learning: motivation, cognitive load, interpretation of the instructional context. At this level, the purpose is not to communicate knowledge directly, but to maintain knowledge communication. The line between epistemic and individual levels of the student model is fuzzy. If individual dimensions and learning strategies become open to direct communication, they become knowledge that can be taught.

2.2 KNOWLEDGE PRESENTATION VERSUS KNOWLEDGE COMMUNICATION

The dividing line in Figure 2.1 distinguishes between two classes of Intelligent Tutoring Systems: knowledge communication systems, which require all the process models and knowledge in the diagram, and more passive systems, which require only those in the lower half, and which can be called knowledge presentation systems.

Although they entail a subset of the process models of knowledge communication systems, knowledge presentation systems implement a different less sophisticated pedagogical approach. By simply making knowledge available rather than actively communicating it, they leave most of the responsibility of managing the learning process to the student-user, who acts as his own tutor. He is expected to have enough understanding of the domain and of his learning needs to decide what to explore or to focus on next, as well as to interpret what is presented. A knowledge presentation system simulates knowledge about the domain under study, thus the student does not explore the domain but the knowledge about the domain.

Because of the emphasis on knowledge, it is advantageous to view knowledge presentation systems as a subset of knowledge communication systems. Knowledge presentation requires a model of communicable domain knowledge as an active communication but it does not have to assume involvement with the student's knowledge states.

From an educational standpoint, an Artificial Intelligence-based learning environment is quite attractive. Such environments provide students with the freedom to explore and a sense of control as they investigate a domain within a simulative context geared toward both operational knowledge and articulate conceptualisation.

Even in the context of exploratory learning, augmenting the presentation level with active knowledge communication capabilities involving the modules of the upper half of Figure

2.1 usually extends the benefits derived from the instructional use of computer tutors. The unobtrusive interventions of a coaching component can save the student from problems typical of unguided learning such as stagnating, floundering excessively, or overlooking learning opportunities.

As an Intelligent Tutoring System assumes a more active pedagogical role and takes some dynamic responsibility for the students learning, the nature of its internal model of expertise becomes crucial. This model provides the language in terms of which the Intelligent Tutoring System can assess needs in order to adapt its actions. Therefore, intelligence at the pedagogical level is not possible without intelligence at the domain, student and instructional level, and the requirements of knowledge communication are more stringent than those of presentation. This argues that fully operational and articulate process models of domain expertise are indispensable for constructing process models of communication functions. Bringing more intelligence into knowledge communication requires an understanding of the communication environment in which it takes place.

This Chapter presented Wenger's model of a didactic operation which provides the framework in which the interrelatedness and interconnectedness of the three knowledge models will be examined. The model of didactic operations assumes the existence of a decision base comprising of the domain, student and tutoring knowledge models. Secondly, it assumes a, or a combination of three, target level for the didactic operation: behavioural, epistemic or individual. Thirdly, it assumes a pedagogical context for the application of the didactic operation (i.e. the context and the nature of the man-machine interaction). Finally, it assumes an explicit didactic plan of action which defines the flow of tutorial interaction.

The model does not explicitly state what the nature of interaction between the three components should be. It only serves to explain the behaviour of an Intelligent Tutoring System that pursues a full-scale didactic operation. To continue the investigation, an evaluation that aims to examine the relationship between the system behaviour, as it is assumed by the didactic operation, and its architecture is required. This would help understand the nature of interaction, and unravel the requirements for interrelatedness and interconnectedness, between the three knowledge models in the context of the didactic operation. This calls for a study of existing knowledge based tutoring systems in which the relationship between their behaviour and architecture with respect to the didactic operation is examined.

CHAPTER 3: KNOWLEDGE BASED TUTORING SYSTEMS: PROUST AND micro-SEARCH

This Chapter introduces two Knowledge Based Tutoring Systems, PROUST and micro-SEARCH, that will be used for the evaluation exercise in Chapter 4. The Chapter gives a detailed account of their architecture and resulting functionality. There are four reasons why these two systems have been selected for the evaluation exercise. First, they have been labelled by the Intelligent Tutoring Systems community as representative of Knowledge Based Tutoring Systems [Wenger, 1988]. Second, they are two of the few that have been used in real environments. Third, they are available to the wider audience. Fourth, they are well documented. The remainder of this chapter discusses the architecture and functionality of each of these systems.

3.1 PROUST: AN AUTOMATIC DEBUGGER FOR PASCAL PROGRAMS

PROUST [Johnson and Soloway, 1985] [Johnson and Soloway, 1987] is a knowledge-based tutoring system for Pascal Programs Analysis. PROUST looks for both syntactic and semantic bugs in PASCAL programs written by beginner programmers. Whenever students attempt to compile a program, and the program compiles successfully, PROUST is automatically invoked to analyse the program. Any bugs that are present in the program are reported by PROUST to the student.

PROUST is not confined to some narrow class of bugs, but is designed to find every bug in most novice programs. When students are assigned moderately complex programming problems, PROUST is capable of identifying correctly all the bugs in over 70% of the programs that students write. When PROUST finds a bug, it does not simply point to the lines of code which are wrong, it also determines how the bug could be corrected. It even suggests why the bug arose in the program in the first place. PROUST came out of the MENO Project [Woolf and McDonald, 1984] which was an attempt to built an Intelligent Tutor for novice Pascal programmers which would assign programming exercises to students, read over their work, and give them helpful suggestions. However, the objective with PROUST is to reconstruct a plausible program-design process so as to provide a problem-specific context for the recognition and discussion of bugs rather than explaining the origins of misconceptions in programming knowledge with a generative theory of bugs.

The argument Woolf and McDonald put forward for developing PROUST was that diagnostic methods that look for bugs in computer programs by merely inspecting the code cannot cope with a wide variety of problems. Such methods fail to recognise that nonsyntactic bugs, e.g. semantic bugs, are not an intrinsic property of the fault program, but reside in the relation between the programmer's intentions and their realisation in the code. This makes code inspection insufficient and even plan-recognition techniques, when used in isolation, can be easily thrown off by faulty code and by complex interactions between various goals and between different plans that implement them. PROUST, therefore, deals directly with the variability of bugs in novice programs, variability both in the students' designs and in their bugs. Some bugs are accidental omissions, which might be easily recognised and corrected while others result from failures to reason through the interactions between program components. Each piece of a program in isolation may appear correct but, when combined, the program does not work. Still other bugs result from misconceptions about programming. In such a case, the code may appear correct to the programmer, but it doesn't do what the student expects, for reasons the student does not understand. Bugs resulting from misconceptions are the most serious and students stand to benefit the most from having such problems pointed out to them.

PROUST attempts to figure out how a program is supposed to work, along with what the program does, via information about the programming problem and knowledge about how to write programs. The system set out to identify the programmer's intentions, and this is worth the effort because knowledge of intentions makes it possible to identify more bugs as well as understand their causes. Novice programmers need help in identifying bugs, whether these are syntax bugs or semantic bugs, but especially the latter type which can cause the programs to fail after unusual inputs, result in a run-time error, or can yield the wrong output, often in paths which the student is unlikely to test [Angelides and Doukidis, 1990].

3.1.1 PROUST's Approach to Debugging

PROUST was written in T, a Lisp dialect. The original full system contains 15,000 lines of Lisp code and would normally run on a VAX 750. Micro-PROUST, an IBM PC version, was written in Golden Hill Common Lisp. Micro-PROUST is a stripped down version and as such there is a variety of tricky bugs which PROUST can identify but Micro-PROUST cannot.

PROUST's analysis of programs is based upon knowledge about the programming problem that the students are working on. The students may solve the programming problem in a variety of ways, and their programs may have a variety of bugs, but they have one thing in common: they are all trying to solve the same problem. Knowledge about the programming problem makes the variability of program solutions more manageable. It also provides some information about the programmers intentions.

In order to provide PROUST with descriptions of the programming problem, the PROUST authors devised a problem-description language with which one can describe a programming problem, and provided PROUST with a library of programming problem descriptions. Each problem description is a paraphrase, in PROUST's problem description language, of the English Language problem statement that PROUST's authors assign to the students. The rainfall problem in the Figure 3.1 below is an example of a programming assignment that PROUST deals with.

Original Problem statement

Noah needs to keep track of rainfall in the New Haven in order to determine when to launch his ark. Write a Pascal program that will help him to do this. The program should prompt the user to input numbers from the terminal; each input stands for the amount of rainfall in New Haven for one day. Note: since rainfall cannot be negative, the program should reject negative input. Your program should compute the following statistics from this data:

1. **the average rainfall per day;**
2. **the number of rainy days;**
3. **the number of valid inputs (excluding any invalid data that might have been read in);**
4. **the maximum amount of rain that fell on any one day.**

The program should read data until the user types 99999; this is a sentinel value signaling the end of input. Do not included the 99999 in the calculations. Assume that if the input value is non-negative and not equal to 99999, then it is valid input data.

Problem statement as input to PROUST (slightly simplified for readability)

```
Objects: ?DailyRain is of the class "scalar measurement"
Goals: Sentinel-controlled input sequence (?DailyRain, 99999)
         Loop input validation (?DailyRain, ?DailyRain < 0)
         Output (Average (?DailyRain))
         Output (Count (?DailyRain))
         Output (Guarded count (?DailyRain))
         Output (Maximum (?DailyRain))
```

Figure 3.1: A programming assignment for PROUST [Johnson and Soloway, 1985]

Included in Figure 3.1 is the formal description of the problem given to PROUST as input along with the student program to be analysed. PROUST would then search in its library of programming problem descriptions for the most plausible interpretation of the program with respect to the problem specifications. PROUST needs to infer a plausible design process that reproduces the programmer's intentions in an analysis by synthesis theme. The method combines reconstruction of intentions with detection of bugs, because bugs can lead to misinterpretations of intentions, and intentions are necessary to distinguish bugs from unusual but correct code!

Knowledge of the problem that the students are working on helps to provide an understanding of the students' programs. Nevertheless, this is only a description of what the program should do, not how it should do it. Solutions to a given programming problem may be implemented in a variety of different ways. PROUST therefore accesses knowledge about programming so that it can understand how each student designed and implemented his solution. Once it understands the programmer's intentions, it can then use knowledge about common bugs in order to identify the bugs in the student's program.

The method which PROUST uses for analysing programs is **synthesis**. When PROUST examines a program, it looks up the corresponding problem description in its problem description library. Using its knowledge about how to write programs, it makes hypotheses about the methods which the programmer may use for satisfying each requirement in the problem description. Each hypothesis is a possible correct implementation of the corresponding requirement. If one of these hypotheses fits the student's code, then PROUST infers that the requirement is implemented correctly. If PROUST's hypotheses do not fit the program, then PROUST checks its database of common bugs, to see if they can explain the discrepancies.

PROUST's intention-based program analysis is a comparison of intended functions and structures to actual ones. PROUST's diagnosis approach distinguishes between three levels: problem specifications give rise to an agenda of goals and subgoals, which in turn lead to the selection of plans, which are finally implemented as code. The exact set of intentions underlying a program is usually not available as data, but must be reconstructed on the basis of evidence provided by the problem specifications given to the programmer and by the program proposed as a solution.

3.1.2 PROUST's Problem Description

PROUST's problem descriptions describe the principal requirements which must be satisfied: the programming goals. Problem descriptors also describe the data which the program must manipulate: objects. Assume the following classic PROUST problem [Johnson and Soloway, 1985], known as the *averaging problem*:

> **Write a program which reads in a sequence of positive numbers, stopping when 99999 is read. Compute the average of these numbers. Do not include the 99999 in the average. Be sure to reject any input which is not positive.**

The first step in translating an English Language problem statement into PROUST's problem description language is to make explicit the various goals which were mentioned in the problem statement. Solutions to the problem operate on a sequence of input data, called NEW. The following goals can be extracted from the problem statement:

1. Read successive values of NEW stopping when a sentinel value, 99999, is read.
2. Make sure that the condition NEW <= 0 is never true.
3. Compute the average of NEW.
4. Output the average of NEW.

These goals must now be translated to a problem description for PROUST. Each data object that the goals refers to, is named and declared. Each goal extracted from the problem statement is recorded in the problem description. The resulting problem description is given below:

```
((Define-Program Average)
 (Define-Object ?New)
 (Define-Object ?Sentinel Value 99999)
 (Define-Goal (Sentinel-Control-Input ?New ?Sentinel))
 (Define-Goal (Input-Validation ?New (<= ?New 0)))
 (Define-Goal (Output (Average ?New))))
```

The problem description is in list notation, where every statement and expression is enclosed in parentheses. The name of the program is denoted by **Define-Program**. Objects are denoted by **Define-Object** followed by the name of the object preceded by ?. Object names followed by a value are constants. With the description language of PROUST objects can have a variety of properties associated with them. Finally, goals are denoted by **Define-Goal** followed by a name of a type of goal and the list of its arguments. Arguments to goal expressions can take a variety of forms. They can be objects, predicates or even other goal expressions.

3.1.3 PROUST's Programming Knowledge

PROUST analyses Pascal programs using an analysis by synthesis approach. It examines the program requirements listed in the problem description, suggests methods for implementing these requirements and then compares each possible method against the method that the student actually uses. In order to suggest the possible methods PROUST uses its own programming knowledge.

PROUST relies on a detailed knowledge base that provides information about the types of programs which is expected to encounter. The knowledge base is not an attempt to reproduce the design process that novices follow. It combines expert knowledge about programming with knowledge about programming errors.

Programming knowledge in PROUST's knowledge base is frame-based (see Figure 3.2) and each frame represents either a goal or a plan. Goals are problem requirements that appear in problem descriptions. These represent problem specifications and the ways in which they can be implemented or reformulated, implicit goals and objects that have to be inferred and can sometimes be omitted in the problem statement along with heuristic rules that can detect goal interactions and generate new goal expectations in connection with certain errors. Plans are stereotypical methods for implementing goals. These implementation lists are indexed by the goals they achieve and they also include information about incorrect applications of plans along with some buggy plans. PROUST's authors argue that a major part of the process of writing programs consists of identifying goals which must be satisfied and selecting plans which implement these goals. PROUST retrieves plans from its knowledge base for each goal referred to in the problem description. It compares these plans against the student's program to determine which fits the program best. Code consists of two types of rules to deal with plan differences: transformation rules which check for equivalence between two versions of a piece of code and bug rules that explain mismatches by hypothesising a bug of a known type.

Figure 3.2 is an extract of a goal from PROUST's Knowledge Base which is frame-based [Minsky, 1986]. The Instances slot lists the various plans in PROUST's knowledge base for implementing this goal. The filler of this slot is a list of five plan names. The InstanceOf slot indicates the class to which the goal belongs which in Read&Process and involves reading a sequence of values and then processing them.

```
(Goal-Definition Sentinel-Controlled-Input
        InstanceOf          Read&Process
        Form                (Sentinel-Controlled-Input ?New ?STOP)
        MainSegment         MainLoop:
        MainVariable        New
        MainPhrase          "sentinel-controlled loop"
        OuterControlPlan    T
        Instances           (Sentinel-Process-Read-While
                             Sentinel-Read-Process-While
                             Sentinel-Read-Process-Repeat
                             Sentinel-Process-Read-Repeat
                             Bogus-Counter-Controlled-Loop))
```

Figure 3.2: A goal from PROUST's [Johnson and Soloway, 1987] Knowledge Base

Figure 3.3 is an extract of a plan from PROUST's Knowledge base. This is one of the instances of the Sentinel-Controlled-Input goal. The Template slot describes the form of the Pascal implementation of this plan. It consists of Pascal statements, subgoals and labels written in Lisp notation, rather than ordinary Pascal Syntax. Symbols preceded by question marks are pattern variables which are substituted when the plan is used. $?^*$ is a wildcard pattern. Subgoals are indicated by (SUBGOAL ...) forms in the template which in turn must be implemented using other plans.

With this knowledge, PROUST tries to construct an interpretation for the program to be analysed. Starting with a goal agenda derived from the problem specifications, PROUST selects successive goals for analysis and after performing any applicable reformulation or decomposition in terms of other goals, searches for corresponding implementations for which there is evidence in the code. Hypothesised plans are then evaluated according to

```
(Plan-Definition Sentinel-Process-Read-While
      Constants             (Stop)
      Variables             (New)
      Template              ((SUBGOAL (Input ?New))
                             (WHILE (<> ?New ?Stop)
                                   (BEGIN
                                    ?*
                                    (SUBGOAL (Input ?New))))))
```

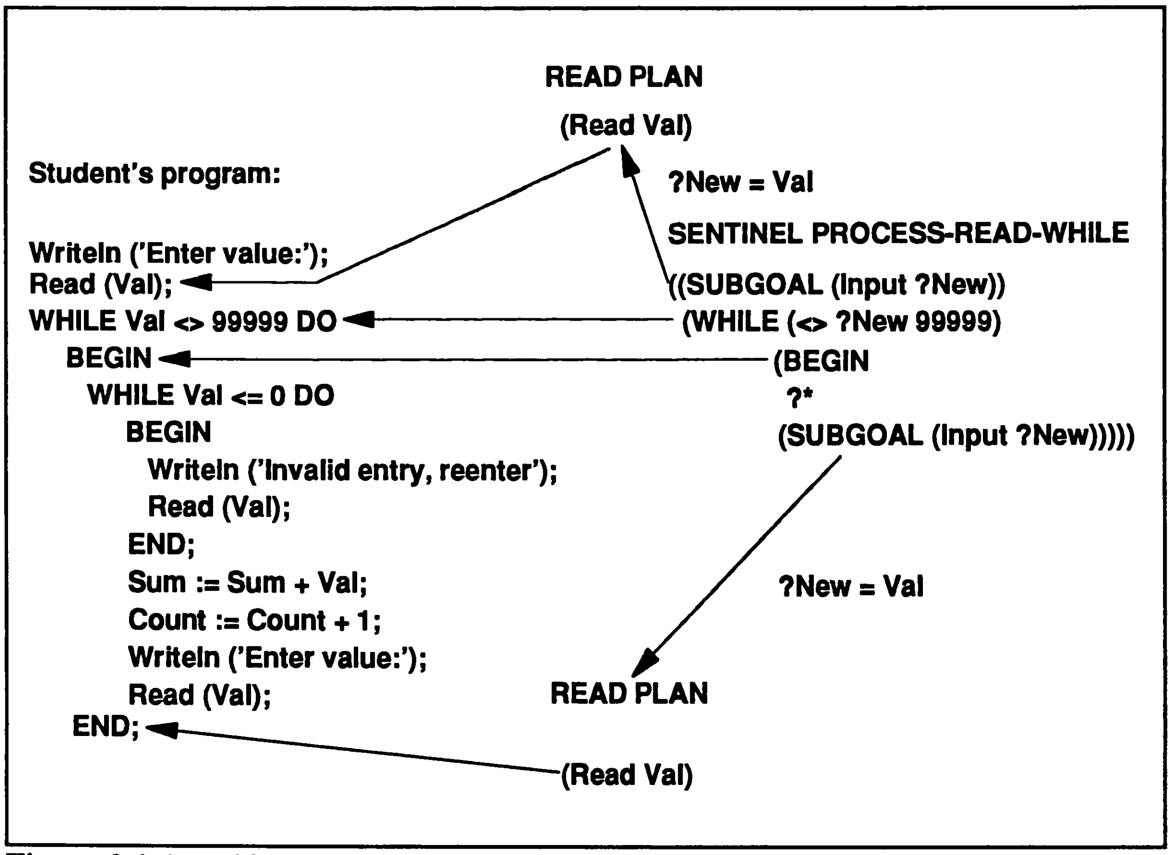
Figure 3.3: A plan from PROUST's [Johnson and Soloway, 1987] Knowledge Base

how well they match the code and how well they fit in the context of the overall interpretation. Transformation and bugs rules are then applied on the code. Competing hypotheses are compared to one another to examine how much code they can explain and how bad the students misconceptions are.

3.1.4 PROUST's Matching Plans

Before any analysis of plans and goals takes place, PROUST parses the student's Pascal program to a parse tree. All subsequent analysis of the student's Pascal program is then performed on the parse tree, rather than on the original program. When PROUST analyses a Pascal program, it selects goals from the problem description, one-by-one. Then, for each and every goal, PROUST substitutes into the goal expression any objects whose values are known and retrieves from its programming knowledge base the plans which could be used to implement the goal. PROUST then hypothesises a plan (initially the first on the list of plans) that the program might use to implement the goal, and then matches this plan against the program. PROUST would then substitute in the selected plan any objects whose values are known. The values for those objects which remain unchanged will be determined during the matching process.

Figure 3.4 shows a successful match, because the plan has been implemented correctly. When PROUST tries to match SUBGOALs of the plan, these are treated as goals. In order to match them against the program, PROUST must go through the same plan selection process as with the main goal. The different plans and subplans for each goal implement a variety of different ways of implementing each goal.


Student's program:

```
Writeln ('Enter value:');
Read (Val);
WHILE Val <> 99999 DO
  BEGIN
   WHILE Val <= 0 DO
     BEGIN
       Writeln ('Invalid entry, reenter');
       Read (Val);
     END;
     Sum := Sum + Val;
     Count := Count + 1;
     Writeln ('Enter value:');
     Read (Val);
  END;
```

```
READ PLAN
(Read Val)
```

```
SENTINEL PROCESS-READ-WHILE
((SUBGOAL (Input ?New))
 (WHILE (<> ?New 99999)
   (BEGIN
    ?*
    (SUBGOAL (Input ?New)))))
```

```
READ PLAN
(Read Val)
```

Figure 3.4: Matching a plan against a student program [Johnson and Soloway, 1987]

After PROUST has converged on one interpretation, it evaluates its reliability by measuring how fully it accounts for elements of the code and the specifications by detecting any flaws. It may discard parts of its analysis and thus warn the student about the completeness of its interpretation. Then it sorts bugs to be reported, trying to group them so that it can point to common underlying misconceptions.

3.1.5 PROUST's Bugs Identification

When there are no match errors, PROUST assumes that there are no bugs in that particular plan. If, however, none of the plans which PROUST selects matches the student code, then PROUST looks for bugs which account for the mismatches in one of the plans. When such a **plan difference** is encountered, that is a difference between the expected plan and the code, PROUST interprets these as bugs.

Plan differences are explained by means of **bug rules**. Each rule has a test part, which examines the plan differences to see whether or not the selected bug rule is applicable and a test part which explains the plan difference. Given below is an example of a bug rule:

```
(Define-Rule WHILE-for-IF
 Statement-Type   IF
 Error-Pattern    (IF . WHILE)
 Bug              (WHILE-for-IF-Conclusion
                         (FoundStmt , *MRet*)
                         (Histinst , *HistoryNode*)))
```

The rule is in slot-and-filler notation. In bug rules, one set of slots constitutes the test part of the rule whereas another set constitutes the action part. In this case, the Statement-Type and Error-Pattern slots are the test part and the Bug slot, the action part. The Statement-Type slot indicates that the plan component that fails to match the program is an IF statement. The Error-Pattern slot indicates that a WHILE statement is found where an IF statement should be expected. The action slot, which is a description of the bug associated with the plan difference, is a WHILE-for-IF confusion. When PROUST presents its findings to the student, it takes each bug description and generates an English Language translation for it, and may also generate data illustrating the presence of bugs. Figure 3.5 shows an example of a program report by PROUST.

```
PROUST: Now reporting MINOR bug in the SETUP part of your program: The
        initialisation at line 7 appears to be unnecessary. The statement in question is:
          RAIN := 0
        (To continue, please press carriage return)
PROUST: Now reporting CRITICAL bug in the CONTROL part of your program:
        You used a while statement at line 19 where you should have used an IF.
        WHILE and IF are not equivalent in this context; using WHILE in place of IF
        results in an infinite loop. The statement in question is:
          WHILE RAIN <> 99999 DO ...
        (To continue, please press carriage return)
```

Figure 3.5: A Bug report generated by PROUST [Johnson and Soloway, 1987]

When PROUST fails to understand a program completely, its ability to recognise bugs deteriorates. In those cases where PROUST analysed partially buggy code, it deleted from its bug descriptions those bugs analyses which were questionable. The remaining descriptions were mainly incorrect.

3.2 Micro-SEARCH: A SHELL FOR BUILDING SYSTEMS TO HELP STUDENTS SOLVE NON-DETERMINISTIC TASKS

Sleeman [1987] claims that students of mathematics and science in general react poorly to tasks that involve the application of non-deterministic algorithms, that is algorithms in which they are required to make arbitrary choices. He reports several reasons that account for this: the student's world views of subjects appear to be small, students expect all tasks to be solvable by well-defined algorithms, their teaching does not prepare them for this kind of algorithms since rarely there is discussion about search, for instance, transformation of algebraic and trigonometric forms and proofs in geometry.

3.2.1 Non-Deterministic Algorithms

Figure 3.6 shows a search tree for transforming a trigonometric expression into two alternative forms and Figure 3.7 shows the paths by which an algebraic expression is transformed into two alternative forms.

$$\begin{tikzcd}
 & \dfrac{\tan x}{1+\tan^2 x} \arrow[dl] \arrow[dr] & & \\
\dfrac{\frac{\sin x}{\cos x}}{1+\frac{\sin^2 x}{\cos^2 x}} \arrow[d] & & \dfrac{\frac{1}{\cot x}}{1+\frac{1}{\cot^2 x}} \arrow[dr] & \\
\dfrac{\sin x . \cos x}{\cos^2 x + \sin^2 x} \arrow[d] & & & \dfrac{\cot x}{\cot^2 x + 1} \\
\dfrac{\sin x . \cos x}{1} \arrow[d] & & & \\
\sin x . \cos x \arrow[d] & & & \\
\frac{1}{2} \sin 2x & & &
\end{tikzcd}$$

Transformation of $\dfrac{\tan x}{(1+\tan^2 x)}$ into both $\frac{1}{2}\sin 2x$ and $\dfrac{\cot x}{(\cot^2 x + 1)}$

Figure 3.6: An example of a trigonometric transformation

The two figures show that there is no one correct transformation to be applied at any stage. The diagrams pictured in the two figures are examples of search trees. The procedure for searching through these trees is non-deterministic because at any one stage, it may not be possible to decide uniquely on a single operator to apply. In such cases, the algorithm makes an arbitrary choice of operator, and only after exploring the path is it

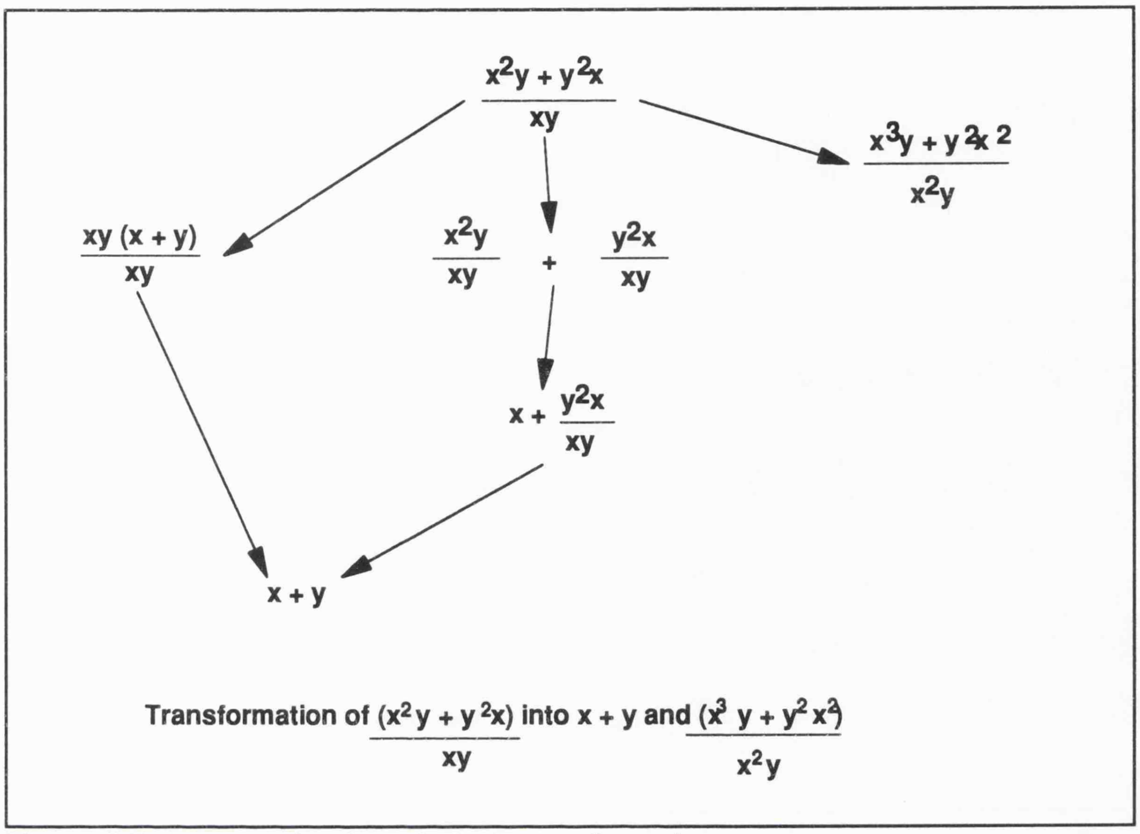

$\frac{x^2y + y^2x}{xy}$
$\frac{xy(x+y)}{xy}$
$\frac{x^2y}{xy} + \frac{y^2x}{xy}$
$\frac{x^3y + y^2x^2}{x^2y}$
$x + \frac{y^2x}{xy}$
$x + y$
Transformation of $\frac{(x^2y + y^2x)}{xy}$ into $x + y$ and $\frac{(x^3y + y^2x^2)}{x^2y}$

Figure 3.7: An example of an algebraic transformation

clear whether the earlier choice was correct. Exploring such trees frequently entails backtracking.

Typical of any transformation, the user is given the initial state, the goal (or goal state) and explicitly or implicitly a set of transformations. Thus, while at the initial node, the person solving the task can know that out of the complete set of transformations only certain transformations are applicable, but would not know which, if any, would lead to the goal. So a strategy to solve such tasks is to apply each of the transformations in turn and after each node in the search tree has been expanded, to check to see if the goal has been achieved. If the goal has not been achieved the tree is expanded further. There are several ways of creating or traversing a possible solution tree. When a node results in a failure, no further expansion is made on the branch of the tree, and the next node is expanded. If there are no more nodes to expand, then the search fails, that is the goal is not attainable.

There are two well-known methods for searching through trees: depth-first and breadth-first. What was described above is known as breadth-first searching. Depth-first searching explores completely one path before considering another.

3.2.2 Teaching Non-Deterministic Algorithms

Sleeman [1987] argues that students are not taught that non-deterministic algorithms are a legitimate search strategy. Furthermore, the teacher frequently states the next transformation to be applied without explaining why this is so, thus giving students little guidance as to how to solve such tasks. He then suggests that students should be explicitly asked, first, to state all the transformations they consider to be appropriate to the task and, second, to systematically explore the complete solution space, by drawing trees like the ones in Figures 3.6 and 3.7. Nevertheless, a major problem is the potential size of the search tree.

3.2.3 Problem-Solving Monitors

Problem-Solving Monitors (PSM) or Coaches (PSC) [Sleeman, 1987] are Intelligent Tutoring Systems that support students' activities when they explore search spaces. These systems ensure that the search space is explored systematically and can provide certain support facilities. In 1975 Sleeman, much influenced by his background as a chemist, implemented the first of a series of such systems to assist students with the interpretation of simple nuclear magnetic resonance spectra. Students were provided with a molecular formula and a spectrum, and were required to produce the molecular structure.

Sleeman's system was able to accept input in three different forms: a solution to the next step of the task in the form of an assertion, for instance, the composition of the next chemical group and the corresponding peak in the spectrum, a request for help that would result in a list of possible next assertions for the user to choose from, and a request for an explanation which would be of great use after an incorrect assertion was made by the student.

When the input by the student is an assertion, the system checks to see if it is syntactically correct and used only resources remaining. For instance, an assertion that tried to use a peak not remaining in the spectrum was rejected. Transformations that passed these tests corresponded to feasible transformations. If the student input was a first request for help, the system listed all the next transformations possible from a particular node of the solution tree. On the second request for help, the system indicated what it calculated to be the best next move or it indicated that the goal was either simply not attainable through the current path or it could not be met in a reasonable number of moves. Solution paths whose lengths were greater than the best solution path by a certain path length were rejected. Finally, if the student input was a request for explanation, which was only made available after an incorrect assertion, the system demonstrated that the goal would not be accessible if the rejected goal was accepted. To do this, the system reported the whole of the tree below the rejected node.

3.2.4 TSEARCH: A Generalised Version of the PSM

TSEARCH [Sleeman, 1982] is a domain independent Poblem Solving Monitor, built to solve tasks that involved non-deterministic searches. It provides a variety of support facilities for its users. TSEARCH can be regarded as a shell for building a certain class of Intelligent Tutoring Systems. Figure 3.8 shows a list of the user commands in TSEARCH along with brief explanation of each command.

COMMAND	Prints a list of commands
PROBLEM	Selects next task
PRINTRULES	Prints a list of all rules in the database
PRINTRULE R	Prints rule R
USERULE N	If possible, apply rule N to current expression; if there are many possible matches ask user which is applicable
HELP	First use on each task gives all the rules which could apply. Subsequent uses for each task give: either the rule TSEARCH would apply or advice that the goal is not reachable in a reasonable number of steps
FORGET	Backtracks one step
REMEMBER	Forgets the FORGET command and reapplies the previously undone transformation
REVIEW	Juxtaposes the user's and TSEARCH's solution paths. (Only available once the task has been solved)
BYE	Allows the user to leave the system

Figure 3.8: User facilities provided by TSEARCH [Sleeman, 1982]

With TSEARCH the student does not type in the transformed expression but the number corresponding to the expression to be applied. Thus TSEARCH would work only in a domain in which the set of operators could be specified in advance. TSEARCH's authors claim this would reduce the number of typing errors made by the students while typing complex expressions in TSEARCH domains. As a result, the system assumes that the student-users know the domain operators and how to apply them. Nevertheless, this elementary form of Human Computer Interaction does not address the difficulty students may have in deciding when the operators should be applied and thus the difficulty in deciding on a solution strategy. TSEARCH's authors developed knowledge bases for Trigonometry, Algebra and Boolean Algebra.

Of the command list in Figure 3.8, TSEARCH's authors highlight the REVIEW command which "places side by side" the student-user's and TSEARCH's solution path for the current task after it has been solved. The expert in such domains has inferred a whole range of heuristics, that is useful rules of thumb which suggest likely rules to be used in given situations. Thus the REVIEW command enables metacognition, that is, it allows users to see gross inefficiencies in their solutions path. Examining the differences between the two traces, it is argued would enable the user to build up a set of good heuristics and refine their own solution process and consequently their own problem-solving strategies.

TSEARCH offers two other important facilities. A command NEW-PROBLEM allows the teacher-user to specify a new task for the student-user. Also, TSEARCH keeps a transformation matrix for each user on each task. The matrix records for each step in the task the transformation chosen by the student and that chosen by the system. Entries on the diagonal of the matric indicated that the student and TSEARCH chose the same transformation and non-diagonal entries indicate that the student selected what the algorithm thought was a non-optimal move. In addition to this matrix, the systems keeps a cumulative matrix that records for every student the transformations that the student applied across all the tasks carried out. In effect, these two transformation matrices can be viewed as student models. Associated with each off-diagonal entry in the matrix can be, TSEARCH's authors suggest, remedial material in the form of either procedural attachments or comments.

The authors of TSEARCH identified four major shortcomings with their implemented system: First, its response time is long especially when a request for HELP is made because it involves expanding a large number of nodes in the solution tree. Second, it can prove to be a very passive learning tool for students because the HELP facility is available at all the stages of the transformation process. Third, the task selection process does not utilise the information in the transformation matrix when selecting tasks for the user. Finally, although the transformation matrices provide valuable information about both the user's and TSEARCH's solution paths the system never points out a "better" solution path than those chosen by the user.

3.2.5 Micro-SEARCH

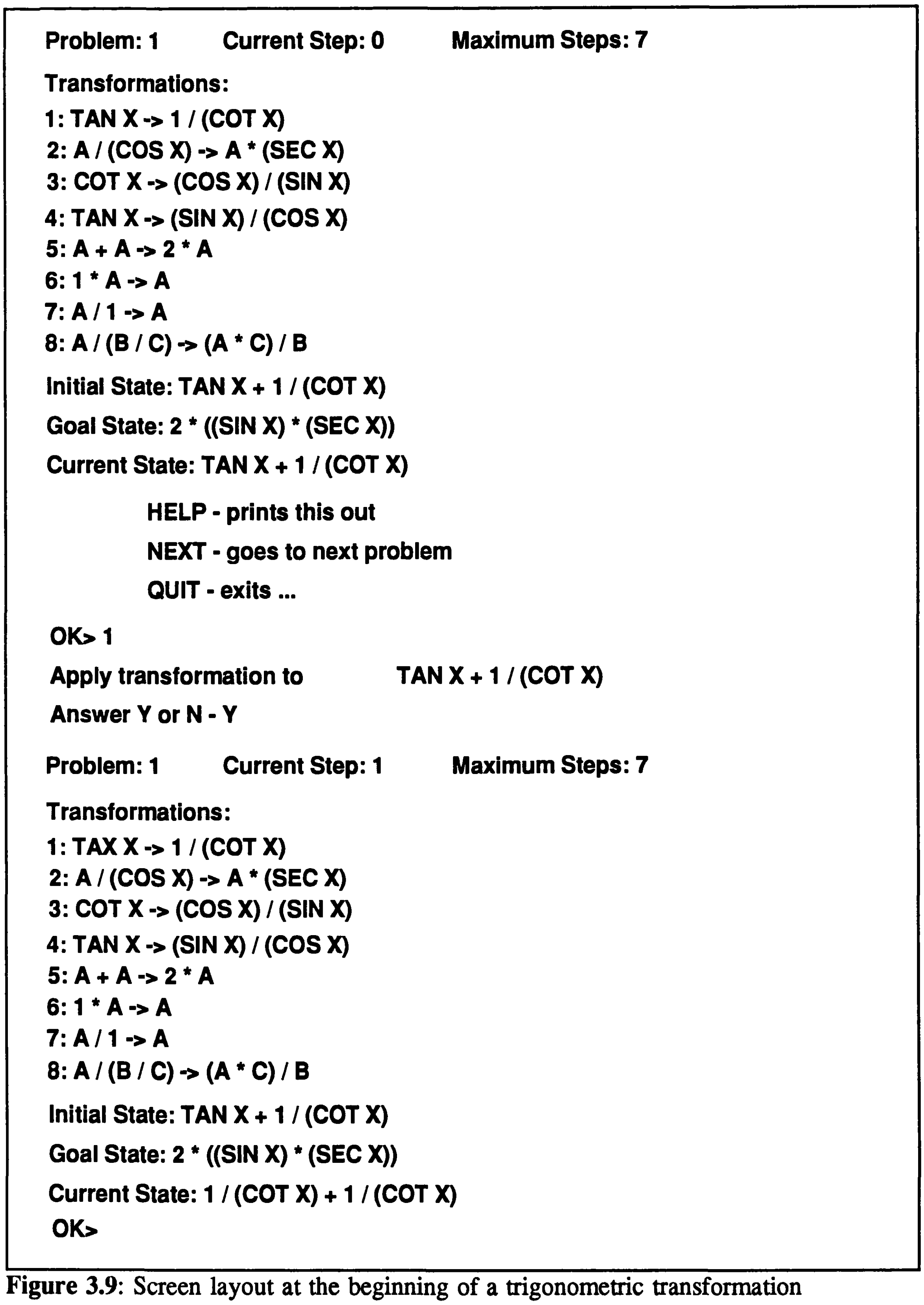
Micro-SEARCH, implemented in Rutgers Lisp, is the IBM PC version of TSEARCH. With the development of Micro-SEARCH, its authors addressed the first of the shortcomings raised in the previous paragraph, that of the response time and thus of processing speed. Furthermore, in implementing micro-SEARCH, several additional changes were made. First, instead of using all the rules for each type of task, only those that are relevant are used. As a result, the rule set was segmented.

Second, the system now has two phases, an off-line phase that creates the complete solution space of correct paths (complete up to some predefined point) in tree structure prior to any interaction taking place, and an on-line phase that accesses the solution space and interacts with the student. This separation is possible because the set of possible transformations can be predefined.

In TSEARCH the complete solution tree was an embedded list. The nodes in micro-SEARCH are in a record structure with three fields: the names of parent nodes which could be more than one, the names of its children nodes and the expression itself. Later micro-SEARCH's authors suggested a fourth field to be included in the record, namely, the list of all applicable transformations. Each node is given a symbolic name and stored on a property list structure. This allows a uniform and rapid access to all the nodes in the solution tree.

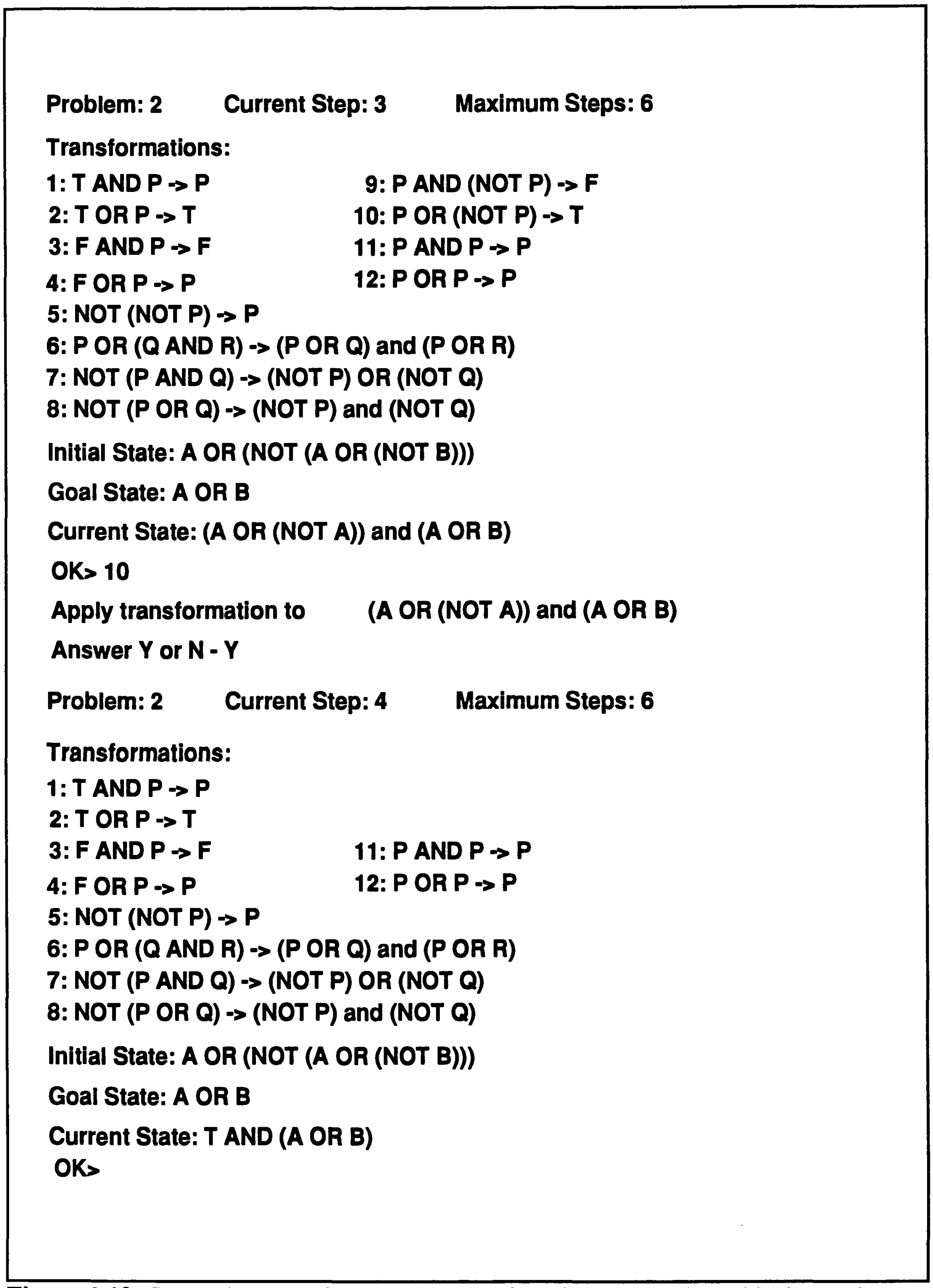
Figure 3.9 shows the complete layout of the screen at the beginning of a trigonometric transformation, with the first transformation made. At each stage the screen displays the list of possible transformations, together with initial and final goals and current states. Figure 3.10 shows an intermediate step in solving a Boolean Algebra task.

Chapter 3 introduced two, of the very few that are available, Knowledge Based Tutoring Systems, namely PROUST and micro-SEARCH, that are used as pilot systems in the study. In particular, it gave a detailed account of their architecture and resulting functionality. In the next Chapter, the thesis presents an evaluation of both systems against Wenger's [1988] model of a didactic operation. The evaluation aims to unveil the requirements for interrelatedness and interconnectedness in Knowledge Based Tutoring Systems in order to be able to support full-scale didactic operations. This would help examine the limitations of existing Knowledge Based Tutoring Systems with respect to these requirements. The functionality of both systems, as presented in this Chapter, will be used in the evaluation exercise in Chapter 4.



```
Problem: 1        Current Step: 0        Maximum Steps: 7
Transformations:
1: TAN X -> 1 / (COT X)
2: A / (COS X) -> A * (SEC X)
3: COT X -> (COS X) / (SIN X)
4: TAN X -> (SIN X) / (COS X)
5: A + A -> 2 * A
6: 1 * A -> A
7: A / 1 -> A
8: A / (B / C) -> (A * C) / B
Initial State: TAN X + 1 / (COT X)
Goal State: 2 * ((SIN X) * (SEC X))
Current State: TAN X + 1 / (COT X)
            HELP - prints this out
            NEXT - goes to next problem
            QUIT - exits ...
OK> 1
Apply transformation to          TAN X + 1 / (COT X)
Answer Y or N - Y
Problem: 1        Current Step: 1        Maximum Steps: 7
Transformations:
1: TAX X -> 1 / (COT X)
2: A / (COS X) -> A * (SEC X)
3: COT X -> (COS X) / (SIN X)
4: TAN X -> (SIN X) / (COS X)
5: A + A -> 2 * A
6: 1 * A -> A
7: A / 1 -> A
8: A / (B / C) -> (A * C) / B
Initial State: TAN X + 1 / (COT X)
Goal State: 2 * ((SIN X) * (SEC X))
Current State: 1 / (COT X) + 1 / (COT X)
OK>
```

Figure 3.9: Screen layout at the beginning of a trigonometric transformation



```
Problem: 2        Current Step: 3        Maximum Steps: 6
Transformations:
1: T AND P -> P                    9: P AND (NOT P) -> F
2: T OR P -> T                    10: P OR (NOT P) -> T
3: F AND P -> F                   11: P AND P -> P
4: F OR P -> P                    12: P OR P -> P
5: NOT (NOT P) -> P
6: P OR (Q AND R) -> (P OR Q) and (P OR R)
7: NOT (P AND Q) -> (NOT P) OR (NOT Q)
8: NOT (P OR Q) -> (NOT P) and (NOT Q)
Initial State: A OR (NOT (A OR (NOT B)))
Goal State: A OR B
Current State: (A OR (NOT A)) and (A OR B)
OK> 10
Apply transformation to       (A OR (NOT A)) and (A OR B)
Answer Y or N - Y
Problem: 2        Current Step: 4        Maximum Steps: 6
Transformations:
1: T AND P -> P
2: T OR P -> T
3: F AND P -> F                   11: P AND P -> P
4: F OR P -> P                    12: P OR P -> P
5: NOT (NOT P) -> P
6: P OR (Q AND R) -> (P OR Q) and (P OR R)
7: NOT (P AND Q) -> (NOT P) OR (NOT Q)
8: NOT (P OR Q) -> (NOT P) and (NOT Q)
Initial State: A OR (NOT (A OR (NOT B)))
Goal State: A OR B
Current State: T AND (A OR B)
OK>
```

Figure 3.10: Screen layout at intermediate step in solving a Boolean Algebra task

CHAPTER 4: EVALUATION OF DIDACTIC OPERATIONS IN KNOWLEDGE BASED TUTORING SYSTEMS: THE CASE OF PROUST AND micro-SEARCH

In this Chapter, the thesis unveils the requirements for interrelatedness and interconnectedness in existing Knowledge Based Tutoring Systems with respect to the full-scale didactic operation, as discussed in Chapter 2, and discusses their limitations with respect to these requirements. The thesis achieves these aims through an evaluation of PROUST and micro-SEARCH against Wenger's [1988] model of a didactic operation by addressing the question, *What should the relationship between a system's behaviour and its architecture be with respect to the didactic operation?*

In the first part, the chapter discusses the evaluation of Intelligent Tutoring Systems followed by the evaluation of PROUST and micro-SEARCH against Wenger's [1988] model. The Chapter then examines the requirements for interrelatedness and interconnectedness in existing Knowledge Based Tutoring Systems with respect to a full-scale didactic operation followed by an investigation of their limitations with respect to these requirements.

4.1 EVALUATION OF INTELLIGENT TUTORING SYSTEMS

It is generally accepted by most Intelligent Tutoring Systems researchers that evaluation of any sort is a neglected practice [Nwana, 1990b]. Nwana [1990b] takes this further by arguing that this largely applies to the Artificial Intelligence domain as a whole. The pay off of evaluation would be in helping to answer two questions that are central to cognitive science, artificial intelligence and education [Littman and Soloway, 1988]:

[1]. ***What is the educational impact of an Intelligent Tutoring System on students?***

[2]. ***What is the relationship between the architecture of an Intelligent Tutoring System and its behaviour?***

Addressing the two evaluation questions leads to a different perspective on evaluation from that of traditional educational evaluation. Traditional educational evaluation consists of two main categories, formative and summative evaluation [Clegg et al, 1988]. Designers of Educational Technology use formative evaluation to define and refine their goals and methods during the design process. They use summative evaluation to determine whether a finished educational product is effective after it has been built.

Because building Intelligent Tutoring Systems is still somewhat of an art, and because there are few Intelligent Tutoring Systems that can be called finished [Littman and Soloway, 1988], designers of Intelligent Tutoring Systems are currently more concerned with usefully guiding the development of their systems than with determining whether they, or can be, effective educational end products. At least for the time being, as Littman and Soloway [1988] claim, the idea of a formative evaluation seems more appropriate for Intelligent Tutoring Systems designers than does the idea of summative evaluation. Hence the two evaluation questions are mainly focused on the development of Intelligent Tutoring Systems rather than on determining whether they are effective educational end products.

Unfortunately, there is no standard set of evaluation methods for addressing either of the two evaluation questions. Nevertheless, Littman and Soloway [1988], in evaluating PROUST, have defined two classes of evaluation guidelines that are useful for this purpose. The first class of evaluation guidelines, which addresses question 1 and is called ***external evaluation***, assesses an effect external to the Intelligent Tutoring System (i.e. the student's learning), by way of examining how an Intelligent Tutoring System affects students and changes their knowledge and problem solving skills.

The second class of evaluation guidelines, which addresses question 2 and is called ***internal evaluation***, assesses an effect internal to the Intelligent Tutoring System (i.e. the inner workings of an Intelligent Tutoring System), by constructing a picture of the architecture of an Intelligent Tutoring System and its relationship to its behaviour. The answers the two resulting classes of evaluations provide to these questions highlight those aspects of a tutoring system that have particular effects on its behaviour and how the design and implementation of the tutoring system lead to its behaviour.

4.1.1 External Evaluation: The Cognitive Perspective

Recent progress in Cognitive Science and Artificial Intelligence has provided the field of Intelligent Tutoring Systems with a tool, process-based student models, for representing student's knowledge and problem solving-skills [Littman and Soloway, 1988]. With early tutoring systems these tools were not available. Nevertheless, the reasonable and pragmatic assumption was made by early tutoring systems developers that the students' answers to test questions reflected their mental processes.

As a result, the goal in evaluating early tutoring systems has been primarily to determine whether students can correctly respond to test questions. With the advent of process-based student models, however, the goal of evaluating Intelligent Tutoring Systems is to determine how well the Intelligent Tutoring System teaches students the knowledge and skills that support the cognitive processes required for solving problems in the content domain of the Intelligent Tutoring System. Thus the cognitive perspective on external evaluation was made possible by the advent of student modelling in Intelligent Tutoring Systems [Littman and Soloway, 1988].

As an Intelligent Tutoring System interacts with a student, it builds up a model of the student, that is an understanding of the student's knowledge and skills, which it uses to interpret the student's behaviour and to guide its own actions. This is achieved via methods for reasoning about the students' problem-solving in the domain of instruction. Many student modelling techniques have been proposed but all these techniques can be grouped under two major cognitive categories: those methods that are based on process models of problem solving and those that are not.

Student modelling techniques based on process models solve problems in a supposedly humanlike way. For example, the student modelling component in the Lisp Tutor is based on a process model of how students write simple Lisp programs and is embodied in their GRAPES simulator [Anderson and Reiser, 1985]. The Lisp Tutor uses the GRAPES simulator to simulate the problem solving of novice Lisp programmers when they write simple Lisp programs. The student model is thus represented in terms of what the GRAPES process model did to solve the problem.

Student models that are not based on comprehensive process modelling do not solve problems as humans do. For example, in WUSOR, the tutor for the discovery game WUMPUS [Goldstein, 1982], the student model consist of the skills that have been checked off in WUSOR's representation of skills. WUSOR does not try to play

WUMPUS as a student would in order to build its student model and thus it does not use process models.

Whether or not student models actually have process models that simulate students' behaviour, they can be used to assess how well the Intelligent Tutoring System teaches students skills and knowledge for solving problems that are like the problems encountered during learning. Student modelling techniques can also guide the construction of new problems for testing the student. Because these techniques require explicit representations of problem solving knowledge and skills, and possibly the actual process of problem solving, they can be used to predict how well the student will perform on the new problems and thus which problems should lead to effective problem solving and which to ineffective problem solving.

Because student modelling techniques capture how students solve problems and not merely that they can solve problems, they can be used to identify problems that the student should be able to solve. Student modelling techniques that are not based on process models can still be used to predict some of the knowledge and skills the student will use to solve problems. Process-based techniques can be used to predict the actual process the student will go through to solve problems. Therefore, the evaluation of early tutoring systems which focused on correct and incorrect answers is different from the evaluation of Intelligent Tutoring Systems which assess the reasons that students give correct and incorrect answers.

Consequently, the focus of the external evaluation of an Intelligent Tutoring System is the degree of completeness or comprehensiveness of the process model underlying the

Intelligent Tutoring System. In the external evaluation of the Intelligent Tutoring System the criterion is not how many of the students' answers are correct but the underlying fine-grained skills that have been learned.

4.1.2 Internal Evaluation: The Architecture Perspective

The goal of internal evaluation is to provide a clear picture of the architecture and its relationship to behaviour. To clarify this relationship it is necessary to characterise the Intelligent Tutoring System in terms of answers to three key questions [Littman and Soloway, 1988]:

[1]. ***What does the Intelligent Tutoring System know?*** The question is addressed by an analysis of what the Intelligent Tutoring System can possibly do based on what it knows.

[2]. ***How does the Intelligent Tutoring System do what it does?*** This requires analysing the Intelligent Tutoring System to determine how the algorithms use available knowledge to produce the observed behaviour of the Intelligent Tutoring System.

[3]. ***What should the Intelligent Tutoring System do?*** This question is answered by clarifying the areas of the tutoring domain that the Intelligent Tutoring System is responsible for teaching.

According to Littman and Soloway [1988], knowledge engineering can help answer all three questions by performing Knowledge Level Analysis, Program Process Analysis and Tutorial Domain Analysis.

Knowledge Level Analysis attempts to characterise the knowledge in the Intelligent Tutoring System and thus answers the first question: *What does the Intelligent Tutoring System know*? It provides useful information about whether the program knows enough to perform the intended tasks. It is concerned not with how the program accomplishes the tasks but with what the program can conceivably do and with whether the program has the competence to perform certain tasks. In other words, it focuses on whether the program has enough of the right kinds of knowledge to meet the requirements that were set for it.

Program Process Analysis answers the second question: *How does the Intelligent Tutoring System do what it does*? It consists of evaluating whether the program does what it does in the right way. In contrast to Knowledge Level Analysis, which asks whether the program is able to perform certain input-output tasks, Program Process Analysis looks just at how a program uses its knowledge in the process of going from input to output. In other words, it focuses on the control structure of processing the knowledge.

Tutorial Domain Analysis answers the third question, *What should the Intelligent Tutoring System do*? by highlighting any lack of tutorial abilities on the domain to be tutored.

The underlying multi-disciplinary nature of the Intelligent Tutoring System cannot lend itself to a single evaluation philosophy [Littman and Soloway, 1988]. Intelligent Tutoring Systems evaluation calls for an evaluation approach borrowed partly from educational technology, partly from Computer Science, partly from Artificial Intelligence and partly from cognitive science. To unveil the requirements for interrelatedness and interconnectedness in existing Knowledge Based Tutoring Systems with respect to the full-scale didactic operation, this thesis is concerned only with the second question which seeks to examine the relationship between a system's architecture and its behaviour. The evaluation of the two systems against Wenger's model of a didactic operation proceeds with a Tutorial Domain Analysis.

4.2 THE INTERNAL EVALUATION OF PROUST AND micro-SEARCH

Wenger's [1988] model of a didactic operation provided the context for the Tutorial Domain Analysis. Nevertheless, Nwana's [1990b] Intelligent Tutoring Systems development principles, Self's [1985] subject-independent model of intelligent behaviour, and O'Shea's et al [1984] thirteen pillars of Intelligent Tutoring Systems Design also contributed towards completing this context. The result is a set of pertinent (to the model of a didactic operation) questions that seek to examine the relationship between the system architecture and its behaviour. This can then help uncover and understand what is required of a Knowledge Based Tutoring System in order to support a full-scale didactic operation, as described in Chapter 2 of the thesis.

The evaluation strategy used involved setting up a laboratory experiment in which a group of students used both systems for a period of one week, at the end of which they were interviewed with the assistance of a questionnaire handed out to them at the beginning of the experiment. The evaluation strategy then involved answering for both systems all the questions about the model of a didactic operation with respect to their architecture and functionality as explained in Chapter 3 of the thesis and results obtained from the laboratory experiment.

4.2.1 Evaluation against real users

For the purpose of evaluating PROUST and micro-SEARCH, the following Laboratory Experiment has been set up [Alessi and Trollip, 1985]:

1. Run a controlled experiment in a classroom in order to determine the relative effectiveness of teaching Pascal using PROUST and Trigonometry, Algebra and Boolean Algebra using micro-SEARCH. Both systems are used as classroom aids.

2. Determine the effectiveness of both Intelligent Tutoring Systems on individuals, probing students' factual and procedural understanding of what the two systems are trying to teach by means of individual clinical interviews after the end of the experiment.

3. Use PROUST and micro-SEARCH as a test-bed for asking how a more individualised set of tasks, discussion of issues, and control over the availability of on-line HELP and ADVICE would affect student performance.

4. Probe the extent to which students have acquired a notion of the various techniques for solving a certain class of tasks.

A group of 10 M.Sc. ADMIS students at the LSE used both systems for a period of one week in August 1991. Both systems were installed on the LSE's Ethernet network and the students gained access to them from an LSE public computer room which was exclusively reserved for this purpose. I supervised most of the sessions the students had with the two systems, in order to give them a sense of direction and also to provide them with some help if they got confused or stuck, whilst making every effort not to bias the experiment in any way. All students who used the two systems had some prior knowledge of programming and mathematical transformations. A larger evaluation with more students would have been preferable but there were practical difficulties with issues of time and resources.

Although I handed all the participants a questionnaire (see Appendix B) which I expected them to fill out after they mastered the use of the two systems by going through a set of prescribed exercises, I encouraged them to voice their opinions during the course of interaction. The questionnaire was intended to unravel issues relating to the two systems tutoring behaviour. At the end of the questionnaire, I invited the students to write their own comments, regarding any aspect of the two systems. I then discussed individually with each of the 10 students the context of their answers to the questions.

Although the experiment was not set to check the student-user's knowledge and skills before and after a successful use of the systems, some students reported acquiring little additional knowledge from the two systems, while others reported mainly to have slightly improved on their current skills. There were also the odd cases who reported zero gains from either of the two systems. The results are by no means conclusive: the students who reported gains either in their current knowledge or skills, argued that they could have improved a lot more if they were taught by a human teacher for a period of one week the art Pascal programming and mathematical transformations.

Some initial fears that the two systems would be impossible for students to use proved to be unfounded. Apart from the few occasions when they needed help, especially during the initial stages, they were observed to be getting ‘carried away’ with the two systems.

The students found the two systems lacking in any real motivation. This was partly because of the inability of the two systems to solve, during the course of interaction, a problem which they set for the student and also explain the solution and partly because of their inability to allow the user-learners to "dream up" their own problem and watch over students while solving the problem. One of the aspects which was criticised heavily in both systems, was the lack of proper explanations of micro-SEARCH’s step-by-step solution to a problem and PROUST’s incomprehensible results during the actual process of Pascal code analysis and its inaccuracy of bug diagnosis in certain cases. All the students were very disappointed with the systems’ inability to detect errors and provide them with some guidance and tutoring about some knowledge and skills, they were having problems with. The students were largely frustrated with the systems’ "canned" text explanations and black box diagnostics! The most appalling feature to the students was their system interface which they found too elementary, inflexible and lacking in many respects.

4.2.2 Evaluation against Wenger’s model of a didactic operation

PROUST’s and micro-SEARCH’s didactic plans of action

Q1. ***What didactic plan of action do the systems follow? Didactic episodes of actions or goals, or one based on diagnostic expectations? Are their didactic plans of action prefabricated during system design or are such decisions made by the system during the course of interaction?***

Neither system is able to perform student diagnosis, thus at its best, both systems' plan of action would be one of a didactic episode where either a goal has to be met or an action be performed. PROUST's plan of action is a didactic episode of a single action, generated by the problem description submitted by the instructional designer to PROUST. In this "episode", the student submits his Pascal code for error analysis. PROUST has different plans in its knowledge base for implementing the goals in the problem description whose descriptions also appear in PROUST's knowledge base. PROUST first parses the student's Pascal code to a parse tree on which all the subsequent analysis is then performed. Then it selects a goal from the problem description, retrieves all the plans for each goal from its knowledge base and makes a comparison of these plans against the parse tree to determine which plan fits best the student's Pascal program. Once PROUST selects the best match, any differences between the plan and the student code are interpreted as bugs. The student-user may choose to watch PROUST performing the analysis of his code, in which case PROUST displays the results of this analysis, which the real students who used the system found hard to follow. In these didactic episodes, PROUST takes the student through a learning path expecting the student to develop enough Pascal knowledge and expertise so that he advances from being a novice beginner to an experienced beginner.

Micro-SEARCH capitalises on the assumption that all possible mathematical transformations can be accurately predefined, thus prior to any interaction taking place with the student, the system creates the complete solution space of correct paths as a tree structure. During interaction the student traverses this tree by applying a series of transformations that take him from node to node towards the goal node which has been set by the system. Thus micro-SEARCH's plan of action is that of a series of didactic "episodes" during each of which the user has to achieve a goal. Each and every node the user visits in this tree is the goal that defines the context of a didactic episode. Solution paths whose lengths are greater than the best solution path by a certain path length are rejected by micro-SEARCH.

The user is given the initial state, the current state, the goal state and a set of transformations. While at the initial node, the person solving the task is assumed to know that out of the complete set of transformations only certain transformations are applicable, but would not know which, if any, would lead to the goal. So a strategy to solve such tasks is to apply each of the transformations in turn and after each node in the search tree has been expanded, to check to see if the goal has been achieved. If the goal has not been achieved the tree is expanded further, by the user selecting the next transformation to apply. The system intervenes to warn the user if a transformation option he has selected would put him on a longer path to the solution but it would not prevent him from applying it. The system performs the transformation for the user, and modifies the current state of transformation. These interventions and any associated advice are predefined by the instructional designer. At all times, the student has access to a pictorial representation of the tree which encompasses the complete set of solutions. Through these "didactic" episodes, the system assumes that the student-user has, or will acquire knowledge, of the necessary domain operators and how to apply them. Nevertheless, this elementary form of Human Computer Interaction does not address the difficulty students may have in deciding when the operators should be applied and thus the difficulty in deciding on a solution strategy.

Although neither system has an explicit representation of a plan of action to follow, nevertheless, both systems entail some form of a primitive didactic plan. However, in both systems these are exclusively used for monitoring problem solving and not for guiding the unfolding of any pedagogical activity. Both systems have an implicit curriculum of ordered topics and associated assignments that the interaction is expected to cover. Both systems' primitive plans have been prefabricated during system design. Neither system is capable of "making" any decisions about the course of their plan of action during the course of interaction. In principle, both systems' decision tree is defined by their knowledge representation and determines the action in the case of PROUST or the next goal in the case of micro-SEARCH. To go to another branch would involve in the case of micro-SEARCH to undo the last transformation (s) and to change the initial code in the case of PROUST. There is no branching in neither of the two systems other than what is precisely defined by the tree. This and the lack of student diagnosis do not allow for an opportunistic didactic plan.

Q2. ***Do the systems enable the student to communicate his plans (i.e. intentions) prior to executing them?***

A positive answer to this question would expose the many situations where students attempt to perform a wrong operation at a particular stage, in other words it would reveal those concepts which the student knows how to perform but does not understand when or where to perform them. This would make diagnosis and remediation easier. PROUST does not facilitate this at all. The student-user must have in hand a solution to the programming assignment given by the system prior to entering the system. Although PROUST has knowledge about how to write Pascal programs, it does not provide the user with an on-line editor with which to create or modify his Pascal code. Instead the student-user has to use a conventional Pascal editor to do so. Thus the student executes any plans which he may have about how to solve his programming assignment before entering PROUST. Therefore, the answer to this question for PROUST is no. Micro-SEARCH, on the other hand, allows the user to communicate his intentions, that is which transformation he is thinking of applying, prior to the system executing it.

PROUST's and micro-SEARCH's pedagogical contexts

Q3. ***What kind of pedagogical strategy do the systems follow to exercise control over the shape of the tutorial sequence? Opportunistic, plan-based or a mixed strategy?***

The lack of student diagnosis leading to a student model does not allow either system to take advantage of any opportunities that may arise in the context of the problem solving episodes they engage the user in. Consequently, neither system can follow any opportunistic strategy for tutoring. Both systems are strictly plan-based as defined by their decision trees rather than by a didactic model. The lack of a proper pedagogical plan of action and consequently pedagogical goals restrict both systems in taking advantage of any plan-based opportunities that may also arise while the user is engaged in a problem solving episodes. The student-user's activities or statements would have provided a focus for diagnosis and for the content of tutorial interventions. Therefore, neither system follows any form of pedagogical strategy according to the model of didactic operations. Their strategy is purely a problem solving strategy.

In the case of micro-SEARCH, opportunities arises when the student selects a transformation to apply on the current state of a mathematical expression during the course of which the system could take the opportunity to provide explanation or remedial action, as a result of local monitoring of the student, or generate a new didactic episode.

Instead the only help the system provides are "canned" pieces of text, prefabricated by the author of the problem during problem design. These explanations refer only to the current state of the transformation and not to any past or future ones since the system lacks an explicit plan of action that would enable the system to trace forward a user-selected option or to backtrack. In the case of PROUST, opportunities arise when the student submits his Pascal code to PROUST for bug identification. After PROUST analyses the student's Pascal code, it generates a report in which it could have provided the student with some useful explanation as to how some of his bugs arose in the first place, and in some cases how some bugs can be corrected. Although PROUST allows the user to watch the system perform the actual process of code analysis where PROUST's results are displayed on to the screen for the user to read these results are computer jargon with little pedagogical value. Both the students who used the system and myself found them hard to read and follow.

Both systems have been programmed to intervene when the user-learner input is not the expected one. Although this intervention may exhibit some of the basic characteristics of an opportunistic strategy neither system has an explicit representation of such a strategy nor do they have any control over how the organisation of their instructional sessions communicate the subject matter to the student. The student-user's learning is entirely left up to him. Although as argued in the previous section, both systems include some very primitive form of a didactic plan of action, this is for guiding the problem solving process and not for planning any global or local learning events as would have been the case with a plan-based context.

Q4. ***Do the systems maintain control over the whole tutoring endeavour or do they***

leave part or all of the initiative to the student?

Both systems maintain control of the interaction, with very little choice left to the student. Nevertheless, the initiative for learning lies very much with the student-user. In that respect both systems are very passive teaching tools. If any learning takes place then this is the result of the user-learner's activities and not of an active teaching aid. The student has the choice of subject area in micro-SEARCH and of the programming assignment in PROUST. Furthermore, in micro-SEARCH he has the choice of a transformation from a list of given transformations. The user is not allowed to invent his own transformation but only to select one from the given list. The tree representation of the complete solution space which is placed at the disposal of the student is solely for visual inspection and does not provide any explanation about it.

In PROUST, there are not really any options: the user submits his Pascal code, PROUST analyses it and produces a bug report, and the student can browse through the report, without being able to ask any questions about it. The student-user traverses both systems through the use of very restrictive push-button menu-based interfaces. If the system was a full-scale knowledge communication system, had a set of pedagogical goals to attain, had a plan of how to attain those goals then the system would have been an active teaching aid. It might then be in control of the whole tutoring endeavour most of the time and at the same time it would allow more flexibility to the user in order to be able to fulfil its diagnostic tasks and apply his plan of action. As both systems currently stand they can offer very little choice to user because of the lack of these two components.

Q5. ***Do the systems motivate and support a flexible style of tutoring?***

Neither system can be classified as a full scale intelligent tutoring system because both systems lack a tutoring strategy centred around a student model, which they also lack. Neither system is able to support various idiosyncratic user solutions, or various tutoring strategies, or various idiosyncratic inputs, nor can they turn a problem solving episode into private tutoring. If the systems diagnosed a model of the student's knowledge from which the most productive teaching strategy for the student could be determined and the systems incorporated some computational teaching strategies and pedagogical goals to attain, then the systems could adapt their instruction by selecting from their pool of teaching strategies the "best" one to ease and accelerate the communication of ideas from the domain expert model of the system to the user. In addition such a system could use the knowledge of the student in the student model (about local goals) to individualise even further the instruction and decide which goal should be attained next.

Having this flexibility in tutoring may not be important for good students who may only want to use a system as a problem solving tool or for undertaking reinforcement exercises, but it may be of great importance to weak students because they can use the system in their own time to master their knowledge and/or skills, their mistakes can be private, they can ask clarifying questions, etc. The students who used the two systems reported that after a little while they got very bored with using the systems. Neither of the two systems makes any attempt at, nor is able to, motivate the student-users, for instance, praise and reward the user-learner, if he gets an answer right. The worse system of two in this respect seems to be micro-SEARCH which supports a very inflexible style of tutoring. Some students who used the systems argued that it is more like an exercise book with a problem solving ability while others suggested that the system should be used only as a revision tool by students who want to practice their mathematical transformations skills.

PROUST, on the other hand, does enable the user to feel some motivation as his actions are automatically reported upon. An editor would prove to be useful in this respect because students are currently discouraged by having to exit PROUST and go their Pascal editor and "fix" their program and come back and resubmit, and start all over again. All the students who used the system reported in one way or another that the systems' interfaces are also a major source of discouragement and boredom.

Q6. ***Do the systems provide an environment in which the interaction between it and the student is close to reality?***

Both systems exhibit a number of shortcomings with respect to how realistic their instructional environment is. Although PROUST takes as input the Pascal program which the student authored, and delivers as output a bug report which contains the results of PROUST's diagnosis on the student's code as specific syntactic and semantic bugs in the student's code, it does not enable the student to modify his code while in PROUST. The system is unable to provide local monitoring of student activities. The student has to exit PROUST, go back to his Pascal editor, do the changes and return back to PROUST to resubmit his code and reiterate through the process. PROUST entails knowledge about Pascal syntax errors, how to write programs and planning descriptions of the assignments it sets to the student which it can use to reconstruct several solutions, all of which it could use to help the user to correct his code and provide tutoring where necessary. In addition to this shortcoming, the student cannot pose any questions to PROUST regarding the contexts of the generated report. This is partly because of the lack of an enabling interface, like a natural language interface, with which to pose questions regarding some aspects of the systems' functionality and output, and partly because the two systems are basically problem solving tools that do not set out to allow the users to interrogate their knowledge representations.

All the students who used the two systems reported that the menu-based system interfaces of both systems were very restrictive and suggested that they should be further improved. In addition, both systems lack proper explanation facilities. Although there is a help function key in both systems, the only purpose this serves is to display a list of all the available function keys in the systems and their purpose.

Micro-SEARCH, on the other hand, lacks both in input and output terms. Although the user interacts with the system in order to improve his transformation skills, the system does not allow the user to type in himself the transformation which he wants to apply to a current mathematical expression. Instead he is given a list of transformation options to choose from. When he chooses a transformation the system will perform the transformation operation for him and display the result. In other words the system does not enable the student to solve problems as they would on paper. Furthermore, micro-SEARCH does not record anywhere the transformation steps which the student went through in order to arrive at the current result or the goal state. When students use pencil and paper, they write in the intermediate steps until they arrive at the correct answer. The student has to resort to the tree diagram to do so where he is left unaided.

In addition, the system lacks in terms of graphics abilities in order to display fractions and powers as they would appear naturally on a piece of paper. Instead the systems developers used some conventions which complicates the expressions. This confused quite a number of students who used the system. Furthermore, the system developers reversed the polarity of the up and down, left and right, pageup and pagedown keyboard function keys while browsing through the tree diagram. Left arrow takes you right and right arrow takes you left, up arrow takes you down and down arrow takes you up. If you press pageup then you move down a page and if you pagedown then you move up a page. This frustrated the students who used the system because every time they were browsing the tree diagram they also had to spend some effort thinking about how to use the interface. Some students argued that interaction with the two systems is very much a one-way interaction mainly because student-users are not allowed to ask any questions, let alone ask ad hoc questions.

Q7. ***Do the systems teach prerequisite skills?***

Neither system teaches prerequisite skills and/or concepts that are necessary for a student to have prior to interacting with the system nor during a session where the student-users consistently exhibited a lack of certain skills. Tutoring of these skills is done by exposition, examples, and testing [Nwana, 1990b]. Both systems assume that any potential user-learner to have the necessary skills in the subject area before they start using the systems. Thus they progress through the tutoring of students with different aptitudes similarly. Although is quite clear what the target user group and the necessary prerequisite skills are in the case of PROUST, with micro-SEARCH it was very difficult to determine precisely either the target user group or the necessary prerequisite skills. In addition to this, a lot of the students felt that both systems assume intelligent user-learners, who may not be in need of tutoring aids after all. Although both systems have as an optional user facility, a tutorial introduction in the form of text pages dumped on the screen, the students found these to be of little use.

The prerequisite skills problem could have been eradicated had the system included an initial model of student user requirements for the student to match his knowledge against. In addition this could serve as a testing mechanism for those students who complete their interaction with the system and wish to know their level of attainment regarding the necessary skills for achieving success with the problems the systems set to the students. However, this requires student modelling abilities which neither systems possess.

Q8. ***Do the systems monitor the student step by step?***

Although PROUST has knowledge about how to write a Pascal program and is able to diagnose both syntactic and semantic bugs in a Pascal code, it does not monitor the student during the problem solving process step by step because it does not enable the student to write his Pascal code on-line. Instead the user has to write his code off-line by using a conventional Pascal editor, which he then submits to PROUST to analyse. The reason for this is the lack of on-line student diagnosis that would enable PROUST to perform local monitoring of the student and also keep in track with the pedagogical goals the didactic plan of multiple actions (not of a single action) must help the student attain.

micro-SEARCH monitors the student during the problem solving process step by step by checking to ensure that his choice of transformation is the right one for the situation in hand, or an acceptable one (i.e. it would still lead to a solution although it would take longer), or an incorrect one in which case it prompts the user to select another transformation. One disadvantage of micro-SEARCH is that it does not keep track of the user-selected transformation so far in the process for the use of the student, and it also does not prevent the user from selecting the wrong transformation twice. Again, micro-SEARCH suffers from the lack of student knowledge and a proper didactic plan of action. The system can only present a range of possible answers but cannot explain them or relate them to any local goals that it indirectly tries to achieve.

To be able to perform step by step monitoring, it would require a set of local teaching goals for every step and the creation of a local student model to both record the results of monitoring and also to determine the next step to perform. It would also require a set of appropriate teaching strategies that would have been suitable for tutoring with these goals.

Q9. ***Do the systems test the student's understanding?***

Testing the student's understanding involves testing for all sort of goals, for example, testing for knowledge of how to use a concept, testing for knowledge of when to use a concept, testing to determine whether the student can handle problems of a particular difficulty level, etc. Neither of the two systems tests a student-user's understanding or his programming or transformation skills overall or subskills, nor his ability to apply these skills. With micro-SEARCH this kind of testing is far from possible. There are many reasons for this. First, the system has no explicit representation of a syllabus to follow other than the sets of randomly prefabricated problems which it uses. Second, it does not have a student diagnostic module that constructs a student model and thus cannot use this to infer any missing conceptions by comparing this to the syllabus or misconceptions by matching these against a library of bugs about the syllabus. Thirdly, its diagnostic techniques are very primitive. They entail only deviations from a problem solving path that leads to the solution.

PROUST, on the other hand, has most of the necessary ingredients to perform this form of testing. It has a knowledge base on Pascal and how to write Pascal programs and is able to diagnose misconceptions in the user-learner's code. However, PROUST's diagnosis of misconceptions is not based on a bug library. PROUST's diagnosed misconceptions are deviations from the correct Pascal program design. Therefore, PROUST is not able to report about their origin or how these can be corrected nor does it attempt to relate these to any missing conceptions because its diagnosis is not proper student diagnosis. Thus it suffers, but only partly, from the same shortcomings as micro-SEARCH.

Q10. ***Do the systems provide remedy in a problem-solving context?***

Providing a remedy in a problem solving context takes many forms, for example, diagnosing an error, instructing the student on what to do, etc. and a system should come to their rescue when they get stuck, confused, etc. For remedy to work both in local or global terms, (i.e. a problem solving episode or in more general terms), it assumes a bugs library which neither systems possesses, a student model in the context of which to place the bug, which neither system attempts to construct, a student diagnostic module to trace the misconception and build the model which again neither system entails, and finally a problem solving diagnostic tool which is the only form of diagnosis both system have. If the problem solving diagnostic tool detects that the student answer is incorrect, the student diagnosis should take over to detect the misconception by reference to the bug library, update the student model and then take remedial action. Both system do not go beyond the first step, that is diagnosing that the student answer is incorrect.

Micro-SEARCH attempts to remedy in a problem solving context, by complying to its own rules and conditions, nevertheless, the student-users found this not to be very effective. As explained before, if the user-learner chooses to apply a wrong transformation, then the system intervenes to warn the user about it but is not able to explain to the user why his choice of transformation is inapplicable in the current situation, for instance, why he may have to apply more transformations to get to the goal state, why it may take him in circles, why it does not lead to the solution, etc. The lack of proper student diagnostic abilities and consequently a student model, does enable the system to detect all these reasons and why a student got stuck or confused and why he cannot decide on the right transformation to apply.

PROUST, on the other hand, provides some remedial text material in the bug report which it generates after it has analysed a student's Pascal code. It suggests why the syntactic and semantic errors which appear in the student's code arose in the first place. Nevertheless, the bugs are not actual misconceptions, as explained before but deviations from the correct code, which may not be bugs at all but superior student solutions methods which the designer of the system did not anticipate. Such abilities would be extremely useful to represent in the student model and also for the system to adapt them in its knowledge base and resulting representation. Furthermore, the system does not allow the user to question its diagnostic abilities or ask for further clarification about these bugs, especially if these are unclear, as it sometimes happens when the system is unable to parse and analyse the student code completely. PROUST's remedial text is very much an off-line process, since the student's next step after the bug report is generated, is to leave the system and go back to his Pascal editor and correct his code.

PROUST's and micro-SEARCH's Decision Base

Q11. ***Do the systems have explicit representations of tutoring, domain and student knowledge?***

According to the model of a didactic operation, an Intelligent Tutoring System must be able to represent explicitly knowledge about its tutoring goals and strategies, about its domain knowledge and about its student-user as three independent models. The importance of having these three pieces of knowledge as three independent modules is that they can be inspected modified, traced and reused for some other domain. Neither of the two systems has any explicit or implicit form of tutoring knowledge nor do they attempt to diagnose student knowledge. Both systems's tutoring strategy is a problem solving one where the only output of diagnosis is a deviation from the correct answer. Nevertheless, as was argued before, PROUST has the potential to perform student diagnosis. With respect to domain knowledge PROUST, as a Knowledge Based Tutoring System, has an explicit representation of knowledge of Pascal syntax, of Pascal programming skills, and of knowledge about the problems it sets for the student to solve. The system's diagnostic abilities are responsible for detecting anomalies in the student code by reference to all these three sources of Pascal knowledge and skills. It checks to ensure that the code is syntactically correct by reference to Pascal syntax knowledge and programming skills, and if it is semantically correct by reference to the solution plan of the problem. Micro-SEARCH, on the other hand, has no diagnostic abilities, nor any domain knowledge. It has a simple black box problem solving tool which is used to build the complete set of solution paths for a given problem, prior to the course of interaction with the user-learner. Thus it can only support surface level tutoring. micro-SEARCH's tree is stored in a basic tree structure which is placed at the disposal of the student in pictorial form in order to browse through it. Student answers are evaluated against this tree.

Q12. ***What didactic constraints and resources affect tutoring: pedagogical goals, domain-independent tutoring strategies, domain-specific tactics, material***

sequencing, student monitoring and diagnosis?

Both systems have one knowledge source of constraints and resources: their domain knowledge model and the corresponding domain knowledge process model. Neither system attempts to build a formal student model to be used as a diagnostic source in the context of tutorial interventions and to enable the system to generate problem-solving episodes, nor do they have a tutoring model (i.e. they do not follow any formal teaching strategies), nor do they attempt to attain any pedagogical goals, or sequence any material. Both systems' only didactic resource, which is also their constraint, is a single pre-programmed domain-specific problem-solving strategy that involves diagnosing deviations in the user-learner's answer from "a" known solution path and invoking a prefabricated intervention accompanied with some form explanation. This results in both systems being rigid and not adaptable to reflect the user-learner needs. Both systems' strategy works through the domain-specific knowledge, the tree representation of the complete set of solution paths in the case of micro-SEARCH, and the frame-based goals and plans in the case of PROUST. This strategy is not a formal teaching strategy.

Although both systems perform some elementary diagnosis as part of their problem solving strategy and some feedback is given to the user, immediately after the user has selected a transformation in the case of micro-SEARCH, and in the bug report in the case of PROUST, this has no purpose other than to warn the user-learner that his answer is not the correct one. This feedback which is entirely prefabricated and not the product of diagnosis in the case of micro-SEARCH and constructed during the student's Pascal code analysis in the case of PROUST, is not used by either of the two systems as a source of constraints for generating remedial action. There is no material sequencing during the course of interaction, because both systems lack in communication skills (i.e. they entail no formal teaching strategies), there is no recipient of communication (i.e. there is no student model) and the domain knowledge is only used as a problem solving source and not as a source for material to be sequenced. Furthermore, both systems' only goal is to "help" the student reach a goal state in the case of micro-SEARCH or provide the correct Pascal code in the case of PROUST.

Lack of a formal tutoring model in any Intelligent Tutoring System gives rise to a lot of shortcomings. The student does not receive any sort of formal training with the concepts that he is exposed to because there is a lack of teaching strategies. The system cannot deduce a teaching strategy that suits best the student needs, and therefore it cannot individualise its instruction. The system does not have any means for helping the student with remedial action or to attain any pedagogical goals. Finally, concepts in the domain knowledge cannot be communicated directly to user-learner other than indirectly through problem solving.

Although both systems lack explicit didactic knowledge, some basic, nevertheless, unintentional tactics have been programmed into the two systems when they perform an intervention or trigger an explanation.

Q13. ***What diagnostic constraints and resources affect tutoring? missing conceptions, misconceptions, the systems' understanding of student behaviour?***

In the context of a didactic plan of action is very useful to make a distinction between local and global levels of decisions. At the local level the tutoring system monitors the progress of the student in carrying out the task set out by the didactic episode via student diagnosis and also defines the context of the system's intervention by resorting to the domain knowledge, if as a result of student diagnosis this is deemed to be necessary. This can certainly not happen in either of the two systems due to the lack of a student diagnostic model. Both systems' diagnostic tools detect deviations from the correct answer but cannot attribute these deviations to any missing conceptions in the student model or any misconceptions. At the global level, a system should, through a pedagogical process model, take decisions about subject material sequencing. If what is needed is remedial action, the student model is interrogated and then the system resorts to the domain knowledge to retrieve those concepts that are either missing from the student model or are the source of misconceptions, and generate a didactic episode to teach these concepts. If what is needed is to attain the next goal, then the next didactic episode is generated by resorting to domain knowledge to provide material for it.

Both systems perform some form of domain-specific diagnosis, but this does not serve to reveal missing conceptions or misconceptions in weak areas in the student's knowledge. Although micro-SEARCH builds some form of an internal representation of the workings of its student-user on each task, this is only the system's mechanism for detecting whether or not the student's answer is a viable transformation proposition. PROUST diagnoses syntactic and semantic errors in the student-user's Pascal code from "a" correct solution plan. PROUST "hypothesises" an origin of these errors and "suggests" how they can be corrected. The use of diagnostic information as constraints and resources for constructing didactic operations usually requires the system to inspect large portions of the student model. Both systems lack student modelling capabilities which is the main reason why they cannot be probed to understand the student-user behaviour or knowledge states and thus adapt both instruction and subject material to reflect the needs of the student. Diagnosis with both systems does not stretch beyond checking the correctness of a student-user's input and what can be derived from it, with reference to the systems's knowledge and skills.

The tutorial interventions of both systems are based solely on the local diagnostic information provided by the match or the absence of a match between the student-user's answer and the expected answer. The lack of a student model results into some rigid explanations being delivered to the student. Such explanations neither satisfy any didactic principles nor is their content tailored to individual student's existing knowledge. This results in expertise-oriented demonstrations in both systems, such as the tree diagram in micro-SEARCH and the Pascal program diagnostics in PROUST. Although PROUST's reconstruction of a program's goal structure allows remediation to situate the descriptions of errors relative to the student's intentions, it only suggests how the errors arose in the first place and not how they can be corrected or how the student's misconceptions arose in the first place and how they manifested themselves in problem solving. Thus errors only, and not misconceptions, are reduced to processes the student recognises as incorrect. Micro-SEARCH's lack of a link between explanations and associated diagnostic expertise does not allow remediation to situate any error description relative to the student-user's intentions.

The absence of proper student modelling capabilities and the fact that diagnosis is performed to detect a deviation from a student answer and without any recording, does not enable either of the two systems to use the results of diagnosis as a source in performing global means-ends analysis or as a source for constraints about the student-user's relative strengths and weaknesses.

Q14. ***What domain constraints and resources affect tutoring: Nature and structure of knowledge and expertise, explanations, content of tutorial intervention, material sequencing?***

Both systems are just problem solvers. Their problem-solving-oriented strategy is centred around the context and structure of domain-specific knowledge by using the systems's representational syntax as an anchor in performing the system intervention and in providing explanations. In the case of micro-SEARCH, the strategy is centred around the complete set of solutions represented as a tree. Micro-SEARCH's strategy takes the student on a didactic "trip" from branch to branch in the tree. The predefined paths of the fixed decision tree allows the system to foresee all the ways in which a student may reach the goal state from the current state, both short, long and "no" paths. Consequently, micro-SEARCH's explanations are not the product of any diagnostic expertise but are entirely produced by means of some predefined textual frame structures instantiated during problem design. This reduces micro-SEARCH's interventions into giving some problem solving hints. In the case of PROUST, the system's tactics evolve around the frame-based representation of both programming goals, and their plans of implementation. PROUST parses the student-user's code into a parse tree and then performs an analysis of this parse tree by matching it against goals in its problem description and the equivalent goals and their plans in its knowledge base. The student may choose to let PROUST take him through the analysis of his code, step by step, in which case the system displays the "results" in some frame-based notation (which the students found hard to read). The bug report which PROUST generates is a piece of normal text, which is the product of explicit chains of reasoning, with the explanations assembled, being directly associated with the system's diagnostic expertise.

As it stands, domain-specific knowledge is not a source for constraints about the relative importance of topics for either of the two systems, since there is no material sequencing process in any of the two systems. Neither of the two systems is capable of adapting to the user needs in terms of its instructional approach or in terms of domain context. There are no links between topics, and material sequencing is a simple linear selection process which the student is always in absolute control of. Nevertheless, the authors of problems contained within both systems can arrange assignments as a series of increasingly complex versions of problems, as happens with PROUST's assignments, where each problem evolves around a new problem requirement with which the student must comply with. This would constitute a natural global teaching sequence of learning experiences with the structure of the domain dominating the course of interaction.

In both systems, their only knowledge source is their domain knowledge, which due to the lack of a tutoring model and a student model can only serve as a source of problem-solving expertise. As a result this limits their abilities as computer tutors. The student-users master Pascal programming and transformation concepts and skills not by being directly exposed to these in the context of a didactic episode during which any misconceptions are cleared away by the system, but by being indirectly exposed to these through practising their current knowledge and skills in a problem-solving episode.

PROUST's and micro-SEARCH's target level

Q15. ***What is the target level of the systems' didactic operation: Behavioural, epistemic, or individual?***

The fact that micro-SEARCH does not address any internalised knowledge about mathematical transformations and expressions or Boolean Algebra or ordinary Algebra or

Trigonometry, that it does not seek to modify the user's knowledge states, and that it also capitalises on the fact that performing a transformation task and being exposed to such an environment constitutes a good learning context which provides students with ample raw material for actively forming their own interpretation and conceptualisation of the mathematical transformation domains, indicate that micro-SEARCH's didactic operation target level is exclusively the behavioural.

PROUST's didactic operation target level, on other hand, is largely epistemic. PROUST's didactic operation seeks to modify the user's knowledge state via practice by organising specific experiences (i.e. sequences of Pascal programming assignments), to expose the student to and thus cause changes (i.e. promote the novice programmer to an experienced programmer). In addition, PROUST addresses internalised knowledge, that is knowledge about Pascal syntax, about how to write programs and about problem descriptions, in a direct and organised way in order to provide some explanations which supply some interpretation of the domain and help towards the articulation of knowledge.

Q16. ***Do the systems provide hints, pieces of advice, corrections, remedial demonstrations, traces of reasoning, interpretations, explanations, simulations, motivation? At what level?***

The students who used PROUST and micro-SEARCH claimed that neither of the two systems motivated them in any way. This is because of the very inflexible style of tutoring, lack of a set of formal teaching strategies to choose from in order to individualise instruction and the rigidity created by their very restrictive interfaces. PROUST, in its bug report, provides some explanations as how the identified bugs arose. The students, however, did not find these adequate, because of the lack of provision of remedial action as to how these can be corrected, either in the form of advice given to the student or in the form of a remedial didactic episode generated by the system, or by performing a remedial demonstration for the user, or do the corrections for the user. Furthermore, the system does not provide the user with the facilities to do the corrections while in PROUST or trace the system's reasoning or ask a question. Nor has it the ability to provide an interpretation of its bug report. All it can do is give the user the choice to watch PROUST performing the Pascal code analysis which the students found extremely hard to follow.

Micro-SEARCH does not offer any real explanations about the reasons why certain transformations would be an "incorrect" choice of transformation to apply on the mathematical expression in hand. Explanation facilities in micro-SEARCH are nothing but pieces of "canned" text associated with each possible entry transformation which have been prefabricated during problem design. During the course of the interaction, micro-SEARCH cannot explain what the consequences would be if the student applies a transformation. Although these "tips", which the system provides to the user-learners, are based entirely on the tree diagram of the complete set of possible solutions, they are not the product of diagnosis like the bug report of PROUST and thus cannot serve as a real guidance to future transformation actions. Furthermore, the student is neither allowed to trace the system's reasoning nor ask for further interpretations of the system's actions. The system remains pretty much a black box throughout the interaction. Micro-SEARCH certainly does not make any corrections to the user input, largely because it does not allow for any big mistakes to occur- it knows which is the correct answer- and partly because it has not been endowed with such abilities. Consequently, it does not support any remedial demonstrations.

Neither of the two systems facilitate any form of simulation. The two systems provide all their facilities at the behavioural level. The problem of the two systems as regards the provision of individualised tuition is that both systems lack a set of formal teaching strategies from which to choose the best for an individual student and record this in the student model, and then apply this strategy in order to convey missing concepts or focus instruction on misconceptions. Another problem is the lack of a student model that records student misconceptions and links these to the domain knowledge so that the system can focus on these bugs and clear them away indirectly by generating a didactic episode or by advising directly the student how to do it. Alternatively, if the system knows what the student's bugs are (by examining the student model) and it knows to what part of the domain knowledge they relate to, then it can correct them in a SOPHIE-like style which involves doing the corrections itself while the student is watching, or it can simply demonstrate how these can be done by using its domain knowledge in another example. If the source of the problem is missing conceptions in the student model then the system can focus instruction on these and generate didactic episodes to fill these gaps. Student diagnosis can provide an indication of what is going wrong with the student, follow the links to the domain knowledge and record this bug in the student model so that the didactic operation can focus instruction on it.

Q17. ***Do the systems perform student diagnosis?***

Neither of the two systems performs student diagnosis in a problem solving context for the purpose of student modelling nor do they attempt to develop a working model of the student-user. Nevertheless, both systems, PROUST more so than micro-SEARCH, are equipped with some diagnostic capabilities. PROUST is able to diagnose interactively deviations of a student's Pascal code from the system's "correct" solution plan and generate a bug report where it lists all the bugs and how they arose in the first place. These, as was previously stated, are deviations from a correct design plan and not the product of student diagnosis against a bugs library. PROUST does not attempt to construct a student model, although it has the capability of producing a student model had it been endowed with a bugs library.

Micro-SEARCH, on the other hand, although it is able to detect deviations from a prefabricated set of solution paths, is completely unable to trace individual errors, why they arose in the first place or how they can be remedied. This is largely because micro-SEARCH's domain knowledge is largely compiled (it is a black box model of expertise). The only action it can take to safeguard the student from entering an incorrect solution path is to prevent him from applying the transformation of his choice or to warn him that the transformation of his choice would take him off a "short" solution path.

Q18. ***Do the systems pre-model the user?***

Neither of the two systems pre-model the user. This would be a useful start for the individual tuition which an Intelligent Tutoring System seeks to achieve, when a student logs on to the tutoring system for the first time. Although this facility may not be necessary for advanced students, it would be particularly useful for the weaker students with whom instruction can begin with teaching some necessary and prerequisite skills and or concepts which they appear to lack, after a close examination of this preliminary student model.

Q19. ***Do the systems model the user?***

As Nwana [1990b] argues, a tutoring system would hardly deserve the prefix "intelligent", if it did no do student modelling. Neither micro-SEARCH nor PROUST attempt to diagnose and model the student-user, in order to record a belief in the lack of knowledge of the task which a student-user has undertaken and try and warrant a counterbelief, according to Wenger [1988] by generating remedial action, for instance, in the form of a didactic episode. As a result, as the students who used the systems noted, both systems fail to individualise instruction: the same teaching rules are applied to all student regardless of a student's level of understanding and competence. PROUST, more so than micro-SEARCH, is able to diagnose interactively a variety of non-prefabricated deviations from "a" correct solution which it does not attempt to reconstruct for addition to a student model, although it has the capability for doing so. Even so, it does not attempt to model the user although in effect, what is actually performing while matching a student's code against its own solution plan, is the production of an overlay model which is almost half the effort, in the student modelling problem.

On the other hand, micro-SEARCH which constructs off-line a complete set of problem solution paths which it places at the disposal of the student-user during interaction, does not perform any diagnosis like PROUST does. This is because it lacks essential knowledge about its subject matters that would enable the system to identify student errors and to provide an explanation as to why they arose in the first place and how they can be remediated. micro-SEARCH is a tutoring system targeted exclusively at the behavioural level of operations, that is the evolution of skills. The system keeps a transformation matrix for each user on each task. The matrix records each step in the task the transformation chosen by the student and that chosen by the system. Entries on the diagonal of the matrix indicated that the student and the system chose the same transformation and non-diagonal entries indicate that the student selected what the computer algorithm thought was a non-optimal move. In addition to this matrix, the systems keeps a cumulative matrix that records for every student the transformations that the student applied across all the tasks carried out. In effect, these two transformation matrices can be viewed as student models. Associated with each off-diagonal entry in the matrix could have been, remedial material in the form of either procedural attachments or comments.

Q20. ***Do the systems support the various idiosyncratic ways which a student might choose to solve a problem?***

Both systems support, within the boundaries of the problems which they set (i.e. structure, syntax and context), the various idiosyncratic ways which a student might choose to use to solve the problem. This is particularly true of PROUST which allows the student-user to write his code free of any restrictions as long as the resulting Pascal program is syntactically and semantically correct, it addresses the problem in hand and remains within the problem bounds.

Micro-SEARCH on the other hand, levies a lot of restrictions on the user-learner. It does not allow the user any freedom other than that of selecting from a list of transformation options and even then, it would not let him apply his choice of transformation if that would take him out of a known solution path. As indicated before, it would not even attempt to explain why that particular transformation does not lead to the solution because it does not have the facility to trace the result of the application of this transformation would be. Nevertheless, micro-SEARCH would let the student go down a long path that would still lead to the solution.

The functionality of both systems, as described in Chapter 3, and the above evaluation suggest that both PROUST and micro-SEARCH could only be classified as knowledge presentation systems because they only possess a domain expert model of very limited scope as our investigation demonstrated. They incorporate neither any form of explicit didactic knowledge nor do they perform any real student diagnosis leading to the uncovering of a student's knowledge. Lack of didactic knowledge means that the system does not have a didactics process model, that is a pool of explicit computational teaching strategies to apply with the domain knowledge in order to satisfy some goals, other than some implicit operation which the system follows in all student cases- which can hardly be described as didactic. Lack of student diagnosis other than user input validation leads to the lack of student knowledge that would influence the course of interaction. Furthermore, the lack of both tutoring and student knowledge means that the system lacks any form of pedagogical control over the user-learner's activities and there in no material sequencing, no local student monitoring with goals and tactics.

This raises an important question: If all major schools of thought agree that a system can only be classified as a tutoring system, if and only if it possesses all three forms of knowledge (domain, student and didactic), why then aren't these two forms of knowledge included in PROUST and micro-SEARCH? Why are they simply knowledge presentation systems? Wenger [1988] claims that very few (finished) systems can be classified as knowledge communication systems.

4.3 DIDACTIC OPERATIONS IN KNOWLEDGE BASED TUTORING SYSTEMS

This evaluation of PROUST and micro-SEARCH against Wenger's model of a didactic operation highlights some important aspects about the three knowledge models (domain, student and tutoring) and their interrelatedness and interconnectedness in Knowledge Based Tutoring Systems.

4.3.1 Requirements for the development of an Intelligent Knowledge Based Tutoring System with a full-scale didactic operation

There are four requirements for the development of an Intelligent Tutoring System with a full-scale didactic operation:

[1]. The system incorporates all three knowledge models.

[2]. The three knowledge models are independent but may reference information within each other.

[3]. The system may branch the student anywhere in the domain knowledge structure as part of an alternative didactic plan of action.

[4]. The system has the ability to create additional domain knowledge from its existing domain knowledge and therefore establish additional didactic plans of action.

Implementation of a full didactic operation in a tutoring system assumes the existence of all three forms of knowledge and their corresponding process models. At the ***local decision making level***, that is within the context of a single didactic episode, the didactic operation assumes access to the domain knowledge to provide the content for a tutorial intervention that it would deem necessary. It also assumes access a set of local tactics or teaching strategies that it would deem appropriate to perform the tutorial intervention and also to guide the student's problem solving step by step. Finally, it assumes access to a diagnostic toolkit that would diagnose and record any missing concepts or skills or any misconceptions in the student's knowledge or behaviour and call for remedial action either to fill the gap created by missing concepts or remedy misconceptions.

At the ***global decision making level***, the didactic operation assumes access to the domain knowledge that serves as the source for material sequencing (i.e. enable the plan of didactic action to traverse the knowledge structure), and also to indicate the relative importance of topics. It also assumes access to a didactic model that can serve as the source of a set of teaching strategies for sequencing material and for guiding the student through didactic episodes. The didactic model observes the global goals that the system sets for the student to attain. Thus, the didactic model defines the pedagogical principles by which the system would tutor the user-learner. Finally, the system, needs access to the student diagnostic model to provide a means-ends analysis of the student which involves determining to which extent the student has met these goals and how the student can be classified as a user-learner, for instance, as a novice, advance beginner, competent, etc. This helps to unravel the relative strengths and weaknesses of the student and as a result of the student being classified, some additional information can be assumed for the student.

The use of the three components in the system's didactic operation, assumes that the three components are developed independently from each other. Nevertheless, their use in the context of a didactic episode or at a global level suggests that they are interlinked. For instance, the expertise process model should be able to infer from the domain knowledge either a correct answer or be able to trace the solution path to a correct answer without any interference any from the other process models or their knowledge. This suggests that the expertise process model should be able to act as a problem solver with its own knowledge. The diagnostics process model should be able to infer the student's current knowledge status and be able to call for remedial action without any references to the domain or tutoring knowledge. The didactics process model should be able to infer which is the best teaching strategy for attaining a goal from an educational point of view, and not which is the "best" for the student. Furthermore, it is good practice not to mix the three forms of knowledge for practical reasons, such as knowledge elicitation, modifications, extensibility, inspection and also the development of the corresponding process model is easier and the end result is actually what its name suggests.

Nevertheless, the three components must work together. For instance, the contents of the student model should point to the "best" for the student teaching strategy in the tutoring knowledge model and to those goals that have been attained by the student and those that are yet to be attained. This would enable the didactic operation to generate a didactic episode to satisfy the next goal, or try and satisfy the next goal, within the context of the current episode. The student model should also point to those domain knowledge parts that have been mastered by the student, including a measure on the level of mastery perhaps through an overlay model. This would help the didactic operation to focus instruction on those parts that have not been mastered by the student, or improve the mastery of current knowledge. The student model should also point to those domain knowledge parts that are the source of misconceptions for the student. This would enable the didactic operation to break away from its plan of action and generate or regenerate episodes to clear away these misconceptions. In the contents of the model of the tutoring knowledge, goals which the didactic operation will try to attain should point to that part of the domain knowledge that contains domain knowledge relevant to the goal, along with a pool of appropriate teaching strategies that would enable this.

The outcome of student diagnosis, may result in the didactic operation breaking away from the didactic plan of action in order to pursue a "remedial" plan of action that, in theory, may take the student anywhere in the domain knowledge structure. After the student has been diagnosed as having resolved the misconception then the didactic operation resumes its proper didactic plan of action. Similarly, the didactic plan of action may follow a different route to fill some missing conceptions that are diagnosed to be a source of problems for the student. In another instance, the system may "jump ahead" in its plan of action, if the student is diagnosed as an expert in a particular domain area.

The scrambled textbook approach suggested above assumes that the didactic operation should be able to pursue alternative didactic plans of action that may have not been incorporated in the system. This would impose an extra requirement on the system: the system must use its existing knowledge to form additional knowledge parts in the domain knowledge. This is because of the wide range of outcomes from a diagnostic process model which cannot possibly be predicted during the course of the design of the system. If a system is endowed with such a facility then this would remove a lot of restrictions from the instructional designer because, in theory, the designer would not have to anticipate all possible paths in the didactic plan a user or the system may follow during the course of interaction. The didactic operation may resort to its domain and student knowledge to construct interactively any "remedial" or other plans that it would deem necessary.

When these requirements are translated into a Knowledge Based Tutoring Systems context, they yield an equal number of requirements for the development of an Intelligent Knowledge Based Tutoring System with a full-scale didactic operation:

[1]. The system incorporates domain, student and tutoring knowledge representations.

[2]. There are explicit and direct links within, and between related knowledge parts of, the three knowledge representations.

[3]. The links include both hierarchical and non-hierarchical links.

[4]. The system is able to generate additional domain knowledge from, and link this to, its existing domain knowledge representation.

The existence of all three forms of knowledge and their independence assumes the existence of at least three knowledge representations: one for the domain knowledge, one for the student knowledge and one for the tutoring knowledge. This also assumes the existence of their equivalent process model: the expertise process model for reasoning with the domain knowledge, the diagnostic process model for reasoning with the student knowledge, and the didactics process model for reasoning the tutoring knowledge.

The knowledge interconnectedness assumes explicit and direct links between related knowledge parts of the three knowledge representations. For example, associated with a given domain knowledge part there should be an equivalent student knowledge part and a set of goals and teaching strategies. These links will have either been preset by the instructional designer or will have been inferred once during the course of interaction and persist thereafter. These links are necessary in order to avoid including, for instance, tutoring knowledge in the domain knowledge and at the same time enable a process model to work in synergy with the other process models by accessing information from other knowledge sources.

In addition to any hierarchical knowledge decompositions in the three knowledge bases that would allow for a variety of hierarchical didactic plans of actions to be implemented, there will also be non-hierarchical, explicit and direct links between different parts of the same knowledge that would enable the system to pursue as a result of some remedial action a different plan of action, for instance, a remedial one. Again as before, these paths will either be preset by the instructional designer or once inferred by the system will persist and will no longer need to be inferred again.

The system may not necessarily have access to a complete set of didactic plans of action (this may not be possible especially for large domains) but be able to generate additional didactic plans, during the course of interaction, by pursuing links in its three knowledge representations. This may be the result of a student request or the outcome of local monitoring of the student during a didactic episode. This consolidates the need for explicit and direct links to other knowledge parts anywhere within the knowledge which would otherwise need to be inferred globally- which may not be as successful as when these are generated in the context of a didactic episode- in order for this generative behaviour to take place. In addition, the system having generated a didactic episode or a series of these, should be able to link them to its knowledge representation for future reference.

4.3.2 Limitations of existing Knowledge Based Tutoring Systems with respect to the requirements for the development of an Intelligent Knowledge Based Tutoring System with a full-scale didactic operation

With respect to the above requirements, there are a number of limiting characteristics of the Knowledge Based Systems approach to developing a tutoring system with a full-scale didactic operation:

[1]. Knowledge decomposition, representation and inferencing is strictly hierarchical.

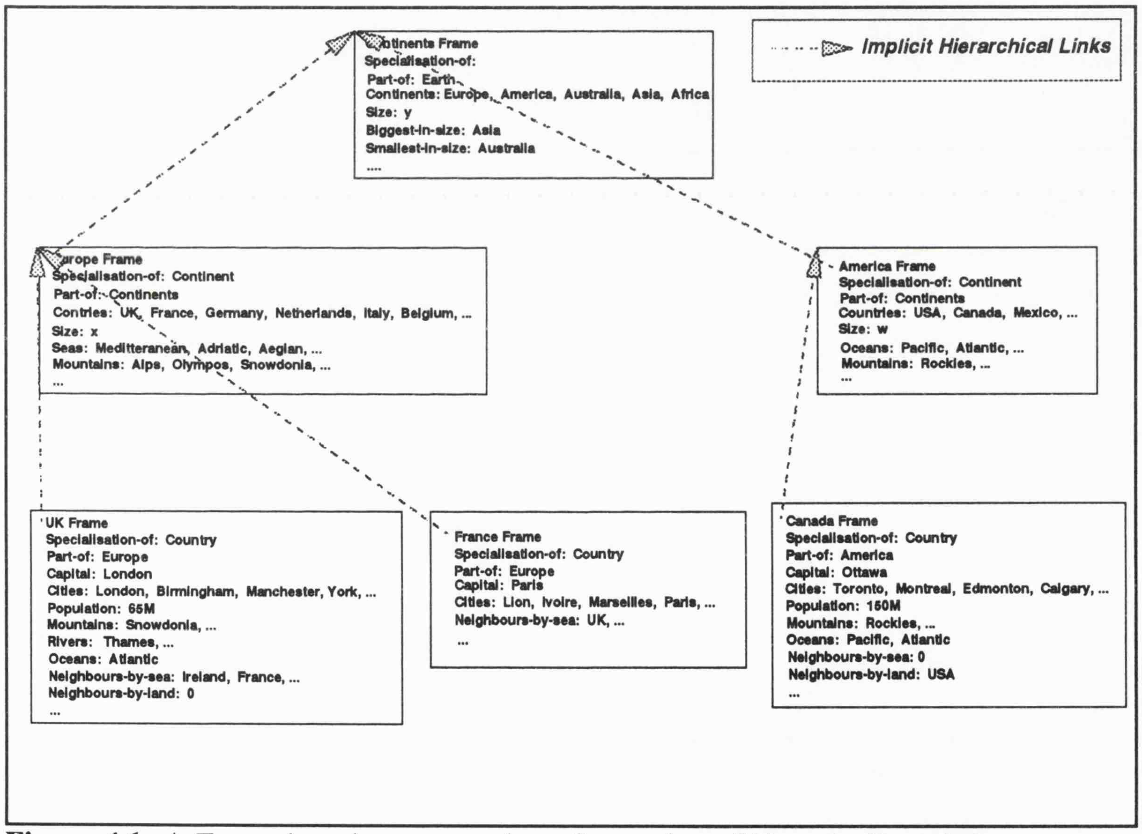
[2]. Knowledge decomposition and organisation is made from a single viewpoint which is then inflicted on the user. Reorganising a knowledge base from another viewpoint during the course of interaction is not currently feasible.

[3]. Reasoning requires that all necessary knowledge be made available to the inferencing mechanism prior to any interaction.

[4]. Lack of explicit information linking- all relationships are established through reasoning.

Artificial Intelligence knowledge representation techniques allow only for hierarchical knowledge decompositions and representations. In order to perform logical inferences in a domain, a knowledge-based system requires access to a knowledge representation of facts about the subject domain. The knowledge in this knowledge base could be represented using one of the many Artificial Intelligence knowledge representation techniques or a combination of one or more of these. With all these knowledge representation techniques knowledge about the subject domain is decomposed into its hierarchical constituents from a single viewpoint and one-way hierarchical trees or networks are built.

Consequently logical inferencing involves some form of depth-first (backward-chaining) or breadth-first (forward-chaining) searching or at its best some form of one-way branching (heuristic-chaining) through a tree or a network in a hierarchical fashion. Every time a new inference has to be performed the whole of the entire tree or network has to be searched from the beginning to the end or vice versa, in order for a goal to be inferred. Knowledge-based systems do not facilitate non-hierarchical knowledge representation and consequently non-hierarchical inferencing. Therefore, we are not able to represent non-hierarchical relationships between constituents of the domain knowledge unless we can establish some form of hierarchical relationship between them. Furthermore, the inferencing procedure has to perform one-way searching through the entire tree or network in order to establish a goal or infer a fact. Figure 4.1 below illustrates a portion of the Domain Knowledge of a Tutoring System for the Geography of Planet Earth in such a hierarchy.



Continents Frame
Specialisation-of:
Part-of: Earth
Continents: Europe, America, Australia, Asia, Africa
Size: y
Biggest-in-size: Asia
Smallest-in-size: Australia
....

Europe Frame
Specialisation-of: Continent
Part-of: Continents
Contries: UK, France, Germany, Netherlands, Italy, Belgium, ...
Size: x
Seas: Meditteranean, Adriatic, Aegian, ...
Mountains: Alps, Olympos, Snowdonia, ...
...

America Frame
Specialisation-of: Continent
Part-of: Continents
Countries: USA, Canada, Mexico, ...
Size: w
Oceans: Pacific, Atlantic, ...
Mountains: Rockies, ...
...

UK Frame
Specialisation-of: Country
Part-of: Europe
Capital: London
Cities: London, Birmingham, Manchester, York, ...
Population: 65M
Mountains: Snowdonia, ...
Rivers: Thames, ...
Oceans: Atlantic
Neighbours-by-sea: Ireland, France, ...
Neighbours-by-land: 0
...

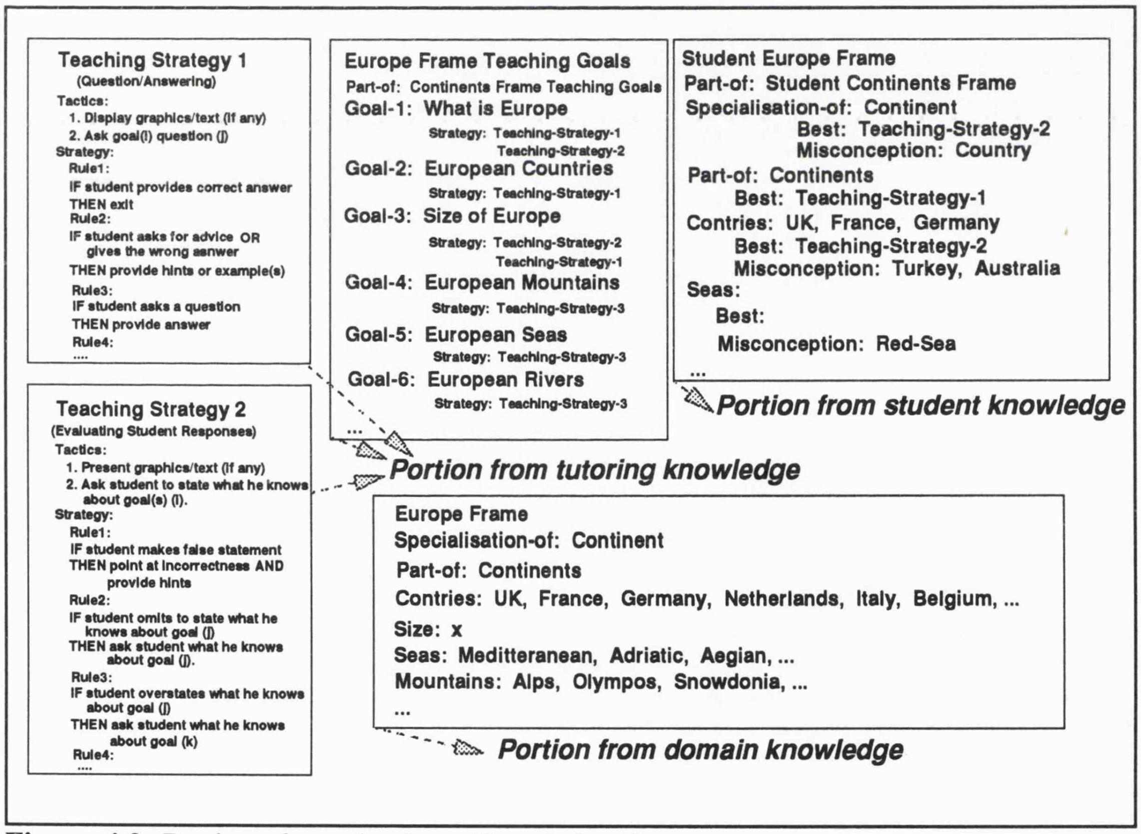
France Frame
Specialisation-of: Country
Part-of: Europe
Capital: Paris
Cities: Lion, Ivoire, Marseilles, Paris, ...
Neighbours-by-sea: UK, ...
...

Canada Frame
Specialisation-of: Country
Part-of: America
Capital: Ottawa
Cities: Toronto, Montreal, Edmonton, Calgary, ...
Population: 150M
Mountains: Rockies, ...
Oceans: Pacific, Atlantic
Neighbours-by-sea: 0
Neighbours-by-land: USA
...

Figure 4.1: A Frame-based representation of a portion of Domain Knowledge

Knowledge in the above example, has been hierarchically decomposed and represented in the knowledge base as frames. The viewpoint that has been imposed on this representation is that of physical boundaries (i.e. both continental and country borders). For instance, two ostensibly unrelated countries like the UK and Canada, which are members of different continents have at least one visible relationship: they are both wet by the Atlantic Ocean. How does one represent this non-hierarchical relationship without having to introduce another frame about the Atlantic Ocean which would be out of the scope of the current viewpoint?

As a result of this limitation, it would not be able to satisfy the second requirement, that is interconnectedness between related parts in the three knowledge components of the system, because it cannot cater for non-hierarchical links. For instance in Figure 4.2, how does one link the three frames from the three knowledge representations and at the same time denote the relationship between related parts?



```
Teaching Strategy 1
  (Question/Answering)
Tactics:
  1. Display graphics/text (if any)
  2. Ask goal(i) question (j)
Strategy:
  Rule1:
  IF student provides correct answer
  THEN exit
  Rule2:
  IF student asks for advice OR
     gives the wrong aanwer
  THEN provide hints or example(s)
  Rule3:
  IF student asks a question
  THEN provide answer
  Rule4:
  ....
```

```
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
  1. Present graphics/text (if any)
  2. Ask student to state what he knows
     about goal(s) (i).
Strategy:
  Rule1:
  IF student makes false statement
  THEN point at incorrectness AND
         provide hints
  Rule2:
  IF student omits to state what he
     knows about goal (j)
  THEN ask student what he knows
         about goal (j).
  Rule3:
  IF student overstates what he knows
     about goal (j)
  THEN ask student what he knows
         about goal (k)
  Rule4:
  ....
```

```
Europe Frame Teaching Goals
Part-of: Continents Frame Teaching Goals
Goal-1: What is Europe
          Strategy: Teaching-Strategy-1
                    Teaching-Strategy-2
Goal-2: European Countries
          Strategy: Teaching-Strategy-1
Goal-3: Size of Europe
          Strategy: Teaching-Strategy-2
                    Teaching-Strategy-1
Goal-4: European Mountains
          Strategy: Teaching-Strategy-3
Goal-5: European Seas
          Strategy: Teaching-Strategy-3
Goal-6: European Rivers
          Strategy: Teaching-Strategy-3
...
```

```
Student Europe Frame
Part-of: Student Continents Frame
Specialisation-of: Continent
          Best: Teaching-Strategy-2
          Misconception: Country
Part-of: Continents
     Best: Teaching-Strategy-1
Contries: UK, France, Germany
     Best: Teaching-Strategy-2
     Misconception: Turkey, Australia
Seas:
  Best:
  Misconception: Red-Sea
...
```

```
Europe Frame
Specialisation-of: Continent
Part-of: Continents
Contries: UK, France, Germany, Netherlands, Italy, Belgium, ...
Size: x
Seas: Medditeranean, Adriatic, Aegian, ...
Mountains: Alps, Olympos, Snowdonia, ...
...
```

Portion from student knowledge
Portion from tutoring knowledge
Portion from domain knowledge

Figure 4.2: Portions from the three Knowledge Bases

There are two ways to overcome this problem but they are also susceptible to problems. One is to attempt to mix knowledge parts but this would lead to information redundancy within the system, and the resulting knowledge bases would lose their identity as domain or student or tutoring knowledge. The other is to develop complex process models for inferencing with their equivalent knowledge, but this would require that the system performs inferences every time before making any kind of decision. These inferencing procedures would have to be performed not on selective parts of the knowledge representation but on the entire knowledge representation because of the knowledge's hierarchical structure.

In addition, because such a system can only support hierarchical plans of action it would be inconceivable, for example, to establish a remedial plan of action through a hierarchical knowledge representation by pursuing non-hierarchical links. This conflicts with the third requirement which assumes that the system is able to follow alternative plans of action which have not necessarily been depicted by the instructional designer. For instance, with Figure 4.1 the basic didactic plan of action would be to traverse the tree of frames by inferring and following the implicit hierarchical links. Every frame would provide the context for a didactic episode. The only deviation from this hierarchical plan the system is able to offer, is basically a change in the mode of node traversal. This involves the system switching from forward chaining (breadth-first searching) to backward chaining (depth-first searching) and visit the same frames but in different order.

Such a change in the mode of node traversal does not constitute a change in viewpoint or generative behaviour. To have real alternative plans of action the inferencing mechanism must be able to explore hierarchical links and alternative links, that is non-hierarchical links, and establish or generate a plan of action which may not be entirely hierarchical. This conflicts with the fourth requirement: the system's generative behaviour is considerably diminished because the system can only follow hierarchical plans of action as depicted by its knowledge structure. The fact that the system is not able to deviate from its hierarchical knowledge organisation when making an inference, lessens considerably its abilities as a generative system. The only form of generative behaviour stems from its hierarchical inferential abilities which does not result to an interesting combination of problems.

The second limitation of Knowledge Based Tutoring Systems is that for inferencing with a traditional knowledge representation to be successful, all available knowledge and any resulting knowledge combinations must all be made available to the inferencing mechanism prior to the interaction because during the course of interaction the system can only infer hierarchical relationships that are deducible from the knowledge structure. Therefore, the inferencing mechanism can only make strictly hierarchical decisions based on a complete knowledge structure.

The third limitation is that knowledge decomposition and organisation in the knowledge base takes place from a single viewpoint, and hence that the system can only impose that particular view of the domain (knowledge) on the system user. If the user wishes, or the systems infers directly or indirectly, that the viewpoint ought to change, then this assumes reorganising the knowledge base, a task which is far from possible to achieve during the course of interaction because it would involve breaking the hierarchical structure and constructing a new one from a different point of view. However, had the system included non-hierarchical links, the knowledge representation might be able to offer a number of alternative didactic plans of actions that would not necessarily assume reorganising the knowledge base because it would endow the system with generative behaviour.

If a system needs to have encoded all knowledge that is required for making a decision because it cannot generate additional knowledge from its existing knowledge structure, and can only impose a single view then how, for instance, in Figure 4.3, can the system engage the user-learner in a didactic episode where he is taught about those German-speaking countries of Europe? To achieve this, it would either require the system to generate the knowledge for such a didactic episode and link this to existing knowledge via non-hierarchical links, or restructure the knowledge base from another viewpoint, that of language boundaries.

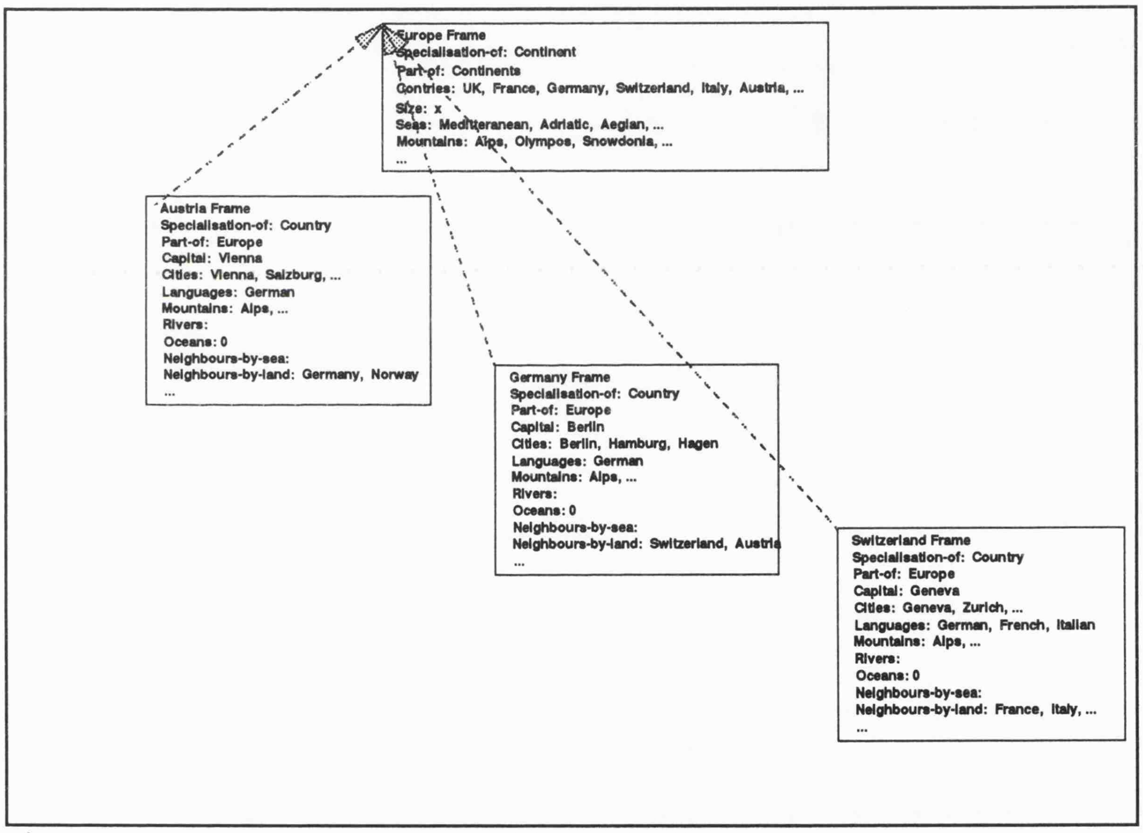

Europe Frame
Specialisation-of: Continent
Part-of: Continents
Contries: UK, France, Germany, Switzerland, Italy, Austria, ...
Size: x
Seas: Mediteranean, Adriatic, Aegian, ...
Mountains: Alps, Olympos, Snowdonia, ...
...
Austria Frame
Specialisation-of: Country
Part-of: Europe
Capital: Vienna
Cities: Vienna, Salzburg, ...
Languages: German
Mountains: Alps, ...
Rivers:
Oceans: 0
Neighbours-by-sea:
Neighbours-by-land: Germany, Norway
...
Germany Frame
Specialisation-of: Country
Part-of: Europe
Capital: Berlin
Cities: Berlin, Hamburg, Hagen
Languages: German
Mountains: Alps, ...
Rivers:
Oceans: 0
Neighbours-by-sea:
Neighbours-by-land: Switzerland, Austria
...
Switzerland Frame
Specialisation-of: Country
Part-of: Europe
Capital: Geneva
Cities: Geneva, Zurich, ...
Languages: German, French, Italian
Mountains: Alps, ...
Rivers:
Oceans: 0
Neighbours-by-sea:
Neighbours-by-land: France, Italy, ...
...

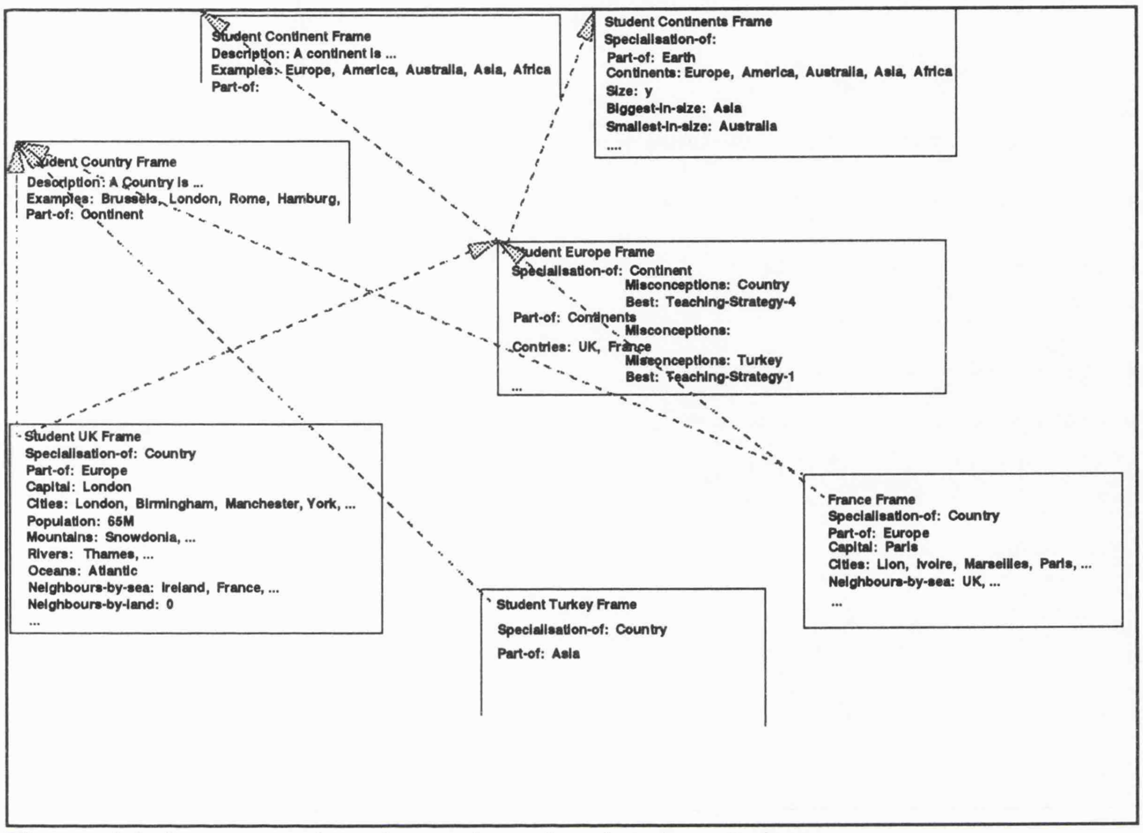
Figure 4.3: Alternative Viewpoints and Generative Behaviour

Finally, information linking with Knowledge Based Tutoring Systems is strictly and exclusively implicit. Therefore, in order for a system to establish relationships between parts in any of its knowledge representations, it has to perform reasoning. Furthermore, the same chain of reasoning will have to be performed over and over again every time the relationship has to be established. This adds to the complexity of the interconnectedness between related parts in the three knowledge representations. For instance, how can one directly describe the relationship between the knowledge of a student in the student model and its equivalent in the domain model? Or how can one depict the connection between a teaching goal and the relevant part in the domain knowledge representation?

Implicit-only information linking also adds to the complexity of the inference mechanism. Since the inferencing mechanism can only follow hierarchical links, the absence of explicit information links, especially non-hierarchical links, assume a very sophisticated inference mechanism in order to either follow or generate alternative plans of action not-defined by the knowledge hierarchical structure.

For instance, in Figure 4.4 below, how does one represent, first, that there is a misconception about Turkey in the Student Europe Frame other than by inserting the word Turkey in the Student Europe Frame, and second, what is the context of this misconception, and the relationship between this misconception and the Student Turkey Frame? Furthermore, how does one represent subset relationships, for instance, the Student UK Frame to the Student Europe Frame, and at the same time depict their relationship as a measure of the student's level of mastery of this subset relationship as, for instance, an overlay statistic? Or furthermore how does one represent non-hierarchical relationships established in the student's knowledge and at the same time provide a measure of the level of mastery of that relationship as an overlay statistic?

The evaluation of PROUST and micro-SEARCH in this Chapter unravelled four requirements about the interrelatedness and interconnectedness between the three knowledge models in order for a Knowledge Based Tutoring System to support a full-


Student Continent Frame
Description: A continent is ...
Examples: Europe, America, Australia, Asia, Africa
Part-of:
Student Continents Frame
Specialisation-of:
Part-of: Earth
Continents: Europe, America, Australia, Asia, Africa
Size: y
Biggest-in-size: Asia
Smallest-in-size: Australia
....
Student Country Frame
Description: A Country is ...
Examples: Brussels, London, Rome, Hamburg,
Part-of: Continent
Student Europe Frame
Specialisation-of: Continent
Misconceptions: Country
Best: Teaching-Strategy-4
Part-of: Continents
Misconceptions:
Contries: UK, France
Misconceptions: Turkey
Best: Teaching-Strategy-1
...
Student UK Frame
Specialisation-of: Country
Part-of: Europe
Capital: London
Cities: London, Birmingham, Manchester, York, ...
Population: 65M
Mountains: Snowdonia, ...
Rivers: Thames, ...
Oceans: Atlantic
Neighbours-by-sea: Ireland, France, ...
Neighbours-by-land: 0
...
France Frame
Specialisation-of: Country
Part-of: Europe
Capital: Paris
Cities: Lion, Ivoire, Marseilles, Paris, ...
Neighbours-by-sea: UK, ...
...
Student Turkey Frame
Specialisation-of: Country
Part-of: Asia
Figure 4.4: Representation of a portion of student knowledge

scale didactic operation:

[1]. The system incorporates domain, student and tutoring knowledge representations.

[2]. There are explicit and direct links within, and between related knowledge parts of, the three knowledge representations.

[3]. The links include both hierarchical and non-hierarchical links.

[4]. The system is able to generate additional domain knowledge from, and link this to, its existing domain knowledge representation.

However, these requirements yield a number of limitations with respect to the knowledge based systems approach to developing a tutoring system with a full-scale didactic operation:

[1]. Knowledge decomposition, representation and inferencing is strictly hierarchical.

[2]. Knowledge decomposition and organisation is made from a single viewpoint which is then inflicted on the user. Reorganising a knowledge base from another viewpoint during the course of interaction is not currently feasible.

[3]. Reasoning requires that all necessary knowledge be made available to the inferencing mechanism prior to any interaction.

[4]. Lack of explicit information linking- all relationships are established through reasoning.

Explicit hierarchical and non-hierarchical information organisation and linking, and consequently generative ability, are considered to be the foremost advantages of hypertext [Pirolli, 1991]. Nevertheless, hypertext on its own does not constitute a framework for developing an Intelligent Tutoring System because it lacks the logical inferencing abilities of Artificial Intelligence [Bonar, 1991]. Chapter 5 presents various hybrid models that integrate logical inferencing techniques from Artificial Intelligence with information nodes and linking from hypertext, and proposes one such model that promises to overcome the limitations of Knowledge Based Tutoring Systems that were discussed in this chapter.

CHAPTER 5: TOWARDS A HYBRID MODEL OF ARTIFICIAL INTELLIGENCE AND HYPERTEXT

Recent research and development on Artificial Intelligence has focused on hybrid models that are made up of Artificial Intelligence and Hypertext [Pirolli, 1991]. Several attempts have been made to create such models, but all these seem to have been made by individuals working in isolation and with no particular problem to solve in mind [Bonar, 1991]. These models utilise hypertext's information nodes and explicit hierarchical and non-hierarchical information linking abilities with Artificial Intelligence's logical inferencing techniques. None of these models have been specifically designed for Intelligent Tutoring Systems Development [Redfield and Steuck, 1991]. This Chapter briefly introduces hypertext, and then presents various such hybrid models, and proposes one such model, Hyperframes, that integrates Minsky's Frames with Hypertext's information nodes and links. The model is evaluated as a way to overcome the limitations of the Knowledge Based Tutoring Systems discussed in Chapter 4.

5.1 HYPERTEXT

[Please note that the term "hypertext" refers to hypertext the concept and the term "a hypertext" refers to a hypertext document].

A good way to understand hypertext is to read through the following: "Imagine walking into a public library and picking up a book on Mozart. You begin to read and learn that Mozart was an Austrian composer in the late 1700s. You wonder what else was happening in Austria then, so you go to the card catalogue, find a book on Austrian history, go to the stacks, locate the volume, and read it before you continue. In this last book, you find a reference to Salzburg, and you wonder what it looked like. Back to the card catalogue, and the stacks, to find a book with images from that time. Finally, you get back to Mozart and read of a piano concerto you have never heard. This time you head for the library's record collection and listening room. This process continues until you have either satisfied your desire for knowledge on the subject or worn yourself out searching for it.

Now imagine sitting at your computer and bringing up an electronic text system on music. You begin to read about Mozart. When you wonder about Austrian history, you simply highlight the text and request more information with a mouse click or a few keystrokes. To find images of old Salzburg, you use the same process. And to hear the piano concerto? The same. The only restriction to this seemingly endless fountain of knowledge is that the author of this electronic text system had to establish the connections for you to follow and provide the additional knowledge for you to retrieve." [Byte, 1988].

Another way to define hypertext is to contrast it with traditional text. Traditional flat text binds us to writing and reading chunks of text (i.e. paragraphs), in a *linear* sequence since all traditional text is *sequential*. This means that [Nielsen, 1990a] there is a single linear sequence defining the order in which the text is to be read. There are tricks for signalling branching in the flow of thought when necessary: parenthetical comments, footnotes, intersectional references, bibliographic references, and sidebars. All these tricks allow the author to say "here is a related thought, in case you are interested". There are also many rhetorical devices for indicating that ideas belong together as a set but are being presented in linear sequence. But these are rough tools at best and often do not provide the degree of precision or the speed and convenience of access that we would like.

Conklin [1987] argues that hypertext allows and even encourages the writer to make such references, and allows the readers to make their own decisions about which links to follow and in what order. Therefore, hypertext eases the restrictions on the thinker and the writer. Hypertext does not force a strict decision about whether any given idea is either within the flow of a paper's stream of thought or outside of it. Hypertext also allows *annotations* on a text to be saved separately from the reference document, yet still be tightly bound to the referent. Begoray [1990] argues that hypertext is *nonsequential*, meaning that there is no single order in which the text is to be read. Figure 5.1 is an example of a hypertext document.

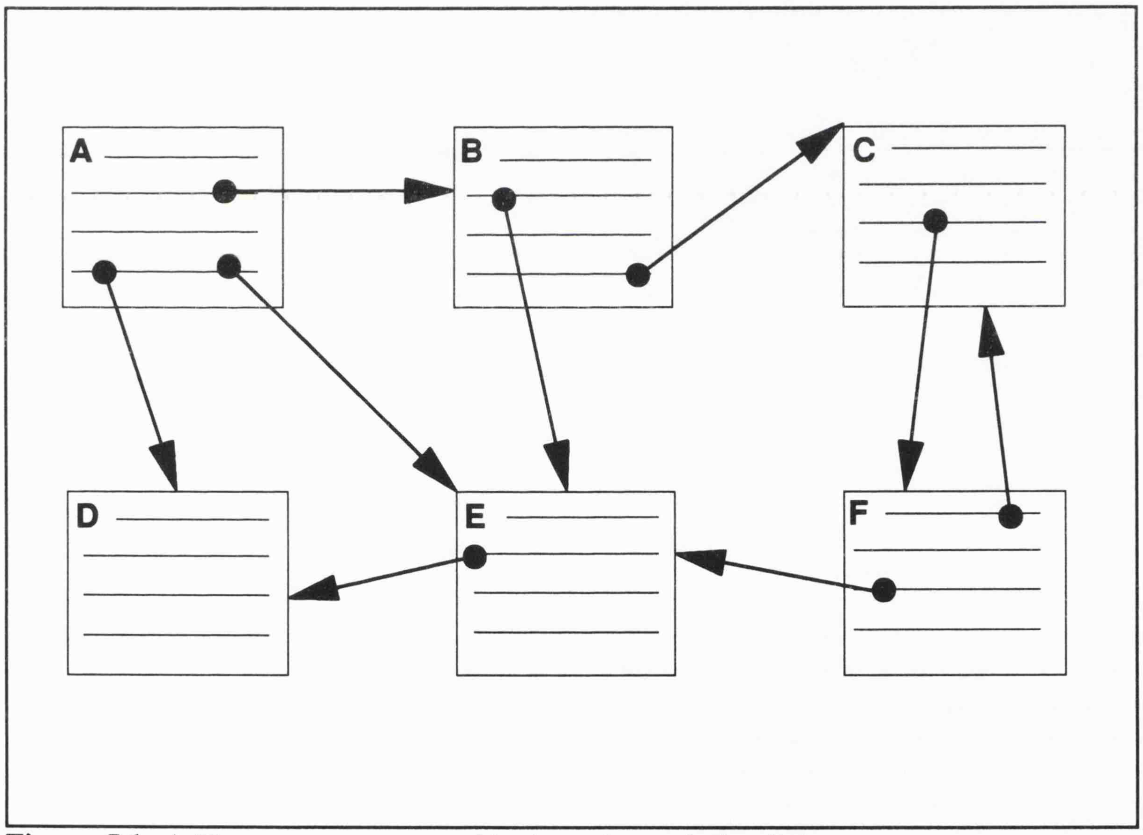


Figure 5.1: A Hypertext structure with six nodes and nine links

Hypertext presents several different options to the readers and the individual reader determines which of them to follow at the time of reading the text. This means that the author of the text has set up a number of alternatives for readers to explore rather than a single stream of information [Smith and Weiss, 1988].

Both Conklin [1987] and Nielsen [1990b] argue that much of hypertext's power is due to its **linkedness**, that is, the machine processible links between textual information which extend the text beyond the single dimension of linear flow, and its **nodedness**, that is the machine processible nodes of information which the hypertext user may use to build flexible networks which model a problem or solution. The information nodes may be [Frisse, 1988] objects relating to real-world objects, textual information, icons, etc. The links form the "glue" that holds the nodes together, but the emphasis with hypertext is or should be on the contents of the nodes. Hypertext nodes are normally illustrated as computer screens but they can also be scrolling windows, files, or smaller fragments of information [Begeman and Conklin, 1988], e.g. GUIDE III. The number of links is normally not fixed in advance but will depend on the content of each node. Some nodes are related to many others and will therefore have many links, while other nodes serve only as destinations for links but have no outgoing links of their own.

Figure 5.1 shows that the entire hypertext structure forms a network of nodes and links. Readers move about this network in an activity that is often referred to as **browsing** or **navigating** rather than reading, to emphasise that users must actively determine the order in which they read the nodes. For instance, if the hypertext-reader is going through document A then when he arrives at the point in the hypertext where there is a reference and a link to document B, he may follow this link and jump to document B without necessarily having finished document A. While in document B he can follow the link to document E, then to D, etc.

Thus hypertext can be defined as the computer supported, non-linear viewing of information where the reader or browser of the hypertext chooses what information to view [Smeaton, 1991]. A hypertext information space is made up of a number of fragments or nodes of information in which each node represents independent and autonomous pieces of information assembled into a network of nodes and links [Sculley, 1989]. Normally, the hypertext reader sees only the current node. It is normally impossible to draw a graphic representation of the entire hypertext on a computer screen since a typical hypertext may contain numerous nodes. Normally, the hypertext network is displayed, if at all, for the local neighbourhood surrounding the user's current location.

The "information" content of a hypertext document is greater than the sum of the information in all nodes as nodes are linked together via specially authored information links [Fiderio, 1988]. Links can be uni-directional or bi-directional. A given node can point to any number of other nodes, or none at all. The information links from a node represent pointers to related information and this binds or cements the whole information space together. By acting as the binding, the links themselves are an information resource. Nielsen [1990b] claims that nearly all hypertext systems are limited to providing one-directional links which means that the system can only show the user the links that have the current node as their departure point but not the ones that have it as their arrival point. This means that the system will tell a user where he can go next but not in what alternative ways he might have arrived at where he is now.

A user reads a hypertext document by doing a simple search on the contents of the nodes, often a string search or a search for boolean combination of word occurrences [Han et al, 1992]. This provides a node or a set of nodes which are a starting location for browsing and effectively jumps the user into the hypertext [Foss, 1989]. From then on the user is completely in control of the information being presented and can browse around the information space, freely following information links until he feels his original information needs have been satisfied.

Hypertext as a way of organising information has found numerous application areas [Nielsen, 1990a]. These include on-line manuals, education and computer-aided instruction, software engineering, computer-supported cooperative work, reference materials such as dictionaries and encyclopedias, etc. Often users are used to having vague information requirements, and to want to use a hypertext to explore the relationships between concepts which would have to be presented using conventional teaching in a linear fashion. By exploring or browsing a hypertext document, students can discover concepts, and how concepts are related, in the order that interests them. At other times, student-users may be precise in their requirements when seeking information from specific areas of the hypertext where they want to clarify their understanding of some concepts or of some concept relationships. Sometimes they want to get some comprehension of an overall picture of information from a hypertext when they would have a vague information need. The point here is that end users who use a hypertext for educational purposes have many types of information requirements corresponding to the many stages of learning.

Hypertext has been identified as a useful method of organising and manipulating information [Smeaton, 1991]. Information links can reflect either hierarchical or non-hierarchical structure of information. Other information links may represent semantic connections between nodes with similar contents. Retrieving information from hypertext is achieved through the various hypertext links, be it semantic links or simply structural links, by using information retrieval techniques.

To date there are no agreed-upon principles as to what actually makes a good hypertext document, in other words, what makes well-authored information links and node contents. However, there are a number of key attributes that seem to be necessary for hypertext usage to be useful to a population of users. Shneiderman and Kearsley [1989] have proposed what they call the three golden rules of hypertext:

[1]. A large body of information is organised into numerous fragments.

[2]. The fragments relate to each other.

[3]. The user needs only a small fraction at any time.

Once a hypertext has been developed, there are usually more problems encountered with actually using it. There is the cognitive effort required from users as they actually browse the hypertext. Related to this is the problem of disorientation. Since each node in the hypertext offers a number of possible directions in which to go by following given information links, users have to make a choice of direction. Often, they may wish to go in two or more directions from the same node, so they must choose one direction and **remember** to come back and follow other information links at a later stage. A hypertext-user may follow links taking him in a full circle that may result in re-visiting a node already viewed. All these may cause confusion to the hypertext-users who must try and maintain a cognitive map or picture of where they are in terms of the overall hypertext, where they have been, where they want to go next and at the same time assimilate the information presented as well.

A popular method that hypertext authors use to help the hypertext-users with navigation in browsing is to create guided tours or specially authored paths or recommended routes through the hypertext. These have the advantage that the readers of the hypertext can follow the authored route if they desire, and leave the route to return later if a node not on the recommended tour looks interesting. The disadvantage of the static authored tour is that it cannot cater for dynamic individual hypertext-user needs. The author of the hypertext is supposing or guessing at the reader's information needs. Satisfying a specific type of information need which is pre-defined in nature, can be done by authored guided tours but it is not the answer for users who have a vague information need or who want to get more of an overview of information.

For a hypertext-user who has an information need which is not satisfiable by a guided tour and who does not want to wander and browse around the hypertext information space but, nevertheless, wants to be guided in some way, a hypertext system should be able to provide explicit retrieval of explicit information from the hypertext while still preserving the browsing facility in some way.

5.1.1 A Hypertext Architecture

As Conklin [1987] and Nielsen [1990a] argue, although hypertext is both a *database method* providing a novel way of directly accessing data and a *knowledge representation scheme*, a kind of semantic network which mixes informal textual material with more formal and mechanised operations and processes, hypertext is fundamentally different from traditional databases from a user perspective. A normal database has an extremely regular structure defined by a high-level data definition language. All of the data follow this single structure where all the records have the same fields. In contrast, a hypertext information base has no central definition and no regular structure. Some of the nodes may be very extensive, with large amounts of information, and some of the nodes narrow with very small amounts of information. Furthermore, the links are put in because it makes sense in terms of the semantic contents of the nodes they connect and not because of some global decision. In addition, they argue that hypertext is an interface modality that features *control buttons* (link icons) which can be arbitrarily embedded within the content material by the hypertext user. As Conklin [1987] suggests, all these are metaphors for a functionality that is an essential union of all three. Nielsen [1990a] argues that in theory one can distinguish three levels in a hypertext system:

[1]. Presentation Level: user interface.

[2]. Hypertext Abstract Machine (HAM) Level: nodes and links.

[3]. Database level: storage, shared data, and networked access.

However, he goes on to suggest that no current hypertext system follows this model in its internal structure because they are a confused mix of features. This serves for providing a standard. The existence of this architecture will be assumed later in this Chapter but the precise functionality suggested by this architecture is beyond the scope of this thesis.

The database is at the bottom of the three-level model and deals with all the traditional issues of information storage that do not really have anything specifically to do with hypertext. It is necessary to store large amounts of information on various computer storage devices like hard disks, optical disks, etc. and it may be necessary to keep some of the information stored on remote servers accessed through a network. No matter how the information is stored it should be possible to retrieve a specified small chunk of it in a very short time. As Nielsen [1990a] claims, this is no different from a specification for a database. The database level should also handle other traditional database issues like multi-user access to the information and various security considerations, including backup. It will be the database level's responsibility to enforce the access controls which may be defined at the upper levels of the architecture. As far as the database is concerned, the hypertext nodes and links are just data objects with no particular meaning. Each of them forms a unit that only the user can view and modify at the same time and that takes up so many bits of storage space. A database level which has more information about its data objects is able to manage its storage space more efficiently and may be able to provide a faster response.

The HAM lies between the database and user interface levels. At this level hypertext determines the basic nature of its nodes and links and it maintains the relations among them. The HAM would have knowledge of the form of the nodes and links, and would also know what attributes were related to each, for instance, the node owner attribute or the last upgrade attribute. Links may be typed (i.e. they are textual), or may be plain pointers (i.e. they inform the user of the existence of a link). The HAM could serve for standardisation of import-export formats for information interchange in hypertexts since the database level has to be heavily machine-dependent in its storage format and the user interface level is different from one hypertext system to the next [Campbell and Goodman, 1988]. This would be particularly useful for interchanging hypertexts, which is more difficult than interchanging the component data in the nodes, since it also involves the transfer of linking information.

The User Interface level deals with the presentation of the information in the HAM, including such issues as what commands should be made available to the user, how to show nodes and links, and whether to include overview diagrams or not. The HAM level defines the links as being typed or simply being plain. The user interface level might decide not to display that information at all to some novice users, and to make typing information available only in an authoring mode, that is in a mode that allows the user to input his own information.

The distinction between reading and writing is one of the basic user interface issues. If the user interface level is to display the link typing to the user then it may introduce special notation for various forms for anchors, it may display an overview diagram, use different colours, etc. Nevertheless, this decision cannot be made at the user interface level without considering the likely form of the data in the HAM level. Icons could support hypertexts with more link types but a hypertext with hundreds of links types would probably require the use of the type names in the interface.

5.1.2 Hypertext Nodes

Conklin [1987] argues that although the power of hypertext lies in its machine-supported links, hypertext nodes also contribute to defining the operations that a hypertext system can perform. Nielsen [1990b] argues that nodes are the fundamental unit of hypertext but there is no agreement as to what constitutes a node. Hypertext nodes express a single concept or idea, hence they are much smaller than traditional files. Consequently hypertext introduces an intermediate level of machine support between characters and files, a level which has the vaguely semantic aspect of being oriented to the expression of ideas.

Hypertext invites the writer to modularise ideas into units in a way that allows, firstly, an individual idea to be referenced elsewhere and, secondly, alternative successors of a unit to be offered to the reader, for instance, further details, examples, bibliographic references, or the 'logical' successor. Normally, a hypertext node tends to be a strict unit which does not necessarily bear any kind of relationship with its neighbour nodes. The boundaries around nodes are always discrete and require sometimes difficult judgements about how to break the subject matter into suitable chunks. The process of building a semantic unit, such as an idea or a concept, with a syntactic unit such as a text paragraph or a hypertext node is a characteristic of hypertext.

Hypertext can enforce information hiding [Smeaton, 1991]. Sometimes the only clue a hypertext user has about the contents of a destination node is the name of the link or the name of the node. The hypertext author no longer makes all the decisions about the flow of the text; the reader continuously decides which links to follow. Since both the author and the reader have the option of branching in the flow of text, they both have to be process aware. Consequently, hypertext is suited for applications which require these kinds of judgements since it offers a way by which to act directly on these judgements and see the results quickly and graphically.

Hypertext supports reifying the expression of ideas into discrete objects that can be linked, moved and changed as independent entities. This offers enhanced retrieval and recognition because, to a much greater degree, abstract objects are directly associated with perceptual objects, like the windows and icons on the screen. Hypertext nodes that express individual ideas provide a vehicle which supports people's thinking and working in terms of ideas, facts, and evidence. According to Conklin [1987] there are four types of information nodes: ***Typed***, ***Semistructured***, ***Composite***, and ***Computed*** nodes.

Typed nodes are "free text" nodes containing textual information. This kind of node can be extremely useful, particularly if one is considering giving them some internal structure, since different kinds of nodes may be used to differentiate the various structural forms. Typed nodes can be used almost for everything: to record *Notes*, to denote *goals* and *constraints*, to represent *artifacts*, to record *decisions*.

Semistructured nodes are nodes which contain labelled fields and spaces for field values. They are very similar to records in the Pascal programming language. Therefore, this kind of node is not the structureless blank state into which one may place a word, sentence or a whole document. The purpose of providing a template for node contents is to assist the user in being complete and to assist the computer in processing the nodes. The less that the content of a node is undifferentiated natural language text, the more likely is that the computer can do some kinds of limited processing and inference on the textual sublinks. Some information elements must always occur together, while others may occur together or not, depending on how related they are in a given context and how important is to present them as distinct from surrounding information elements. Nevertheless, an information element that is atomic at one level may contain many components some of which are clustered together.

Composite nodes are used for aggregating related information in hypertext. Several hypertext nodes are affixed together and the collection is treated as a single node, with its own name, types, versions, etc. A composite node may be a collection of typed or semi-structured nodes or a collection of both. Composite nodes are most useful for situations in which separate items in a list or entries in a table are distinct nodes but also cohere into a higher level structure such as a list or table. A composite node allows a group of nodes to be treated as a single node. The composite node can be moved and resized and attached to a suitable icon reflecting its contents. The subnodes are separable and rearrangeable. The most flexible means of displaying a composite node is to use a constraint language which describes the subnodes as panes in the composite node window and specifies the interpane relationships as dynamic constraints on size and configuration. Composite nodes can be effective means of managing the problem of having a large number of named objects in a computer environment. Nevertheless, Conklin [1987] argues that as the member nodes grows and change, the aggregation may become misleading or incorrect.

Computed nodes are only available in computational hypertext systems with an embedded programming language. Such nodes are generated for the reader by the system. These may be typed, semi-structured or composite nodes.

5.1.3 Hypertext Links

The most distinguishing characteristic of hypertext is its machine support for the tracing of references. The issues here are, first, what constitutes a particular reference-tracing device as a link and, second, how much effort is permissible on the part of a user who is attempting to trace a reference. To qualify as hypertext, the system interface must provide links which move the user quickly and easily to a new place in the hyperspace.

An essential characteristic of hypertext is the speed with which the system responds to referencing requests. Often the reader does not know if he wants to pursue a link reference until he has had a cursory look at the referenced node. Sometimes, not all link traversals can be instantaneous. Providing cues to the user about the possible delay that a given query or traversal might entail is important. For instance, some visual feature of the link icon could indicate whether the destination node is in memory or on the disk, somewhere else on the network or archived off line. As Nielsen [1990b] argues hypertext links are frequently associated with specific parts of the nodes they connect rather than with the nodes as a whole. As illustrated in Figure 5.1 links are anchored at specific locations in the departure node while their destinations are the entire arrival node. Links provide the user with some explicit object to activate in order to follow the link. This anchoring takes the form of embedded menus where part of the primary text or graphics does double duty as being both information in itself and being the link anchor.

Links can be used for several functions [Conklin, 1987]. They can connect a document reference to the document itself, they can connect a comment or annotation to the text about which it is written, they can provide organisational information (for instance, establish the relationship between two pieces of text or between a table of contents entry and its section), they can connect two successive pieces of text, or a piece of text and all of its immediate successors, they can connect entries in a table or figure to longer descriptions, or to other tables or figures.

Links are explicit [Nielsen, 1990a] since they have been defined by someone to connect the departure node with the arrival node. Some hypertext systems also provide implicit links which are not defined as such but follow from various properties of the information. A hypertext system should make clear to the user why the destination for a link was an interesting place to jump to by relating it to the point of departure and following a set of conventions for the process of arrival. This calls for cues and conventions in hypertext notation.

Links have names and types. They can have a rich set of properties. Some systems allow the display of links to be turned on and off (that is, removed from the display so that the document appears as ordinary text). Hypertext links assume a set of mechanisms for creating new links, deleting links, changing links names or attributes, listing links, etc. Bielawski and Lewand [1991] argue that hypertext information links can be defined from a *conceptual* point of view or from a *functional* point of view. Although many taxonomies have been proposed for labelling *functional links*, there are two types of dominant functional links [Nielsen, 1990a]: *Associative* or *referential* links and *annotations*. With respect to conceptual links, there is one type of dominant conceptual link [Nielsen, 1990a]: *organisational* links. Some additional forms of hypertext information linking is by *computation*, and by *keyword.*

Organisational links differ from referential links (see below) in that they connect explicitly hierarchical information. They connect a parent node with its children and thus form a strict tree subgraph within the hypertext network graph. They correspond to the hierarchical ISA links of Semantic Networks. By being configured as a Semantic Network, hypertext defines a set of possible relationships between the units of information contained in the system. Having established organisational links in the hypertext, the functional dimension of these links may then be considered.

Associative/Referential links connect non-hierarchical information. Referential links are the kind of link that most clearly distinguishes hypertext. They generally have two ends, and are usually directed. The origination of the link is called the *link source*, the link source node is called the *anchor node* or the *departure node* and usually acts as the reference. The source can logically be either a single point or a region of text. At the other end, the *destination* node of the link is called the *arrival node* and it usually functions as the *referent* and can also be either a link point or a link region. A *link point* is some icon indicating the presence of the link. It usually shows the link's name and perhaps also its type. Or it may show the name and or type of the destination node. Sometimes, the display of links can be suppressed, so that the document appears linear. A *link region* is a set of contiguous characters which is displayed as a single unit, normally, an entire node of text and or graphics. This suggests that a chunk of text, the link region, can be referenced by a smaller chunk, for example, a sentence or even a word. Normally, the link region does not show the name of the link unless it is an entire node in which case the name of the node is displayed.

Annotations is a special link type which points to a small additional amount of information. The reading of an annotation typically takes the form of a temporary excursion from the primary material to which the reader returns after having finished with the annotation. Annotative links establish a part-to-whole relationship between units of information. Hypertext annotations are less intrusive because they may not be necessarily shown unless the reader asks for them. Many hypertext systems allow readers to add new annotations to the primary material even when they do not allow the reader to change the original nodes and links. Readers can use these facilities to customise the information space to their own needs.

Computed links are determined by the system while the hypertext-user is reading through the hyperdocument instead of being statically determined in advance by the author.

Keyword links occurs through the use of keywords. Hypertext entails mechanisms for allowing scanning of their content in search of selected keywords which can apply to nodes, links or regions or for arbitrary embedded strings. Link following and searching are similar. Each is a way to access destination nodes that are of possible interest. Link following usually yields a single node, whereas search can yield many. This makes a keyword link a kind of computed link.

SuperLinks connect a large number of nodes. There are several ways for dealing with having a single anchor connected to several destinations: show a menu of the links, go to all the destinations at the same time, or have the system choose for the user in some way based on the system's model of the user or simply by making a random decision.

Cluster Links can connect more than two nodes. Cluster links can be useful for referring to several annotations with a single link and for providing specialised organisational structures among nodes. One useful way to extend the basic link is to place attribute-value pairs on links and to query the network for them. Coupled with specialised routines, attributes lists allow the users to customise links in several ways, including devising their own type system for links and performing high-speed queries on the types. It is also possible to associate procedural attachments with a link so that traversing the link also performs some user-specified side-effect, such as customising the appearance of the destination node.

On the question of how sufficient hierarchical structures are, the answer is that they appear to be the most natural structures for organising different levels of abstraction which is a fundamental cognitive process. However, there may be cases where cross-hierarchical structures are required. For strictly tree-oriented hypertext navigation is very simple: from any node, the most one can do is to go to the parent, a sibling or a child. This diminishes the disorientation problem because of the simpler cognitive model of the information space. However, there may be cases where information elements need to be structured into multiple, interlinked hierarchies which hypertext can support with either hierarchical or non-hierarchical information links or with both.

5.1.4 Navigating through Hypertext

As Nielsen [1990a] explains there have been many approaches to navigation through a hyperspace. The simplest approach for navigation is to provide **guided tours** through the hypertext. A guided tour may be thought of as a superlink that connects a string of nodes instead of just two nodes. As long as users stay on the guided tour, they can just issue a next node command to see further information. With guided tours the reader can leave the guided tour at any spot and continue browsing along any other links that seem interesting. When the reader wants to get back on the tour, it suffices to issue a single command to be taken back to the point where the tour was suspended. Even though guided tours provide the option of side trips, they cannot serve as the only navigation facility since the purpose of hypertext is to provide an open exploratory information environment for the reader.

Another navigation facility is the **backtrack**, which takes the user back to the previous node. The great advantage of backtracking is that it serves as a lifeline for the user who can do anything in the hypertext and still be certain to be able to get back to familiar territory by using the backtrack. Backtracking is an essential for hypertext and it must always be activated in the same way. Furthermore, it should, in principle, be possible for the user to backtrack enough steps to be returned back to the very first node.

There have been other general 'history' mechanisms than the simple backtrack [Marchionini and Shneiderman, 1988]. The **History list** allows the users direct access to any previously visited node. Since users may want to return to nodes they have visited recently, it is also possible to use a visual cache where a small number of nodes are kept visible on the primary screen either by using icons or by simply displaying the names of the nodes.

Alternatively Hypertext may allow users to define **bookmarks** at nodes they may want to return to later. The difference between bookmarks and history lists is that a node gets put on the bookmark list only if the user believes that there might be a later need to return to it. A bookmark list is smaller and more manageable. However, it will not include everything of relevance. When the user defines a bookmark, the system may put the node's name on the bookmark list or it may prompt the user for a small text to remember the node by. Bookmarks would allow the user to resume the session with a hypertext system after an interruption and keep the state of the hypertext unchanged.

Hypertext may put **overview maps** at the disposal of the reader in order to ease navigation since the readers are expected to find their own way around the hyperspace. Since the information space will normally be too large for every node and link to be shown on a single map, a hypertext system may provide **overview diagrams** to show various levels of detail or provide **in and out zooms** to allow the users to see more or less detail.

Alternatively a hypertext system may provide a **fisheye view** and show the entire information space on a single overview diagram but in varying levels of detail. A fisheye view shows a great detail for those parts that are progressively further away. A fisheye view requires to being able to estimate the distance between a given location and the reader's current focus of interest and to be able to display the information at several levels of details. These conditions are easily met for hierarchical structures but they are harder to meet for non-hierarchical ones. Overview diagrams in general serve to help users understand their current location and their own movements by usually displaying the reader's footprints on the map to indicate both the current location and the previous ones.

Another navigation facility is the use of **landmarks** in the form of predominant nodes that denote special regions in the information space which may stand out in the overview diagram. It may be made possible for the hypertext system to define landmarks automatically by the use of connectivity measures but normally landmarks is the work of the hypertext author.

Another example of structured hypertext mechanisms is the use of **link inheritance** to allow simplified views of an information space without having to show all the links. In Figure 5.2, link inheritance replaces the individual links between nodes in an overview diagram with lines connecting clusters of nodes, thus simplifying considerably the diagram.

In general, there are two navigational dimensions: backward and forward moves among a given node and hypertext jumps. Moving back and forth within a node is seen as a linear left-right dimension like orthogonal left-right page turning while reading a book.

A search for information in hyperspace may be performed purely by navigation but this

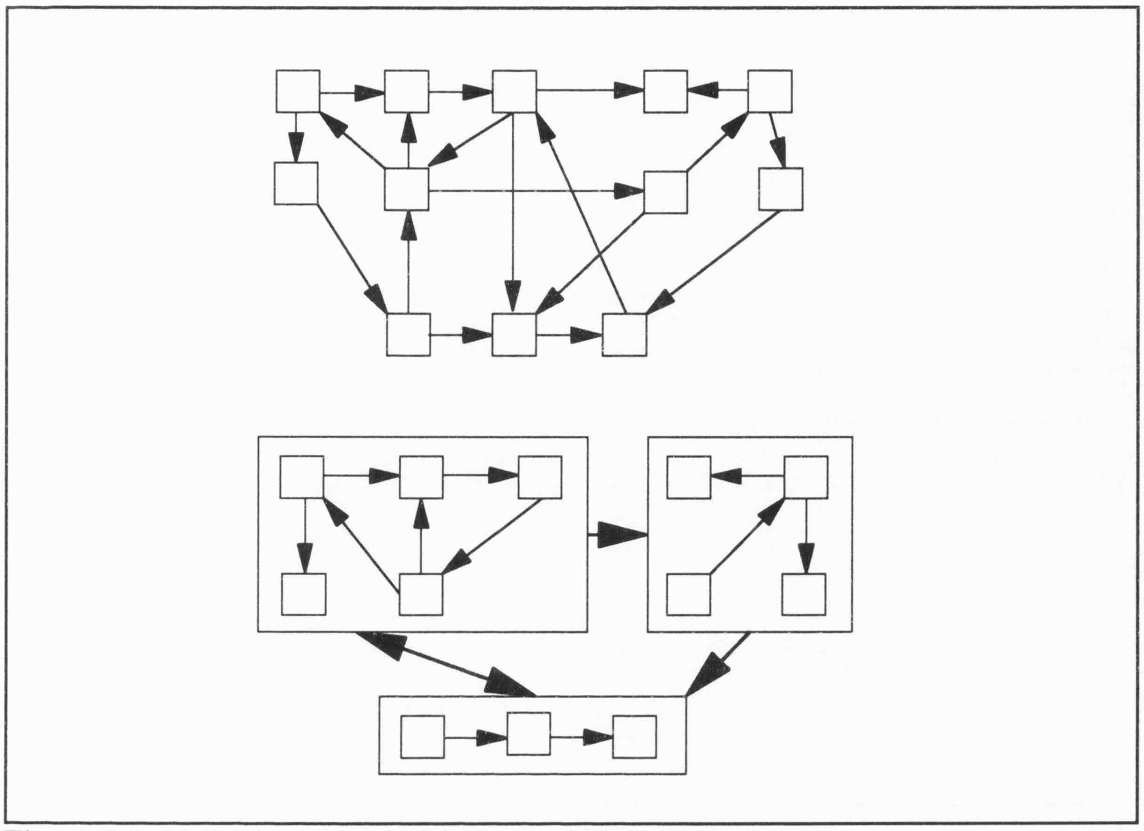


Figure 5.2: Link inheritance with structured Hypertext

is only best for information spaces that are small enough to be covered exhaustively and familiar enough to the readers to let them find their way around. Some hypertext systems make it possible for the user to have the computer find things through various search mechanisms. The simplest **query** principle is the full text search which finds the occurrences of words specified by the user. Some hypertext systems simply take the user to the first occurrence of the search term and some display a menu of the hits. Some hypertext systems integrate the search results with the overview diagram by highlighting those nodes that contain "hits". Some hypertext systems take this further by constructing a fisheye view since the number of hits in a given region of the information space would indicate how interesting that region might be to the user.

Some hypertext systems incorporate some sophisticated methods from the field of **information retrieval** [Smithson, 1991]. In a case where we have a hypertext available in which the links have already been constructed, one should be able to utilise the information inherent in the linking structure to perform more semantically meaningful searches than just plain full text searches. If a node matches a search, then one should also assign a higher score for the other nodes it is linked to, since the belief that the connected nodes are related justifies the propagation of scores among them. Nielsen [1990a] suggests that one way of calculating this score is by assigning the final search result for a node as the sum of the number of hits in the node itself- the *intrinsic score*- and some weighted average of the scores for the nodes it is linked to- the *extrinsic score*.

Query mechanisms can also be used to **filter** the hypertext so that only relevant links are made active and only relevant nodes are shown in overview diagrams which yields in much more navigable sub-hypertext. Finally, it could also be possible to filter a hypertext based on **relevance feedback** from other users in a kind of **voting filter**. Hypertext readers may choose only to see hypertext elements judged relevant by many previous readers.

5.2 HYBRID MODELS OF ARTIFICIAL INTELLIGENCE AND HYPERTEXT

Halasz [1988] argues that research into the next generation of hypertext systems must address seven key issues which, he argues, are the major weaknesses of current hypertext systems. These seven items are, Search and Query, Composition, Virtual Structure, Computation, Versioning, Collaborative Support, and Extensibility and Tailorability. Halasz [1988] foresaw that these needs could be best met by adding some "intelligence" to hypertext. He claimed that hypertext and Intelligent Knowledge Based Systems would be a "natural fit". There have been several attempts towards integrating Artificial Intelligence features into hypertext and vice versa but it appears that most of these have been the product of an individual's research and development within a single project, which often lead to conflicting views. Consequently, there is lack of consensus among members of both research communities about any potential integration of hypertext into Artificial Intelligence systems and vice versa.

Woodhead [1991] claims that in Artificial Intelligence systems decision making in a dynamic context rests with the system rather than the user. This results in a context-sensitive guidance by the system as opposed to the undirected navigation or browsing by the user in hypertext. On the other hand, in hypertext systems decision making in a dynamic context rests with the user who must make the decision about which node to visit next. Artificial Intelligence systems cannot use the real-world knowledge which their users have, thus their decisions will only be valid for specific structured information, governed by a set of rules which depict their machine intelligence. With hypertext systems, on the other hand, strategic decision making rests with users and this process is governed by the user's human intelligence. However, as the amount of available hypertext information increases, there will be an ever increasing need for additional means to reduce the apparent complexity to manageable presentation of information, to orientate and to navigate.

Bielawski and Lewand [1991] claim that the critical features of Artificial Intelligence to take into account when considering the integration of Hypertext and Artificial Intelligence, are knowledge representation, inferencing, and nonlinear association of information. They claim that neither Artificial Intelligence or hypertext alone are sufficient to integrate these functions efficiently. Artificial Intelligence and hypertext systems may have a synergistic relationship whereby they combine structure, control, knowledge representation, inferencing capability and problem-solving with flexible non-linear access to information and conceptual relationships and program navigation.

Information (or knowledge) in an Artificial Intelligence system is coded in a machine-readable knowledge base and requires the assistance of the knowledge engineer for updates. A hypertext is under the control of the user who can customise it by adding links and annotations. Rada [1991] argues that one way to integrate Artificial Intelligence capabilities in hypertext systems is to embed knowledge in links and to allow these links to trigger arbitrary computations. By doing so, he claims, human expertise is integrated into the system.

Diaper and Rada [1991] suggest where there is a certain degree of complementarity between hypertext and Artificial Intelligence. The differences between the use of semantic networks to support knowledge representation in Artificial Intelligence systems, and as a model of documents in hypertext lie in the nature of what constitutes the node content and in the properties of the labelling relations, that is the links. The nodes are semantically rich in hypertext since they are basically natural language text and relatively drained in Artificial Intelligence systems because the nodes contain a formal knowledge specification, for instance, as rules or semantic networks. The problem with Artificial Intelligence systems is that they are domain knowledge restricted. In contrast, the links are (implicitly) specified for Artificial Intelligence systems whereas they are virtually unspecified semantically in hypertext. The problem of hypertext links is that they are very rich in meaning and as such they may not be very well received by the reader.

In contrast to Artificial Intelligence, hypertext specifies default paths for navigating through information, but much of the decision making is left to the user. The representation is not structured enough to be interpreted by a machine rule base alone. However, an Artificial Intelligence system's structure is no more rigid than that of a hypertext to support hypertext-like interrogation of the knowledge base. Artificial Intelligence systems are usually required to provide explanations of their recommendations to users. Interrogation of the Artificial Intelligence system by the user is quite similar to browsing. Directionality in the hypertext system is more arbitrary. At each node in the hypertext system, information is linearly structured as flat text. The textual and graphic forms used in hypertext are typically more familiar to end users than the internal representations used by the Artificial Intelligence system. In addition, there is a non-linear structure of document links. The contrast between primary purposes of hypertext and Artificial Intelligence systems is given in the table below.

System	Primary Purpose
Artificial Intelligence system	Symbolic Reasoning (limited or expensive explanations)
Hypertext	Symbolic Annotation (limited or expensive calculation)

The provision of both facilities can be complementary, without becoming redundant argues Garg and Scacchi [1989]. Artificial Intelligence provide the means to proceduralise, to control. There are many domains where the necessary or available knowledge is so great in quantity, or complexity, that is not feasible for humans to make effective decisions. Providing Artificial Intelligence systems functionality either in the form of automated reasoning strategies or in the form heuristics, can make these domains tractable.

Hypertext, in turn, allows freedom to the user to explore beyond the rather narrow channels of rule-based information. In the remainder of this section several of the proposed hybrid models of Artificial Intelligence systems and Hypertext are presented.

5.2.1 Hypertext and Semantic Networks

A Semantic Network consists of nodes interconnected by various kinds of associative links. Links from one concept node point to other concepts which collectively form a definition of the original concept. These concepts are formal objects used to represent objects, attributes and relationships of the domain being modelled. A concept normally represents an intensional object and no concepts are used to represent directly extensional objects. Generic concepts represent classes of individuals by describing stereotypical members of the class and individual concepts are represented by relationships to more general concepts. Objects in the world have complex relational structure and thus they cannot be usefully taken as atomic entities or mere lists of properties. A concept must therefore account for this internal structure as well as for the object as a holistic entity. An intersection search is conducted as a spreading activation, breadth-first search of the nodes surrounding two concepts. The search spreads out by following links from the original two concept nodes until a point of intersection is found between the two concept nodes. The resulting path would indicate a potential relationship between these two concepts. Thus implicit relationships may be inferred from the explicit defined network.

Conklin [1987] suggests that building a directed graph of informal textual elements is similar to the Artificial Intelligence concept of semantic networks. Jonassen [1990] argues that a hypertext engine is primarily associative, enabling users to navigate through an associative network of ideas. The types of relationships denoted by the link structures may vary, though typically they are based upon associative relationships between the two nodes that they are connecting. Jonassen [1990] claims that hypertext structures are able to represent knowledge declaratively.

However, as Conklin [1987] suggests, what distinguishes a semantic network as a knowledge representation scheme is that concepts in the representation are indexed by their semantic content rather than by some arbitrary ordering as happens with hypertext. One benefit of semantic networks is that they are natural to use, since related concepts tend to cluster together in the network. In addition to this, an incompletely or inconsistently defined concept is easy to locate since a meaningful context is provided by those neighbouring concepts to which is linked. Woodhead [1991] claims that hypertext tends to have a relatively sparse control structure, and less density of attributes than semantic networks.

The analogy to hypertext is as follows: Hypertext nodes can be thought of as representing single concepts or ideas, internode links as representing the semantic interdependencies among these ideas, and the process of building a hypertext network as a kind of informal knowledge engineering. The difference [Schlumberger, 1989] is that Artificial Intelligence Knowledge Engineers are usually striving to build representations which can be mechanically interpreted, whereas the goal with the hypertext is to capture a collection of ideas without regard to their machine interpretability.

The computer can exploit the pattern of links in a hypertext and give the user different perspectives on the hypertext. The user may express an interest in causes and the system would organise information so as to emphasise the causal links. Rada [1991] also claims that the nodes and links of hypertext may be viewed as a semantic network. A link attributes meaning to the pair of nodes it connects, and a node may have more than one meaning, when it participates in relations of different types. Inheritance properties along hierarchical links and spreading activation in a semantic network, would both take advantage of hypertext's semantic network-like structure.

5.2.2 Hypertext and Minsky's Frames

Frames [Minsky, 1986] are formal models of knowledge representation which have a degree of psychological appeal as metaphors for reducing semantic complexity. Semantics is the key word; content nodes are the main features and the closer the similarities between nodes, the greater the physical proximity between them in the frames network. The nodes in this generalised structure of a semantic network are organised hierarchically so that properties can be inherited by nodes lower in the hierarchy. Nodes in the network of frames are linked by typed arcs but individual nodes have attribute slots at each node. These hold default values at their creation, or they may be instantiated with specific occurrence values or they may have executable procedures or methods attached to them, which are tested whenever a value is accessed or changed. Nodes normally denote concepts or specific instances of concepts.

Frame-based systems are formally more rigorous than the linking structures in hypertext. They are designed to be used in conjunction with automated procedures, whereas automated hypertext is still a research idea [Woodhead, 1991]. Only a handful of hypertext systems have implemented typed links as in frames or semantic networks. Another difference between hypertext and frame representations is in the number of links relative to the actual amount of information stored at the nodes.

Declarative representations have a particular useful property: they can be used to generate additional information by means of heuristic procedures. In addition, frame classification and inheritance provides useful object-oriented features to the system. If frames are used to implement a hypertext [Carlson and Ram, 1990], each hypertext information node in the hypertext would be represented as a single frame. Hypertext links to other nodes would be the slots within each frame, thus semantic information would be carried by the link names. This frame based hypertext system, would then support inheritance, default values for slots, and inference engines and reasoning. Specific link names would set up the (semantic) hypertext network of nodes an links. If frames are used strictly in their context, that is frames are linked to other frames in a hierarchical fashion, then the resulting hypertext network structure would be a hierarchical one. If this is overlooked, that is frames are linked to other frames with which they do not necessarily relate hierarchically, then, in principle, the resulting network structure would not be a hierarchical one.

5.2.3 Expert Systems with Hypertext support

This model of Hybrid Systems is primarily an Expert System that utilises hypertext features in accomplishing its problem solving tasks. The desired application of this hybrid model is to solve problems or provide decision support as opposed to locating retrieving and linking information in a nonlinear way which is the object of the next model of integration of the two technologies.

With this system model the Expert System component may contribute the following features to the overall design of the system [Bielawski and Lewand, 1991], [Gaines and Linster, 1990]:

[1]. Knowledge Acquisition and Representation Techniques for prototyping the knowledge base.

[2]. Knowledge Inferencing Techniques for providing an advisory role based on the knowledge base.

[3]. Techniques for dealing with Uncertainty.

[4]. An on-line validation facility for the domain knowledge.

These Expert System features may be coupled with the following features which the hypertext component may contribute to the overall design of the system [Bielawski and Lewand, 1991], [Gaines and Linster [1990]:

[1]. A programmable user interface to the Expert System that would allow the reader to extend knowledge and explanations in the Expert System knowledge base with further nodes.

[2]. A way of linking, locating and retrieving critical information either in the knowledge base and or in the hypertext information base.

[3]. A alternative method of explaining the system's reasoning.

[4]. An Annotation facility allowing background knowledge and explanations to be captured from the expert that would otherwise be entered in the structured knowledge base of the Expert System or that would not fit in the computational framework of the Expert System.

In this model of combination the Expert System's problem solving function dominates the overall system design and ensures procedural control or progression through the system

[Whitley, 1990]. The Expert System, being the dominant part, accomplishes its goal through formal mechanisms for knowledge representation and inference, such as production rules, decision trees, etc. Hypertext, however, also facilitates the problem solving process. The knowledge and inference that represent the core of the system can be obtained through traditional knowledge engineering techniques. Once this portion of the Expert System is constructed, the hypertext component can then augment the Expert System in any number of ways, from providing a user friendly interface, to locating pertinent information, to improving navigation and even to helping to extend the Expert System task by bringing forward procedural information contained within on-line texts.

With such a combination the system would use the heuristics based on the Expert System's rules to guide the user through the many decisions leading to a particular goal, and it would use hypertext to extend the communicative power of the Expert System in giving its results, by incorporating hypertext as an interactive front-end or an interface to the Expert System component. A hypertext interface would serve a much better purpose in communicating with the user, in accessing help, in acquiring critical information, providing explanation of the system's reasoning process, or narrowing the system's working domain. The hypertext component with its search and query techniques can then help reduce the amount of knowledge or information the system needs in order to reach a result or provide advice.

This hybrid model highlights an Expert System approach to problem solving, while using a hypertext component to increase functionality, efficiency or non-linear access to information [Rada, 1991]. The model deals primarily with systems that are intended to solve problems, offer advice, predict or behave in other ways like classical Expert

Systems. In a typical data-driven or goal-driven rule based Expert System that deals with a problem for which the data is incomplete, which involves uncertainty, for which there are many ways to reaching a solution, many variables or for which solution to the problem calls for procedural knowledge, hypertext may add to the functionality to accomplish a diagnostic task by providing textual or graphical information to the end-user. When diagnostic or problem-solving procedures are stored in on-line documents that are organised and presented via hypertext, it may be possible to develop a hybrid system that behaves like an Expert System but in which the knowledge based components may be explicitly linked via hypertext information links thus allowing the hypertext engine also to process the linked information.

With respect to the user interface, hypertext would offer an alternative method for user input into the system and would provide a means for creative dialogue-like interactions to take place. In most cases, a hypertext interface would allow a user to simply make choices among screen options. For instance, graphics can also be used for user input. The user's hypertext response would directly apply to the system rules (or decision trees, or frames, etc.) and cause specific actions to occur.

With respect to the linking, locating and retrieving of information in the knowledge base component of the system, hypertext would provide a way of improving an individual's access to information needed during the consultation process with the Expert System [Rada, 1991]. In some cases, the user may not be able to provide an answer in response to a question without resorting to additional information. In such a case, hypertext augments the knowledge representation and inferencing processes by replacing additional sets of rules, decision tress, etc. intended for naive users, with hypertext information. In some other cases, the hypertext engine may process hypertext information links into the knowledge base to locate and retrieve information in place of the Expert System inference engine.

With respect to explaining reasoning and providing on-line help, hypertext would offer triggering devices, such as buttons, that can be attached to rules or conclusions to retrieve specific textual and or graphical representations in order to help explain a system's reasoning process or to provide a degree of context-free sensitive help. In a similar fashion, any question or advice posed by the system to the user may entail a hypertext help facility which links to the system's knowledge bases or libraries. Such a form of help would yield a highly context-sensitive response by the system.

5.2.4 Hypertext with Expert Systems support

This model of a Hybrid System is primarily a hypertext information access system containing an integrated or embedded Expert System component [Rada, 1991]. In this model of combination, the hypertext features dominate the overall system design by providing the source materials and organisational structure for the system, leaving the Expert System component to provide specialised functions or assist in navigation. As a result, the system is a large depository of information, an on-line document, a collection of related graphics, or other type of linked information.

With this model of a Hybrid system the hypertext component provides a means of linking related text units in a conceptual, non-linear way. The Expert System component is embedded within the system to provide an alternative approach to finding information contained in the document, as well as additional information from related passages of text and graphics that are not found in the original documents. In such a system the two technologies work together to retrieve critically needed information to assist end-users in decision making processes. This model of the Hybrid System is then essentially an Intelligent Information Retrieval System. Rada [1991] argues that hypertext appeals because of its intuitive and graphic faculties, whereas Expert Systems appeal because of their formal, logical inferencing faculties.

Such systems usually originate with the text of original documents or a collection of graphics. These information sources often provide the backbone of the system, while the Expert System component would provide specialised local functions. This model of the hybrid system does not normally require the application of knowledge engineering since information in the system is mainly "text based" rather than "knowledge based". Although hypertext systems deal primarily with relational knowledge, Expert Systems representing procedural knowledge may be integrated into the overall system design. With this hybrid system model, there are two possibilities for integrating Expert Systems within the hypertext: to use procedural knowledge to help in locating pertinent information or to introduce applied knowledge and inference abilities that cannot be represented in the hypertext engine alone.

In this model, the Expert system component would provide its inference techniques for locating information in hyperspace by narrowing down the search domain or pointing to information that is used functionally as, for instance, in a problem-solving process. Hypertext as an information retrieval tool is useful only when the reader knows what he is looking for and can identify the information by its link or by how is labelled within the system. The Expert System component can take over the information retrieving task by asking the user questions about the information use. The Expert System can then apply a set of rules to separate what information might be useful from what might not and therefore, narrow the search domain based on the user's needs. Rada [1991] argues that better navigation of hypertext can be achieved by using the Expert System inference engine to construct paths in the hypertext in response to user queries. Sections of the hypertext may be used to illuminate the rules across which logical inferencing has arrived at particular conclusions. In explaining how a particular conclusion has been reached the inference engine interacts with the hypertext and highlights the textual sources for the various inferences. The facts utilised by the rules are available in an expanded and more accessible within the hypertext.

The Expert System component may also be used to integrate procedural knowledge into hypertext for solving problems, such as diagnosis, configuration, classification, etc., in which case the hypertext author may associate Expert System capability with any given node with hypertext. The Expert System would work in synergy with hypertext, communicating important information back and forth to hypertext.

Diaper and Rada [1991] suggest that an obvious combination is to use the semantically rich nodes of hypertext with the well specified, computable links of Expert Systems. They argue that the opposite combination of a weak knowledge representation within nodes and a rich but incomputable set of links is almost certainly disastrous as the users of such a hypothetical system would have problems understanding both the nodes and the links. They claim that a potentially useful **Expertext** system would have nodes that are readily understandable by users and rich because they would constitute natural language text, diagrams, figures, tables, photos, etc. Then by having well specified computable links that can be operated on by an inference engine, the Expertext user can be advised or guided as to the order of node presentation and thus the human navigation problems associated with hypertext may be considerably reduced or eliminated.

This is not to say that Expert Systems would provide better support for information linking as opposed to hypertext. The rules on which an Expert System is based are often inadequate when the Expert System attempts to explain its decisions to users. Hypertext can be integrated in an Expert System so that by working in synergy with hypertext the Expert System can explain better its decisions by offering hypertext information to the user when the Expert System is questioned.

Expertext would offer a textual description of the rules being activated by the Expert System component and the reader would have the ability to influence the traversal of the underlying semantic network during run time. The reader would guide the Expert System because he would be able to understand what the Expertext was attempting. The Expertext would potentially be less domain restricted than a traditional Expert System since the reader would detect inappropriate activation of rules at run time and could, in effect, suggest more efficient strategies of traversing the network, without having to be concerned with the low level complexities of such traversals.

Hypertext links are often difficult to follow. An Expert System can be integrated in hypertext to help the user find relevant information. Guidance may involve providing automatically the next section of text, or a set of suggestions listed according to some degree of priority for what the reader should see next, backtracking with non previously visited side path options, information about previously visited paths.

5.2.5 Automating Search, Linking and Inference in Hypertext

Rules in Artificial Intelligence Systems are often conditional match-execute pairs that are themselves based upon a considerable amount of knowledge. If a fact condition comprising the first half of a rule is positively matched, then the second half of the rule, often an action, is fired. In declarative representations facts and rules take broadly the same form. Rules do not normally operate in isolation. Instead they are nested or linked together into inference chains- which are actually paths through the decision tree composed of all possible rule combinations. In some Expert Systems, special meta-rules are sometimes invoked. These may add new facts to the knowledge base, or modify the rule base.

In hypertext the function of the meta-control is normally the responsibility of the user. However, there are two areas in which Artificial Intelligence techniques may lend themselves to hypertext: the knowledge representation and the user interface. A loosely structured knowledge base will require greater development of interface facilities for a non-expert user to achieve an acceptable solution with the same degree of ease.

A further problem with hypertext is the speed with which links can be created between nodes. Although hypertext systems have sophisticated text search facilities across some or all the nodes, it is still necessary for the user to initiate the linking of nodes manually, at a local level. Currently, hypertext systems provide little or no computational power to provide new group solutions, for instance, to generate new links from a batch of dynamic data items.

The approach followed by many hypertext systems is very basic: the user can choose which paths to explore, to what depth, and how to backtrack. Very few hypertext systems provide filtering mechanisms which combine criteria and operators, similar to certainty factors found in Expert Systems, that can be shifted to produce either a set of close-fit alternatives or a single best-fit solution from the set of the existing explicit paths.

What would reduce the expensive activity of manual searching or linking in those cases where a path has not been found is some means by which the hypertext system could spread its network more widely. The system would be able to anticipate the reader's likely requirement at run-time, from some combination of attributes of the network structure or of the search criteria so far.

Artificial Intelligence Search Techniques, like best-first search and heuristic search, can be welded into hypertext to take over this computational task searching [Woodhead, 1991]. These are semantic filtering techniques which use heuristic rules for searching or for linking new concepts in hypertext by making use of the hierarchical inheritance of attributes or properties of objects in the domain knowledge as opposed to the purely mechanistic searching techniques currently employed in hypertext systems. Such semantic filtering techniques would use any knowledge which is available about the problem of direct searching, although they do not actually require a complete problem description.

With these techniques the solution does not necessarily have to exist in hyperspace, it may have to be generated from existing information using inference rules by creating ad hoc implicit links. These techniques would apply inference rules to the problem knowledge to determine which direction, from the present position, offers the most promising chance of a solution. This may involve abandoning some branches of the tree or problem space only to return to them as other possibilities are themselves exhausted or become unlikely to yield an acceptable result. The solution will be an optimal or best fit within constraints designed to limit a combinatorial explosion of possible generated solutions, in other words, it may not be a perfect match. These techniques may be used within a framework of natural language querying or in combination with standard Boolean criteria, such as the operators AND, OR NOT.

Searching and linking are bound up with each other in hypertext. Some form of the searching, whether automated or user-driven, is necessary prior to information linking. With automated information indexing in hyperspace, linking results in a key index. This precedes and facilitates individual term searches. This yields to a double problem. Firstly, recall measure; a search mechanism needs to be able to access a high percentage of relevant items for any given search criterion. Secondly, precision measure; of the items retrieved, a high percentage need to be relevant to the search criterion. Unfortunately, the two measures tend to be inversely related in automated systems. To encompass a high percentage of relevant items, it is likely that search criteria will have to be fuzzy. This in turn means that items will also be retrieved which are not themselves relevant to the search for a given term. Approaches such as term weighting that increase the precision measure of a search also run the risk of being too specific [Smithson, 1989]. All items may be relevant, but they may be only a subset of relevant material. Term weighting presupposes certain *a priori* conclusions about relevance.

Combinations of searching and linking may lead to a compromise, more closely approximating the ideal or retrieving all and only those items corresponding to the search criteria. A typical solution to this problem would be the integration of a bi-directional chainer to help hypertext converge on optimum solutions. With this hybrid model of a hypertext system, the forward chainer would establish a high recall measure, being able to treat fuzzy search criteria, and thus access a high percentage of search material before converging, and the backward chainer would establish a high precision measure, being able to focus only on relevant search material.

5.2.6 Artificial Intelligence techniques for dealing with Uncertainty in hypertext

The techniques employed within Artificial Intelligence Systems to deal with uncertainty can be broadly divided into those methods which seek to reduce the search space (tree/graph pruning) or direct search along a single path, and those which calculate the degree of uncertainty accompanying the eventual solution. The techniques can also be divided according to the means with which they achieve their goals: probabilistic and heuristic methods. Sometimes there is an overlap between probabilistic and heuristic categories, probabilistic reasoning often underlies the implementation of fuzzier or more relativistic representations.

With Artificial Intelligence Systems raw data and/or rules are used to compare incremental possibilities, and perhaps to generate new information. Most Artificial Intelligence Systems are structured to produce a single answer or limited range of answers to a user query, for example, a diagnosis, although a few more sophisticated environments give their user the options to develop parallel models, for instance, to generate and explore alternative worlds [Woodhead, 1991].

With an Expert System, for instance, the goal may be to estimate the most likely outcome (i.e. the match of a result against a starting position and conditions) given uncertainty, lack of sufficient information to produce an exact answer, or because testing all possibilities is infeasible. Similarly, users of hypertext usually encounter problems of uncertainty, disorientation in hypertext terms, as to where they are in the hyperspace, as to where they can find something which they are looking for, as to where is the best place to find something which can be of interest to them, as to which direction will be the most promising one, etc. As with the Expert System, the user is likely to be seeking a relatively small number of possible and/or acceptable solutions for each new situation encountered.

To reduce the problem of having to manually search through the entire hyperspace, hypertext could incorporate certainty factors with nodes and apply uncertainty techniques which would calculate the degree of uncertainty accompanying possible solution paths and attach certainty weights to the links to indicate how likely the path is to lead to the desired goal. This would help reduce the search space by eliminating all those paths that are unlikely to lead the user to the goal, and then direct the user search along those paths which are likely to lead to the solution. Alternatively, it could just calculate the degree of uncertainty accompanying possible solution paths and let the user make the decision as to which paths to eliminate, if any. Both the probabilities and the heuristics techniques could then be used to carry out the process of calculating certainty weights and select those paths which most likely lead to the user desired goal.

5.2.7 Expert Help in Hypertext

Correlating Expert Information according to common themes, and eventually organising it into a hierarchical rule-base or decision tree is an enormous task without the use of automated tools. A number of Artificial Intelligence toolkits now include hypertext-based tools, [Woodhead, 1991], for transcript analysis and even induction mechanisms for finding patterns or rules in the data. These patterns or rules may be used in two ways. Firstly, to help authors structure their materials and, secondly, to construct a dynamic user model rather than rely on the kind of limited stereotype or average model implicit in most Knowledge Based Systems.

In addition, hypertext materials may need to be structured so that naive users can easily query the finished, self-contained system. This is desirable both to train new experts and because is highly dubious whether even the most subtle Expert System could ever be allowed to make unsupervised decisions. Thus users must be able to ask for justification. In many cases, these requests are likely to be *ad hoc*, unforeseen by the system designers [Whitley, 1990].

Hypertext mechanisms have the power to provide such a conceptually simple overview of the system's concepts and dialogue interactions. This is a less formal approach than that of conventional Help facilities or of the Explain Decision option available in many Expert Systems. Indeed, very few Expert Systems allow their users to browse through their knowledge bases or rule bases in an *ad hoc* fashion [Whitley, 1990] as distinct from their active decision-making nodes.

5.2.8 Natural Language Processing Interfaces in Hypertext

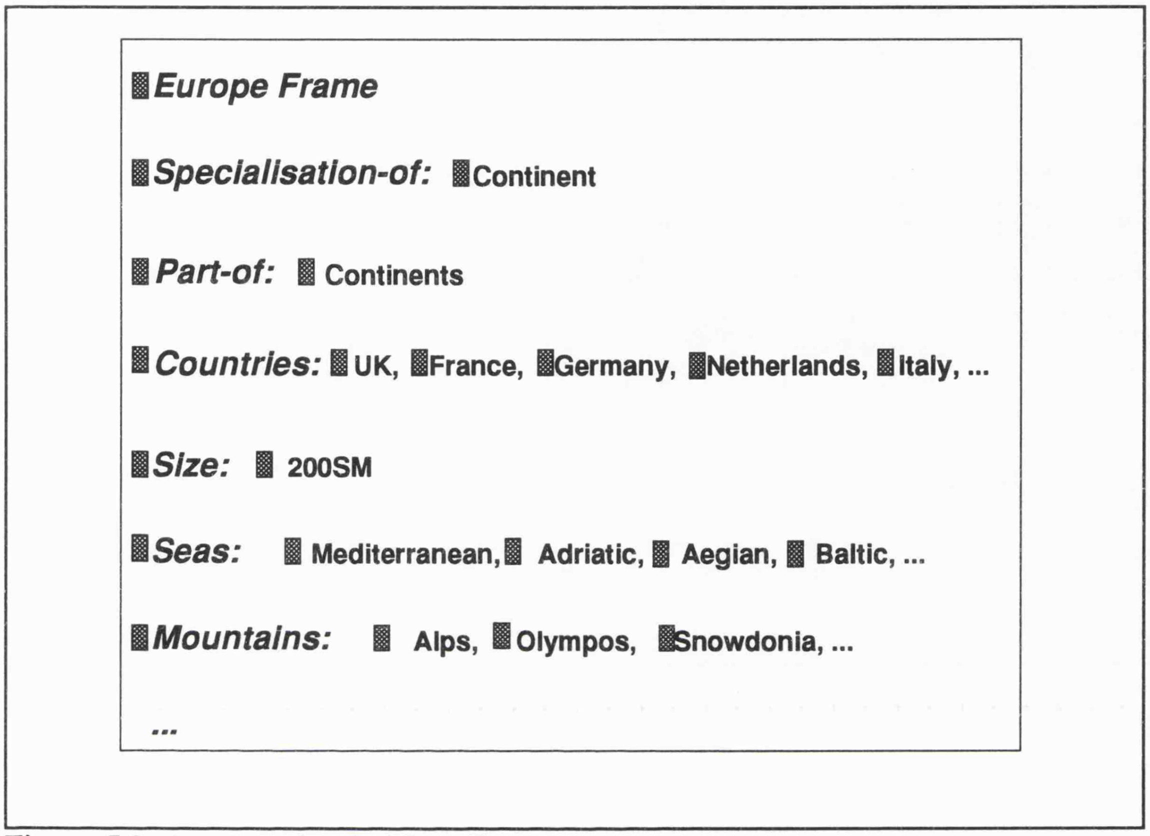
Textual information is the most crucial content of most hypertext applications. To be useful to the broad community of end-users, information ideally would be represented in natural language. Artificial Intelligent approaches to Natural Language Processing are attractive in that they offer a seamless transition from interface control to node content.

Nevertheless, one of the major drawbacks against Natural Language Processing is ambiguity, either structural or referential, which does not allow the fine level of granularity required by hypertext to explore the microlevel structure in individual sentences and relations among sentences. In addition, Artificial Intelligence Techniques deal predominately with the structure of the system and information at the node level and above.

In the next section, the thesis proposes a hybrid model, Hyperframes, that integrates Minsky's Frames with Hypertext's information nodes and links. The resulting model is presented as an alternative knowledge representation scheme that promises to resolve the shortcomings of Knowledge Based Tutoring Systems with respect to a full-scale didactic operation.

5.3 HYPERFRAMES: A KNOWLEDGE REPRESENTATION SCHEME THAT INTEGRATES MINSKY'S FRAMES WITH HYPERTEXT INFORMATION NODES AND LINKS

The basic knowledge unit in this knowledge representation scheme is the frame. A frame has attribute slots which either have default values, may be instantiated with specific occurrence values. They may also have procedural attachments which are executed whenever a value is needed or changed. The frame is stored in a "semistructured" hypertext information node. The semi-structured kind of node is chosen because of its ability to allow labelled fields and their values to be stored inside the node which is very similar to the frame attribute slots and values. The labelled fields in the semi-structured node will be the frame attribute slots and the labelled field values (when a frame is filled) will be the slot values. Therefore, a hypertext information node will represent a single frame. We call this unit of representation a ***HyperFrame***. See Figure 5.3, for an example of such a Hyperframe.



▨***Europe Frame***

▨***Specialisation-of:*** ▨**Continent**

▨***Part-of:*** ▨ **Continents**

▨***Countries:*** ▨**UK,** ▨**France,** ▨**Germany,** ▨**Netherlands,** ▨**Italy, ...**

▨***Size:*** ▨ **200SM**

▨***Seas:*** ▨ **Mediterranean,**▨ **Adriatic,**▨ **Aegian,**▨ **Baltic, ...**

▨***Mountains:*** ▨ **Alps,** ▨**Olympos,** ▨**Snowdonia, ...**

...

Figure 5.3: An example of a HyperFrame

The hyperframes is linked to other hyperframes with which they are related by a class-instance relationship. This establishes a semantic network of hyperframes which are organised hierarchically so that properties can be inherited from generic hyperframes (i.e. hyperframes higher in the hierarchy) to hyperframes lower in the hierarchy. Thus, a hyperframe which represents a concept will be decomposed in its hierarchical constituents, and allow these to inherit all its properties. A composite node may be used to aggregate related hyperframes. Links to other hyperframes with which a given hyperframe is hierarchically related in this hierarchical network, will be the slot values within this hyperframe. These links will be set up as "organisational" hypertext information links to connect a parent hyperframe with its children and thus establish a hierarchical tree in this hypertext network.

Hyperframes are also linked to other hyperframes with which they are not hierarchically related via "referential" hypertext information links and thus establish a non-hierarchical structure in this network. Any information related to a hyperframe which cannot be included in the hyperframe structure, will be "annotated" to the hyperframe as a "typed" hypertext information node, if it is text, or as a "graphical" hypertext information node, if it is an image, via an "annotation" link. This will establish a part-to-whole relationship with a given hyperframe. Within this annotation, there may be further referential, keyword or annotation links to hyperframes which the annotation may relate to. Finally, a hyperframe may be linked to another hyperframe by a "keyword" link, if two hyperframes have the same value for a given attribute slot. The link names will carry semantic information. Figure 5.4 below is an extract from a knowledge representation on the geography of planet Earth.

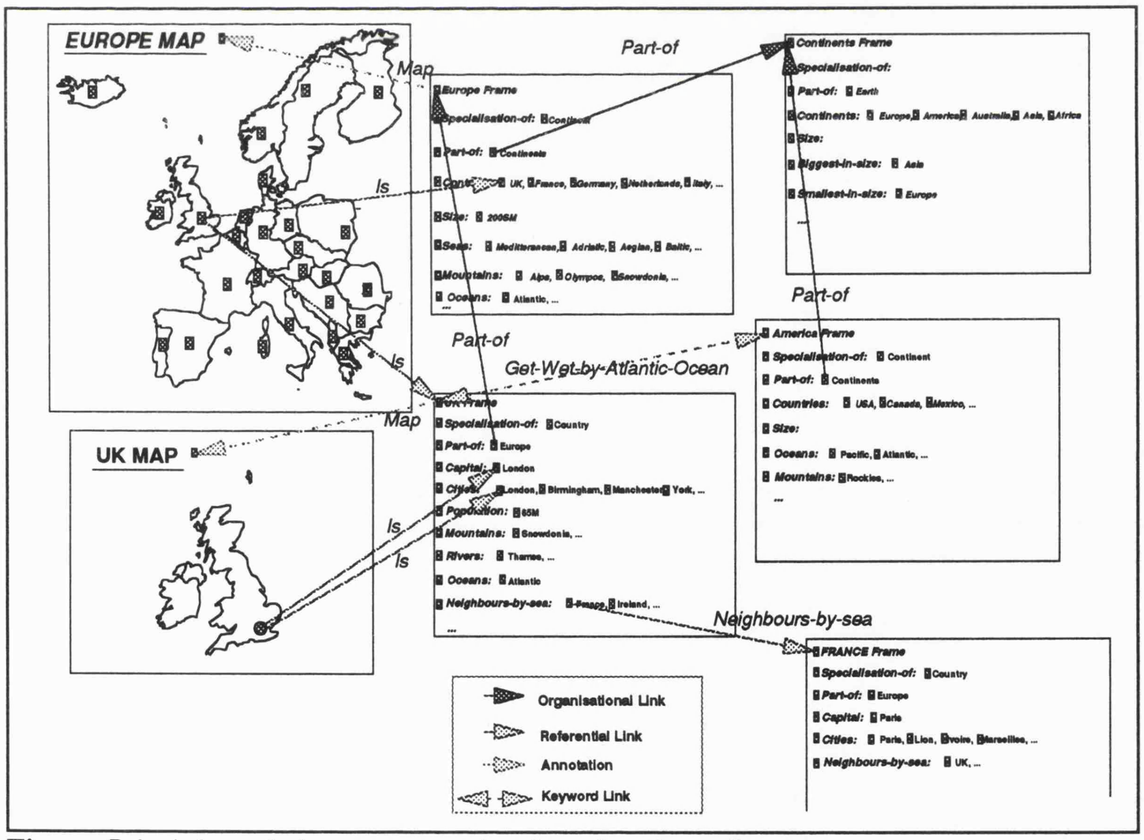

EUROPE MAP
UK MAP
Map
Is
Part-of
Get-Wet-by-Atlantic-Ocean
Neighbours-by-sea
Europe Frame
Specialisation-of: Continent
Part-of: Continents
Contains: UK, France, Germany, Netherlands, Italy, ...
Size: 200SM
Seas: Mediterranean, Adriatic, Aegian, Baltic, ...
Mountains: Alps, Olympos, Snowdonia, ...
Oceans: Atlantic, ...
Continents Frame
Specialisation-of:
Part-of: Earth
Continents: Europe, America, Australia, Asia, Africa
Size:
Biggest-in-size: Asia
Smallest-in-size: Europe
UK Frame
Specialisation-of: Country
Part-of: Europe
Capital: London
Cities: London, Birmingham, Manchester, York, ...
Population: 65M
Mountains: Snowdonia, ...
Rivers: Thames, ...
Oceans: Atlantic
Neighbours-by-sea: France, Ireland, ...
America Frame
Specialisation-of: Continent
Part-of: Continents
Countries: USA, Canada, Mexico, ...
Size:
Oceans: Pacific, Atlantic, ...
Mountains: Rockies, ...
FRANCE Frame
Specialisation-of: Country
Part-of: Europe
Capital: Paris
Cities: Paris, Lion, Lvoire, Marseilles, ...
Neighbours-by-sea: UK, ...
Organisational Link
Referential Link
Annotation
Keyword Link

Figure 5.4: A knowledge representation based on hyperframes

The five hyperframes in this knowledge representation segment utilise all four kinds of links presented above. First, there are organisational types of links that set up inheritance hierarchies, for instance, the *part-of* link from the UK frame to the EUROPE Frame, the *part-of* links from the EUROPE frame and the AMERICA frame to the CONTINENTS frame. Second, there are referential type of links that link hyperframes that are not hierarchically related with respect to the viewpoint with which hierarchical decomposition that resulted in the knowledge representation took place. An example of such a link is the *Neighbours-by-sea* link from the UK frame to the FRANCE frame. Third, there are keyword links between hyperframes that have the same value for a common attribute slot, for instance, the *Get-wet-by-Atlantic-Ocean* bi-directional link between the UK frame and the AMERICA frame.

Finally, there are two graphical hypertext information nodes annotated to two of the hyperframes via an annotation link called *map*: the EUROPE MAP is annotated to the EUROPE frame and the UK MAP is annotated to the UK frame. As was indicated before, from within an annotation there may be any type of links (except organisational links) to related hypertext information nodes. For instance, the *Is* referential link from that part of the EUROPE MAP annotation that portrays UK to the UK frame and the value UK in the Countries slot of the EUROPE frame. Similarly, the *Is* referential link from that part of the UK MAP that portrays London to the value London in the Capital slot and the Cities slot in the UK frame. If a hyperframe for London exists, then there would also be a referential link from the UK MAP to it.

The resulting knowledge representation system would allow three modes of reasoning: logical inferencing with some hypertext support, hypertext browsing with some support by automated procedures used with hyperframes, or a mixture of logical inferencing and hypertext browsing as described in section 5.2.5.

By being a declarative knowledge representation system, that is a frame-based system, it can then infer additional information for a given hyperframe by means of logical inferencing. Automated inferencing procedures can be applied to infer information from hyperframes the given hyperframe is hierarchically linked to. Thus this knowledge representation scheme can support inheritance and defaulting of slot values for a given hyperframe. In addition, the advantage of a frame-based system is that if hierarchical inferencing fails to produce any results, any procedural attachments to a given hyperframe may be executed. Once a value has been produced for the hyperframe slot, the hypertext engine is then called to create information links between the slot value and any related hyperframes. If there is a hyperframe describing this value, then if the value sets up an inheritance hierarchy with this retrieved hyperframe then an organisational hypertext information link is created between the slot value and this conceptually higher hyperframe.

If the value does not set up an inheritance hierarchy then a referential hypertext information link is created between the slot and the hyperframe. The hypertext engine will then generate a keyword or string search in order to find other hyperframes which include this value. The hypertext engine then creates keyword links between these values. Because of the inferential abilities of the frame based component, the hypertext engine can support computed links in addition to those statically determined by the author of the system. This removes the restriction of having to generate all the necessary links prior to interaction. Similarly, since a hyperframe may contain several default values for an attribute slot, this suggests that several instances of this hyperframe can be produced and be linked to this hyperframe. This will give the hypertext engine the ability to create computed nodes in which to store these frames and also create any organisational and referential links. These computed nodes will also be semistructured hypertext information nodes.

By being a hypertext system such a system, it can support hypertext information retrieval, that is browse through "selected" hypertext information nodes by following hypertext information links to these nodes and infer additional information from these nodes for a given hyperframe. The frame based component of the system can only infer additional information via inferencing with hierarchical links. It does not have the ability to explore non-hierarchical information links. The hypertext engine can follow both organisational and referential links from a given hyperframe and infer additional information either from hyperframes with which this hyperframe sets up an inheritance hierarchy or from hyperframes with which the hyperframe is linked to with referential links. In addition, the hypertext engine can follow keyword links to other hyperframes. Alternatively, the hypertext engine can issue a keyword search or, since the hypertext nodes are semi-structured, a database-like query.

Alternatively, the system may follow an inference mechanism that is partly based on hypertext browsing and partly based on logical inferencing. With this reasoning approach the hypertext engine establishes a path consisting of hyperframes related to a given hyperframe, by following organisational, referential and keyword links from this hyperframe. This path of hyperframes may be linked with a superlink or the nodes containing the hyperframes may be linked to a temporary composite node. Once the superlink path or the composite node is created then logical inferencing may commence in order to infer an additional value. If that fails to provide an answer then procedural attachments are instantiated with information from this set of related hyperframes and then executed.

5.3.1 Resolving the Limitations with Knowledge Based Tutoring Systems

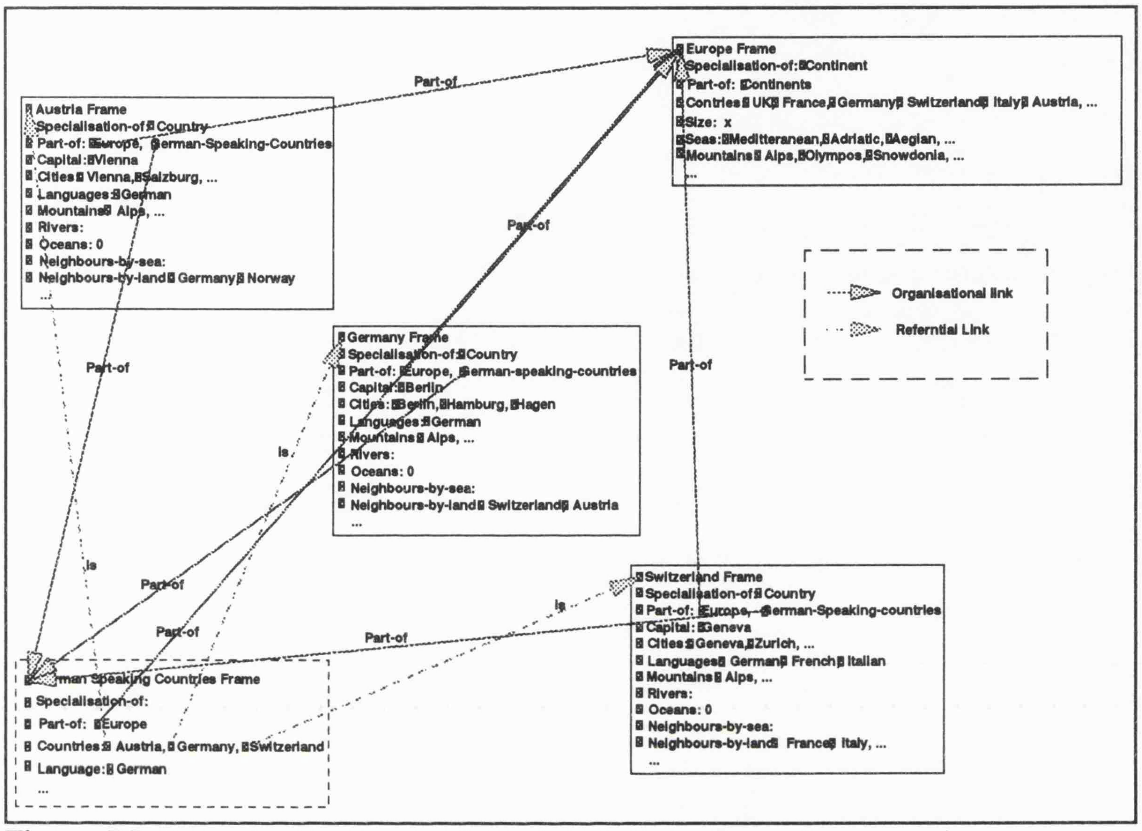
The knowledge representation scheme that results from linking Minsky's Frames with Hypertext Information nodes and links resolves the limitations of Knowledge Based Tutoring Systems with respect to a full-scale didactic operation.

The use of organisational, referential and keyword hypertext information links and annotations to link related hyperframes, as opposed to the hierarchical-only links and inferencing allowed by Minsky's frames, resolves the first shortcoming of Knowledge Based Tutoring Systems, that of the hierarchical-only knowledge decompositions, representations and inferencing with a domain knowledge base. Organisational links are used to set up the traditional inheritance hierarchies that are inherent in all knowledge representation schemes, but referential and keyword links as well as annotations are used to link non-hierarchically related frames. This has been demonstrated in Figure 5.4 above.

The use of hierarchical as well as non-hierarchical links between hyperframes facilitates, as explained in the previous section, different modes of reasoning and inferencing with this network of hyperframes, as opposed to the single and exclusively hierarchical mode of reasoning and inference which is used with traditional frames. Firstly, the use of organisational hypertext information links facilitate hierarchical inferencing with hyperframes. In this case, the automated procedures used with hyperframes establish inheritance hierarchies paths through the network of hyperframes, with the following of the links that set up these paths performed by the hypertext engine. Secondly, the use of hypertext information links facilitates browsing through the network of hyperframes in any fashion, either hierarchical or non-hierarchical. In this case, the hypertext engine establishes hierarchical and non-hierarchical pathways through the network of hyperframes and calls the automated procedures used with hyperframes to perform reasoning and inferencing along those paths. This second mode of reasoning and inference eliminates the need to perform one-way inference searchings through the entire tree or network every time the inferencing procedure has to establish a goal or infer a fact. The hypertext engine can simply follow links in the network of hyperframes to this goal or fact and thus establish a pathway without necessarily having to call automated procedures to perform inference. Thirdly, a mixture of logical inferencing and hypertext browsing can be used to deduce a fact from the network of hyperframes.

Because the knowledge representation scheme supports both computed links and nodes and consequently computed hyperframes, this resolves both the second and third limitations of Knowledge Based Tutoring Systems, namely that of the inference mechanism needing to have access to a complete knowledge representation of facts about the domain, and that of the system imposing a single viewpoint on its user. This "generative" behaviour (i.e. the ability to compute links and hyperframes) removes the need to have access to a complete knowledge representation of facts because any additional links or hyperframes can be generated during the course of interaction. This eases the restriction posed on the designer of the tutoring system to predict and prefabricate every single path the system or the user may follow during the course of interaction.

In addition, generative behaviour can sustain alternative viewpoints to the domain knowledge without having to reorganise the knowledge representation in a way that would involve breaking the hierarchical structure. Figure 5.5 below gives an example of generative behaviour from an alternative viewpoint.


Europe Frame
Specialisation-of: Continent
Part-of: Continents
Contries UK France, Germany Switzerland Italy Austria, ...
Size: x
Seas: Mediterranean, Adriatic, Aegian, ...
Mountains Alps, Olympos, Snowdonia, ...
...
Austria Frame
Specialisation-of Country
Part-of: Europe, German-Speaking-Countries
Capital: Vienna
Cities Vienna, Salzburg, ...
Languages German
Mountains Alps, ...
Rivers:
Oceans: 0
Neighbours-by-sea:
Neighbours-by-land Germany Norway
...
Germany Frame
Specialisation-of: Country
Part-of: Europe, German-speaking-countries
Capital: Berlin
Cities: Berlin, Hamburg, Hagen
Languages: German
Mountains Alps, ...
Rivers:
Oceans: 0
Neighbours-by-sea:
Neighbours-by-land Switzerland Austria
...
Switzerland Frame
Specialisation-of Country
Part-of: Europe, German-Speaking-countries
Capital: Geneva
Cities Geneva, Zurich, ...
Languages German French Italian
Mountains Alps, ...
Rivers:
Oceans: 0
Neighbours-by-sea:
Neighbours-by-land France Italy, ...
...
German Speaking Countries Frame
Specialisation-of:
Part-of: Europe
Countries Austria, Germany, Switzerland
Language: German
...
Part-of
Part-of
Part-of
Part-of
Part-of
Part-of
Part-of
Is
Is
Is
Organisational link
Referntial Link
Figure 5.5: Example of generative behaviour from an alternative viewpoint

In Figure 5.5, following a request either by the user or the system to find all European countries with German as one of their languages, the hypertext engine issues a search in all hyperframes to find which hyperframes for European Countries have "German" as part of their value in their language attribute slot. Once the hypertext engine has found these countries, Austria, Germany and Switzerland, automated procedures used with frames are called to construct a frame. The frame is called "German speaking Countries Frame" and several slots are created. An obvious slot is the "Countries" whose values are the names of the three countries. Another slot is a "language" slot whose value is "German". A "part-of" slot is created that would link this frame to the network of hyperframes. Since this is part of Europe, "Europe" is set as its value. If the three countries have other features in common, then additional attribute slots are created in this frame. The "part-of" slots in the hyperframes for the three countries are also updated to include "German-speaking-countries" as their value.

The hypertext engine would then store this frame in a semi-structured hypertext node and would then establish hypertext information links between this new hyperframe and any related hyperframes. Organisational links from the "German-speaking-countries" value in the "part-of" slot of the three hyperframes depicting Austria, Germany and Switzerland to the "German-speaking-countries" hyperframe are established. An organisational link from the "Europe" value in the "part-of" slot of this newly created hyperframe to the "Europe" hyperframe is also established. Referential links from each of the values in the "countries" slot of the new hyperframe to the hyperframes depicting these values are also established. Similarly, annotations, keyword links or any additional referential links from this new frame to any related frames are also established by the hypertext engine.

In addition to generative behaviour, Figure 5.5 also exemplifies the creation of a hyperframe from a different viewpoint and its linkage to existing hyperframes without having to reorganise the knowledge representation. The viewpoint with the original knowledge representation is that of physical borders between countries whereas the viewpoint with this new frame is that of language boundaries.

The use of hypertext information links resolves the fourth limitation of Knowledge Based Tutoring Systems, namely that of strictly and exclusively implicit information linking within the knowledge base. Hypertext information links are exclusively explicit. Therefore, as suggested in the previous section semantic information can be carried by names given to links and thus specific link names will set up a semantic hypertext network of nodes as in Figures 5.4 and 5.5. Explicit hypertext information links eliminate the need to perform reasoning in order to infer any relationships between knowledge representation parts which also eliminates the need to perform the same chain of reasoning every time a given relationship has to be established. For instance, in Figure 5.4 and Figure 5.5, hypertext information links serve the double purpose of establishing explicit information links between hyperframes and depicting the relationship between the hyperframes by giving the link a name that describes the relationship.

Chapter 5 introduced a hybrid model of Artificial Intelligence and Hypertext, Hyperframes, that integrates Minsky's Frames with Hypertext's information nodes and links. The use of Hyperframes overcomes the limitations of Knowledge Based Tutoring Systems with respect to the requirements for the development of a full-scale didactic operation.

Chapter 6 will show how to use hyperframes to design a generic model for the architecture of an Intelligent Tutoring System which is able to support a full-scale didactic operation. At first, the Chapter is concerned with the development of the Decision Base which entails the three necessary knowledge representations (i.e. domain, student and tutoring). In doing so, it pays particular attention to the interconnectedness of the three knowledge representations and the resulting generative ability of the system. Then the Chapter examines how the didactic operation would function with such a decision base in the context of a specific domain of discourse, by examining the resulting didactic plan of action, the pedagogical context of the didactic operation and the target level of the student model.

CHAPTER 6: USING HYPERFRAMES TO DESIGN A GENERIC MODEL FOR THE ARCHITECTURE OF AN INTELLIGENT KNOWLEDGE BASED TUTORING SYSTEM WITH A FULL-SCALE DIDACTIC OPERATION

This chapter proposes the use of hyperframes, introduced in chapter 5, for the design of a generic model for the architecture of an Intelligent Knowledge Based Tutoring System according to the requirements set out in chapter 4, that will support the full-scale didactic operation described in chapter 2. The architecture follows Wenger's model of an Intelligent Tutoring Systems architecture as described in Chapter 2 of the thesis.

The purpose of this model is not to promote a particular tutoring strategy nor to advocate a specific Intelligent Tutoring Systems design. The purpose is to offer an architecture that allows for a variety of system components, teaching styles, and intervention strategies to be combined into a single model. Within such a model, the Intelligent Tutoring System will reason about its own choice of intervention method; switch between teaching strategies to suit different student learning styles; use a variety of tactics and teaching approaches; make decisions about the most useful method for managing one-to-one tutoring; allow the student enough freedom to influence interaction by being able to modify the process of instruction should the need arise.

An Intelligent Tutoring System built with this model might be called a **Hybrid Guided-Discovery Generative Instructional Environment**. It will be hybrid because it would incorporate two different technologies, namely, Artificial Intelligence and Hypertext. By exploiting hypertext, the system can be a discovery learning environment, and since there is support from Artificial Intelligence techniques and the system can guide the student-user's learning, the system will be a guided-discovery system. Finally, since an attribute of the model is to exhibit generative behaviour, this instructional environment will a generative one.

The model will be explained in the context of the didactic operation for a specific domain of discourse. The domain of discourse chosen is that of the **geography of the planet Earth.** The goal of this system is to help the student-users review their knowledge on the geography of planet Earth in a context which can be as general as, for instance, geography of Europe as a whole or as specific as, for instance, mountains in Britain.

6.1 DEVELOPING THE DECISION BASE: DOMAIN EXPERT, STUDENT AND TUTOR KNOWLEDGE AND PROCESS MODELS

The first requirement states that a full-scale didactic operation must have access to three knowledge representations (i.e. domain, student and tutoring) and their corresponding process models:

[1]. It requires access to the domain knowledge which will both serve as a source for material sequencing and for providing the content for a tutorial intervention.

[2]. It requires access to the tutoring knowledge including a set of global goals that the system sets for the student to attain during the course of interaction, a set of intervention-specific goals that instruct the system what to do with the domain knowledge in the context of a tutorial intervention, and also a set of teaching strategies with which it will perform the tutorial intervention.

[3]. It requires access to the student knowledge which will both serve as a source for means-ends analysis of the student, that is to determine to which extent the student has met the global goals and how the student can be classified as an end-user (e.g. novice, advanced beginner, competent, etc.) and also as a diagnostic toolkit that would diagnose and record any missing concepts or misconceptions in the student's knowledge or behaviour and call for remedial action either to fill the gap created by missing concepts, or simply to remedy misconceptions.

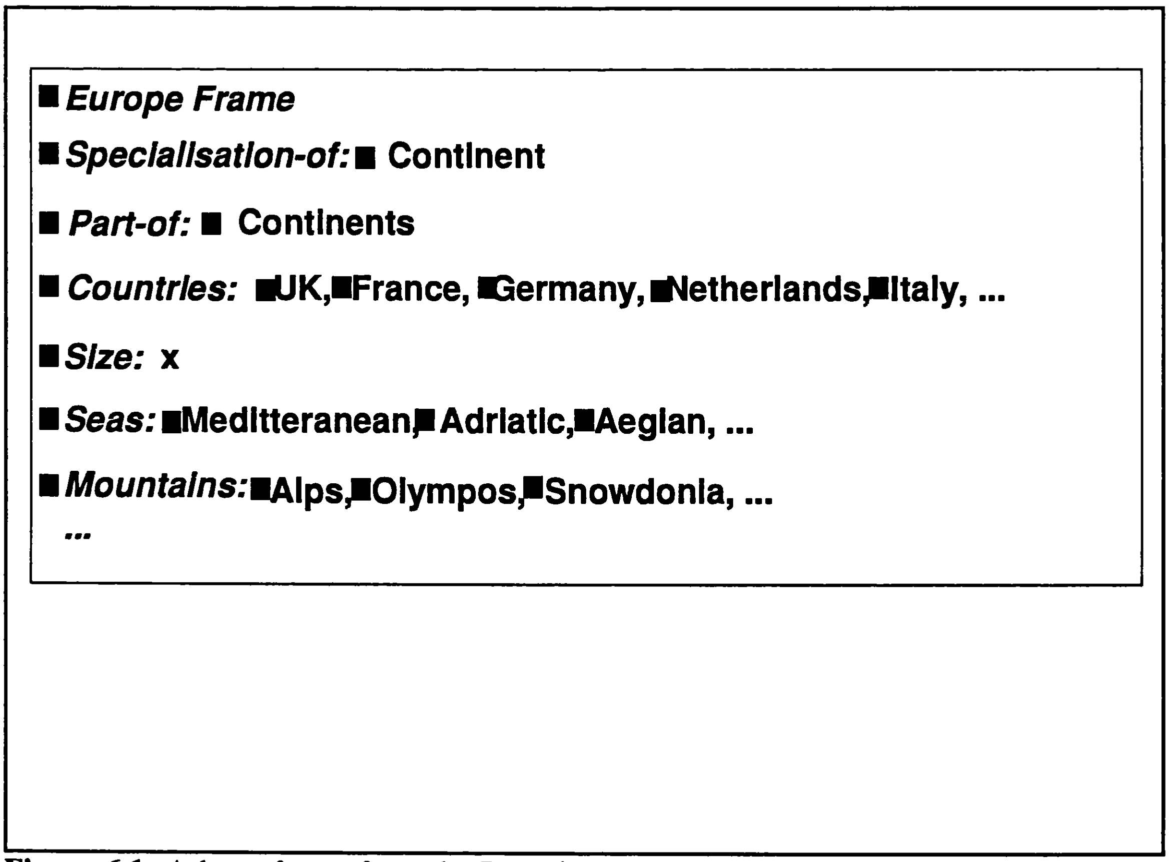
The second requirement states that although the use by the tutoring system's didactic operation of all three forms of knowledge suggests that these are interlinked, the three knowledge components are developed independently from each other. For instance, the expertise process model should be able to infer from the domain knowledge either a correct answer or be able to trace the solution path to a correct answer without any interference from any of the other process models or their knowledge. This suggests that the expertise process model should be able to act as a problem solver with its own knowledge. The diagnostics process model should be able to infer the student's current knowledge status and be able to call for remedial action. The didactics process model should be able to infer which is the best teaching strategy for attaining a goal, not which is the best for the current student-user.

In order to satisfy the first two requirements, the design of the decision base proceeds on the basis that all three kinds of knowledge are kept separately as three distinct 'knowledge representations'.

6.1.1 The Domain Expert Knowledge Model

The first task in developing the domain model is to decide on the different global goals the tutoring system will try and help the student achieve with its domain knowledge. The next task is to acquire and organise domain knowledge around these goals. Organising the domain knowledge around these goals involves decomposing the domain knowledge, from a default viewpoint, into different hierarchical and non-hierarchical **knowledge units** whose level of domain detail depends on their position in the hierarchical structure.

Each knowledge unit in this hierarchical decomposition will be represented using a hyperframe. The context of these hyperframes is exclusively domain knowledge. It contains neither any knowledge about the student or what to do with this knowledge (i.e. tutoring knowledge). The domain hyperframe in Figure 6.1 is an example of such a knowledge unit.



■ ***Europe Frame***

■ ***Specialisation-of:*** ■ **Continent**

■ ***Part-of:*** ■ **Continents**

■ ***Countries:*** **■UK,■France, ■Germany, ■Netherlands,■Italy, ...**

■ ***Size:*** **x**

■ ***Seas:*** **■Meditteranean,■ Adriatic,■Aegian, ...**

■ ***Mountains:*** **■Alps,■Olympos,■Snowdonia, ...**

...

Figure 6.1: A hyperframe from the Domain Expert Knowledge

A hyperframe may be linked with any other hyperframe via an organisational hypertext information link, if there is a hierarchical relationship between the two hyperframes, via a referential hypertext information link, if the two hyperframes are non-hierarchically related under the current viewpoint, via a keyword hypertext information link if they share the same attribute, and finally via an annotation, if additional information about a hyperframe cannot be included in its context, for example, graphs. By being explicit, a typed hypertext information link carries a name which designates the relationship between the two hyperframes it links.

The resulting network of hyperframes can express relationships between topics such as prerequisites, corequisities. It is important to note that the network is declarative (i.e. it contains a structured space of concepts) but does not assume any particular order for traversal of this space. Figure 6.2 is a portion from the Domain Knowledge representation of an Intelligent Tutoring System on the geography of planet Earth.

In Figure 6.2, the organisational links "part-of" and "specialisation-of" set up hierarchical relationships, for instance, the UK Frame to the EUROPE Frame and the COUNTRY Frame to the CONTINENT Frame. The referential link "is" sets up non-hierarchical relationships, for instance, between the EUROPE Frame and the SNOWDONIA Frame. The bi-directional link "Oceans:Atlantic" between the UK Frame and the CANADA Frame is an example of a keyword link between the two countries that denotes that the two countries although they are in different parts of the hierarchy they share the same attribute slot value.

As with domain expert knowledge, both tutoring and student knowledge will also be

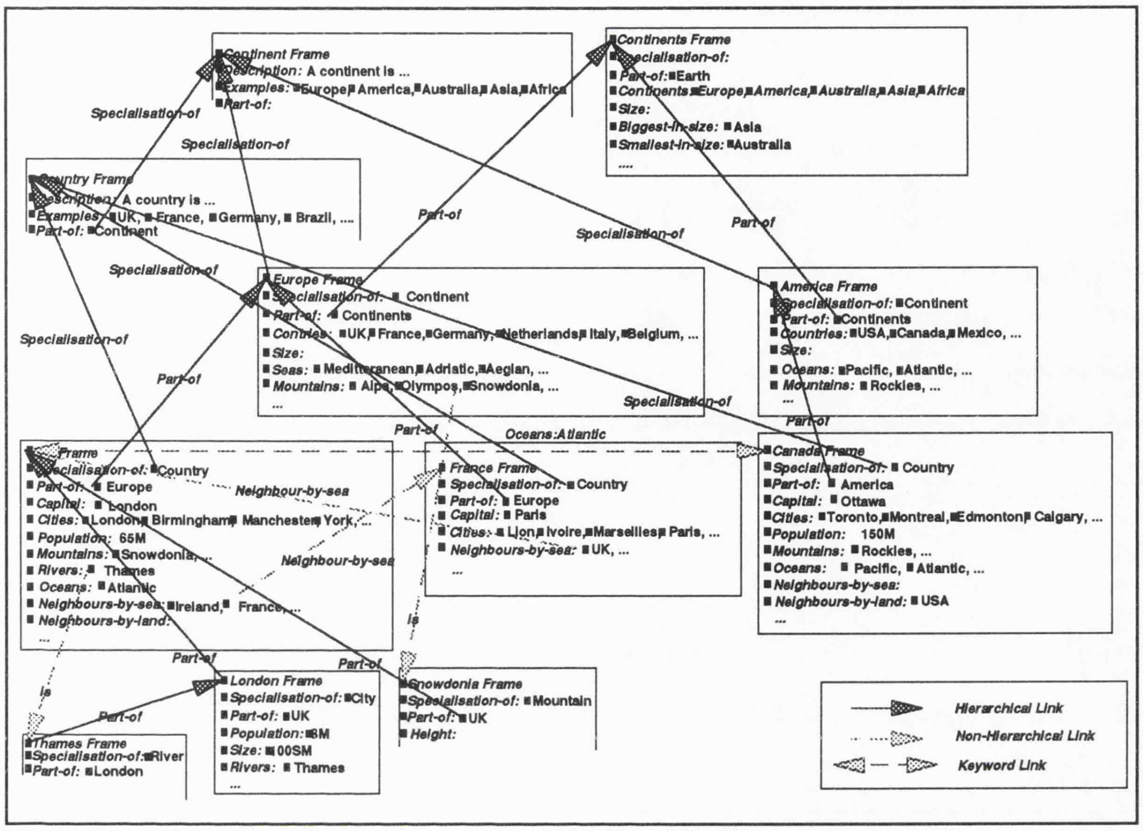


Figure 6.2: Representation of a portion of the Domain Knowledge

represented using hyperframes. Although these are kept in two separate knowledge representations and hence during the course of interaction form their own hierarchical structures, the contents of either a student or a tutoring hyperframe is determined by the context of a corresponding domain knowledge hyperframe.

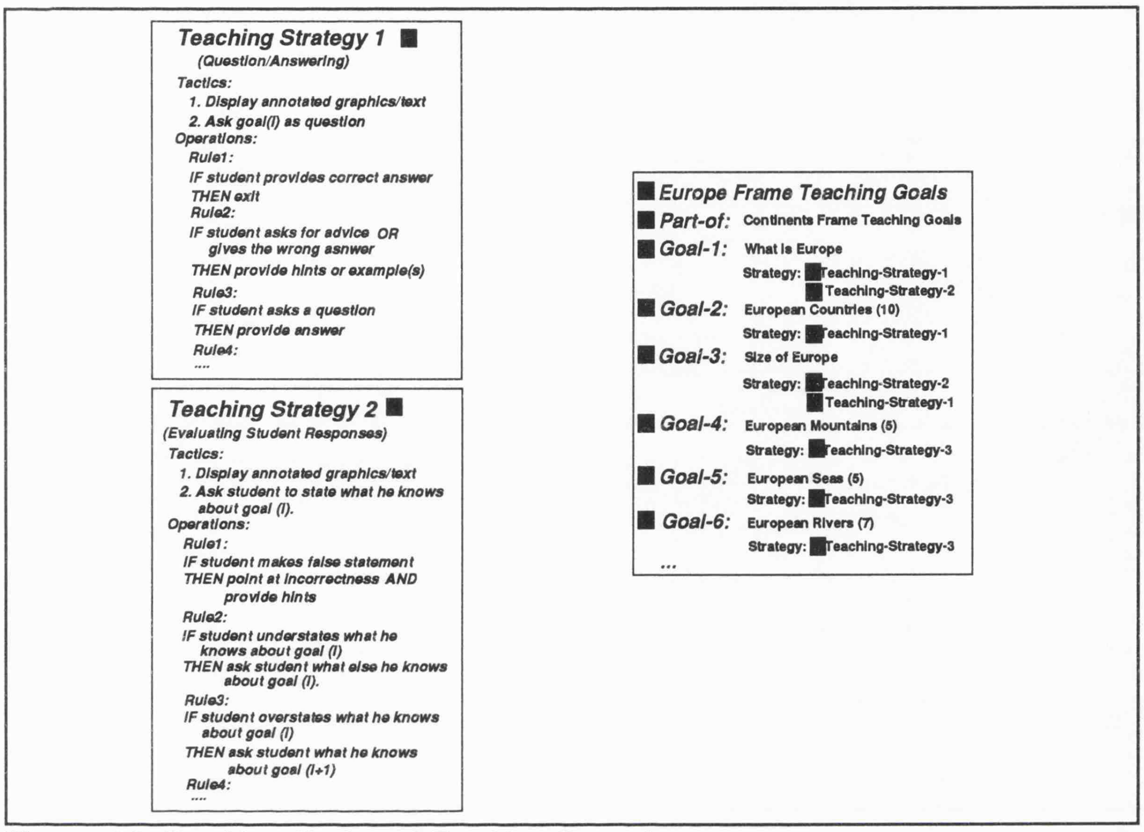
6.1.2 The Tutoring Knowledge Model

Tutoring knowledge comprises a set of global goals underlying system development, that the system will try and help the student to attain, a set of local teaching goals for each and every domain expert hyperframe, and a pool of teaching strategies for tutoring with a domain expert hyperframe. The global goals that the system sets for the student-user are a set of conditions for terminating interaction with the system. These are universal goals and are not included in the tutoring knowledge representation. What is kept in the tutoring knowledge representation are the sets of teaching goals and the teaching strategies.

Teaching goals are stored in hyperframes. A teaching goals hyperframe contains attribute slots whose values are the local goals which the system will try to attain during the course of interaction with a corresponding domain expert hyperframe. All hyperframe attribute slots also contain the names of those teaching strategies that are suitable for attaining the goal. Which of these teaching strategies will be applied is decided either by reference to the student model or by the user, during the course of interaction.

A teaching strategy is a rule-based implementation of a particular teaching strategy used by human teachers, for example, coaching, questions/answering, evaluation of student responses, etc. This rule-based implementation contains tutoring knowledge about material presentation, for formulating tasks/responses to the student-user, for student evaluation and for remedial action. This is the local mechanism that provides tutoring with the contents of a domain expert hyperframe. This rule-based implementation of a teaching strategy is stored in a "typed" hypertext information node. A "typed" hypertext information node is neither part of any form of hierarchy nor does it contain any hypertext information links to any other hypertext information nodes. Figure 6.3 illustrates a teaching goals hyperframe and two "typed" hypertext information nodes each containing extracts from a rule-based implementation of a teaching strategy.

As with the domain expert model, a teaching goals hyperframe is linked with organisational hypertext information links to other teaching goals hyperframes. This sets up a hierarchical structure of teaching goals hyperframes as in Figure 6.4 below. Since a teaching goals hyperframe is designated for a specific domain expert hyperframe, there can be no referential or keyword hypertext information links from a teaching goals hyperframe to another. However, there are annotations from each and every attribute slot



```
Teaching Strategy 1 ■
  (Question/Answering)
 Tactics:
   1. Display annotated graphics/text
   2. Ask goal(I) as question
 Operations:
   Rule1:
   IF student provides correct answer
   THEN exit
   Rule2:
   IF student asks for advice  OR
      gives the wrong asnwer
   THEN provide hints or example(s)
   Rule3:
   IF student asks a question
   THEN provide answer
   Rule4:
   ....
```

```
Teaching Strategy 2 ■
(Evaluating Student Responses)
 Tactics:
   1. Display annotated graphics/text
   2. Ask student to state what he knows
      about goal (I).
 Operations:
   Rule1:
   IF student makes false statement
   THEN point at incorrectness AND
          provide hints
   Rule2:
   IF student understates what he
      knows about goal (I)
   THEN ask student what else he knows
          about goal (I).
   Rule3:
   IF student overstates what he knows
      about goal (I)
   THEN ask student what he knows
          about goal (I+1)
   Rule4:
   ....
```

```
■ Europe Frame Teaching Goals
■ Part-of:  Continents Frame Teaching Goals
■ Goal-1:   What is Europe
            Strategy: ■Teaching-Strategy-1
                      ■Teaching-Strategy-2
■ Goal-2:   European Countries (10)
            Strategy: ■Teaching-Strategy-1
■ Goal-3:   Size of Europe
            Strategy: ■Teaching-Strategy-2
                      ■Teaching-Strategy-1
■ Goal-4:   European Mountains (5)
            Strategy: ■Teaching-Strategy-3
■ Goal-5:   European Seas (5)
            Strategy: ■Teaching-Strategy-3
■ Goal-6:   European Rivers (7)
            Strategy: ■Teaching-Strategy-3
  ...
```

Figure 6.3: Teaching Goals and Teaching Strategies

(i.e. a goal) to all teaching strategies that are suitable for attaining the goal denoted by the slot as illustrated in Figure 6.4 below. A specific teaching strategy can be used by hyperframes which are at different hierarchical levels. As stated above, teaching strategies are not part of any hierarchical structure.

6.1.3 The Student Knowledge Model

The student model, unlike the domain expert model and the tutor model, is constructed during the course of interaction as an overlay model of the domain expert model, including diagnosed misconceptions. For each and every domain expert hyperframe that the system uses for tutoring, a corresponding student hyperframe is constructed, the contexts of which are determined by the context of the domain expert hyperframe. The attribute slots of the student hyperframe are a copy of the attribute slots of the corresponding domain expert hyperframe with the inclusion of some additional attribute

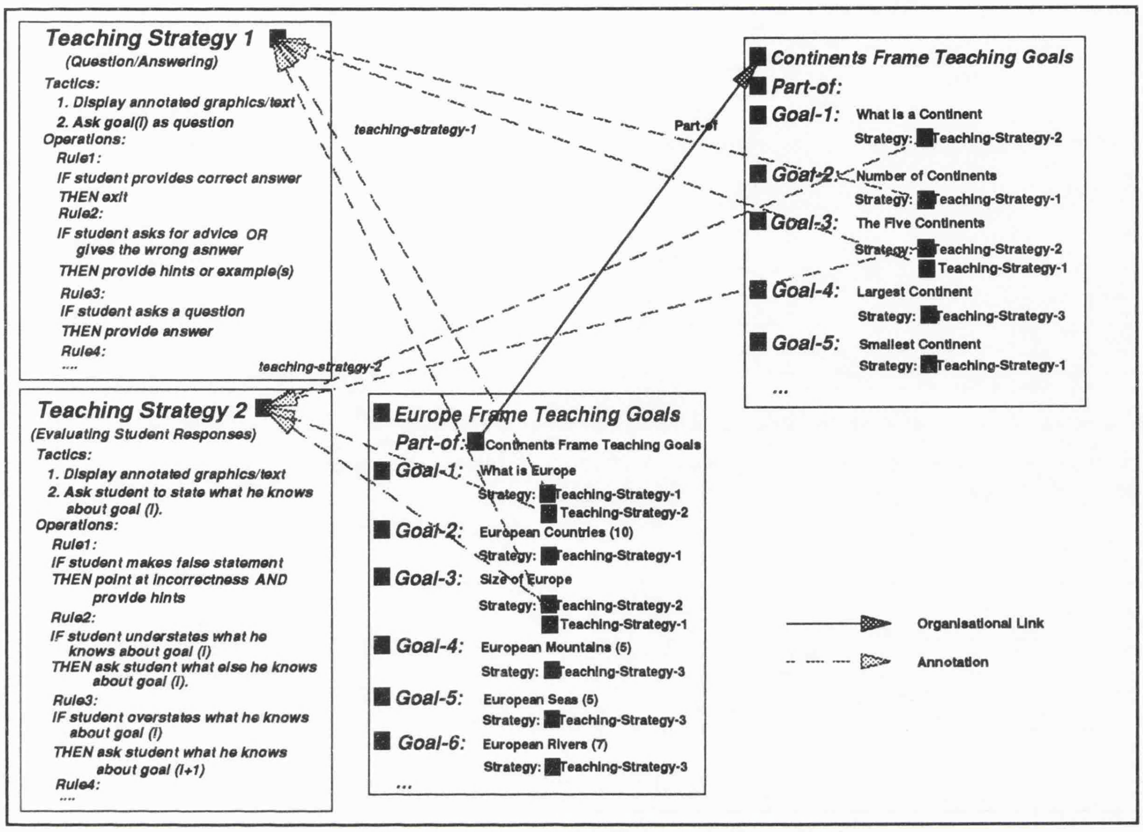
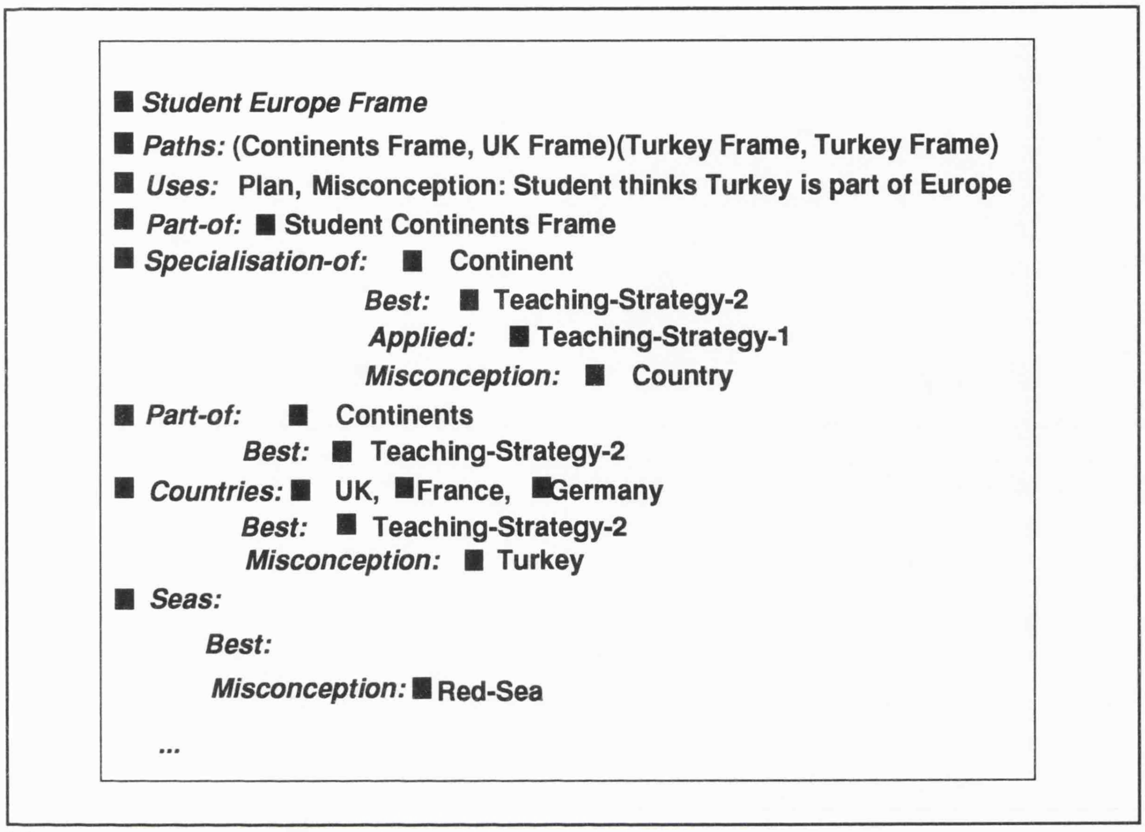

Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(i) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
....
teaching-strategy-1
teaching-strategy-2
Part-of
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
1. Display annotated graphics/text
2. Ask student to state what he knows about goal (i).
Operations:
Rule1:
IF student makes false statement
THEN point at incorrectness AND provide hints
Rule2:
IF student understates what he knows about goal (i)
THEN ask student what else he knows about goal (i).
Rule3:
IF student overstates what he knows about goal (i)
THEN ask student what he knows about goal (i+1)
Rule4:
....
Europe Frame Teaching Goals
Part-of: Continents Frame Teaching Goals
Goal-1: What is Europe
Strategy: Teaching-Strategy-1
Teaching-Strategy-2
Goal-2: European Countries (10)
Strategy: Teaching-Strategy-1
Goal-3: Size of Europe
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
Goal-4: European Mountains (5)
Strategy: Teaching-Strategy-3
Goal-5: European Seas (5)
Strategy: Teaching-Strategy-3
Goal-6: European Rivers (7)
Strategy: Teaching-Strategy-3
...
Continents Frame Teaching Goals
Part-of:
Goal-1: What is a Continent
Strategy: Teaching-Strategy-2
Goal-2: Number of Continents
Strategy: Teaching-Strategy-1
Goal-3: The Five Continents
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
Goal-4: Largest Continent
Strategy: Teaching-Strategy-3
Goal-5: Smallest Continent
Strategy: Teaching-Strategy-1
...
Organisational Link
Annotation

Figure 6.4: Representation of a portion of the Tutoring Knowledge

slots to indicate the different paths that lead to the domain expert hyperframe (e.g. as part of the regular didactic plan of action or as part of a remedial action), how the domain expert hyperframe was used (e.g. to clarify the content of an attribute slot), and various teaching strategies that have been successfully or unsuccessfully applied with the domain expert hyperframe. The values in the attribute slots of the student hyperframe are the student input obtained during the course of interaction. Figure 6.5 represents the student Europe hyperframe for the corresponding domain expert hyperframe.

Since student hyperframes are constructed during the course of the tutoring process, any resulting hypertext information links are computed at the same time. Nevertheless, the end result will be, as with the domain expert model and the tutoring model, a network of student hyperframes that includes the student model's hierarchical structure.



■ *Student Europe Frame*

■ *Paths:* (Continents Frame, UK Frame)(Turkey Frame, Turkey Frame)

■ *Uses:* Plan, Misconception: Student thinks Turkey is part of Europe

■ *Part-of:* ■ Student Continents Frame

■ *Specialisation-of:* ■ Continent

Best: ■ Teaching-Strategy-2

Applied: ■ Teaching-Strategy-1

Misconception: ■ Country

■ *Part-of:* ■ Continents

Best: ■ Teaching-Strategy-2

■ *Countries:* ■ UK, ■France, ■Germany

Best: ■ Teaching-Strategy-2

Misconception: ■ Turkey

■ *Seas:*

Best:

Misconception: ■ Red-Sea

...

Figure 6.5: A student hyperframe

A student hyperframe may be linked with organisational hypertext information links to other student hyperframes. These organisational hypertext information links set up a hierarchical structure of student hyperframes. These links, however, have an overlay statistic attached to the name which they carry that indicates the level of mastery of the concept relationship implied by the link. For instance, a negative value suggests that the student does not comprehend the concept implied by the link, a positive value suggests that some degree of mastery has been achieved by the student and a 0 value suggests that no attempt has been made by the student to perceive the concept implied by the link. These links, in addition to the hierarchical structure which they delineate, also define the overlay model of the student-user. The numerical figure is a standard yardstick of measurement in overlay models. There may be referential or keyword hypertext information links drawn from one student hyperframe to another, if the student-user establishes a non-hierarchical relationship between two domain expert hyperframes or if two domain expert hyperframes share the same attribute slot value. As with organisational links, overlay statistics designate the level of mastery of the concept implied by the link.

With every attribute slot there may be a set of associated misconceptions. Every time the student-user gives an answer which is not recognised by the system as the correct one, the system checks through the Bugs Library to see if the student answer is a known bug. See Figure 6.6 for an extract from a Bugs Library.

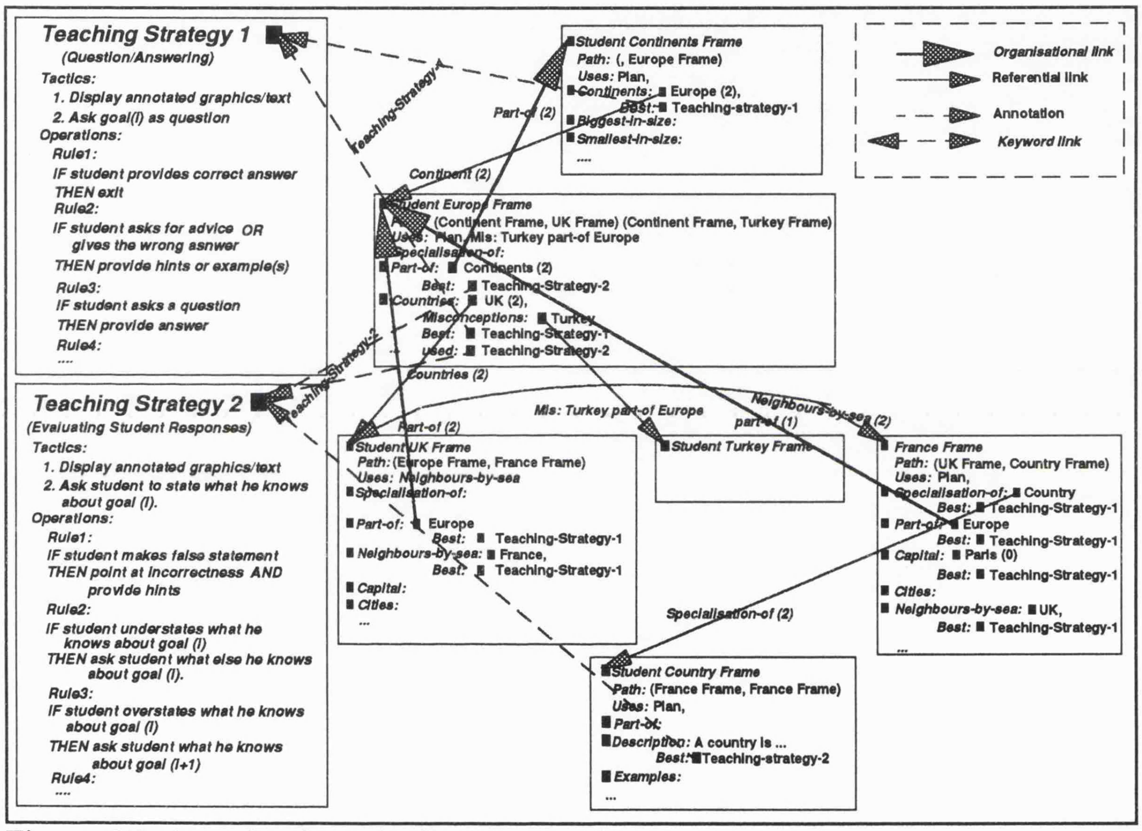
```
....
Mal-Rule 34: student thinks (? continent) is a country
IF
specialisation-of (? continent) = continent
and student reply specialisation-of (? continent) = country
THEN
student thinks that (? continent) is a country

....
Mal-Rule 58: student thinks (? country) part of (? Continent)
IF
part-of (? country) <> (? Continent)
and student reply part-of (? country) = (? Continent)
THEN
student thinks that (? country) part of (? continent)
....
```

Figure 6.6: Extracts from the Bugs Library

If this is the case, the system would insert the student answer in the attribute slot as a student misconception and create a referential link from it to the student hyperframe that describes the misconception. The name of the rule that proves that the answer is a known misconception is set as the name of the referential link.

Finally, from each and every attribute slot that has been filled with a value (i.e. the student has attempted to acquire the knowledge contained in the corresponding domain expert hyperframe attribute slot and thus tried to attain the goal described by the corresponding teaching goals hyperframe attribute slot) there are annotations to the best teaching strategy for acquiring the knowledge and achieving the goal and also to those teaching strategies that have been tried unsuccessfully. Figure 6.7 below represents a portion from the student model.



Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(I) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
....
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
1. Display annotated graphics/text
2. Ask student to state what he knows about goal (I).
Operations:
Rule1:
IF student makes false statement
THEN point at incorrectness AND provide hints
Rule2:
IF student understates what he knows about goal (I)
THEN ask student what else he knows about goal (I).
Rule3:
IF student overstates what he knows about goal (I)
THEN ask student what he knows about goal (I+1)
Rule4:
....
Student Continents Frame
Path: (, Europe Frame)
Uses: Plan,
Continents: Europe (2),
Best: Teaching-strategy-1
Biggest-in-size:
Smallest-in-size:
....
Organisational link
Referential link
Annotation
Keyword link
Teaching-Strategy-1
Part-of (2)
Continent (2)
Student Europe Frame
Path: (Continent Frame, UK Frame) (Continent Frame, Turkey Frame)
Uses: Plan, Mis: Turkey part-of Europe
Specialisation-of:
Part-of: Continents (2)
Best: Teaching-Strategy-2
Countries: UK (2),
Misconceptions: Turkey
Best: Teaching-Strategy-1
used: Teaching-Strategy-2
Countries (2)
Teaching-Strategy-2
Mis: Turkey part-of Europe
Neighbours-by-sea (2)
part-of (1)
Part-of (2)
Student UK Frame
Path: (Europe Frame, France Frame)
Uses: Neighbours-by-sea
Specialisation-of:
Part-of: Europe
Best: Teaching-Strategy-1
Neighbours-by-sea: France,
Best: Teaching-Strategy-1
Capital:
Cities:
...
Student Turkey Frame
France Frame
Path: (UK Frame, Country Frame)
Uses: Plan,
Specialisation-of: Country
Best: Teaching-Strategy-1
Part-of: Europe
Best: Teaching-Strategy-1
Capital: Paris (0)
Best: Teaching-Strategy-1
Cities:
Neighbours-by-sea: UK,
Best: Teaching-Strategy-1
...
Specialisation-of (2)
Student Country Frame
Path: (France Frame, France Frame)
Uses: Plan,
Part-of:
Description: A country is ...
Best: Teaching-strategy-2
Examples:
...

Figure 6.7: A portion from the Student Model

The "part-of" and "specialisation-of" links set up the hierarchical structure in the student model network, for instance, the "part-of" link from the Student Europe Frame to the Student Continents Frame and the "specialisation-of" link from the Student France Frame to the Student Country Frame. The "Continent" and "Country" are two examples of referential links that link hyperframes which are not hierarchically related under the current viewpoint. The link names also carry a numerical figure which sets up the student overlay model. A special referential link is one whose name is a diagnosed student misconception, for example, the "Turkey part-of Europe" link. The name of the link is the name of the rule that proves the misconception. This link connects the incorrect student answer (i.e. the misconception) from the attribute slot in which it was raised to the Frame whose name is the same as the student answer, for instance, from the "Countries" slot in the Student Europe Frame to the Student Turkey Frame. In addition, to the referential links there are keyword links that link two frames that either have the same value for the same attribute slot, or for a given attribute slot they contain each other's name, for instance, the UK Frame is linked to the France Frame via a keyword link called "Neighbours-by-sea". The name of the link is the name of the attribute slot. Finally, there are "annotated" teaching strategies for each and every attribute slot that is filled with a value as a result of a student answer.

6.1.4 Knowledge Models Interconnectedness

The second requirement for the development of a full-scale didactic operation states that although the three knowledge components are developed independently, they must, nevertheless, work in synergy. Although student and tutoring knowledge form their own network data structure, inclusive of a hierarchical tree structure, the contents of both of these knowledge structures are determined by the domain knowledge: associated with each and every domain knowledge hyperframe there is a corresponding student knowledge hyperframe as a local overlay student model of the domain expert knowledge hyperframe that registers acquired student knowledge exhibited by the student. There is also a teaching goals hyperframe that designates the use of the knowledge in the domain hyperframe, and there is a set of teaching strategies and finally there may be annotated hypertext information nodes containing information that cannot be included in any of the three semi-structured hypertext information nodes, for instance, graphs.

This interconnectedness of the three knowledge models as stated by the second requirement assumes explicit and direct links between related parts of the three knowledge representations (i.e. the three hyperframes), the teaching strategies and any annotated hypertext information nodes containing, in addition to the links that have already been used to set up the three networks independently. These links will have either been statically determined by the knowledge models designer or once dynamically computed by the system during the course of interaction will be maintained thereafter. As explained in chapter 4, these links will also help to avoid mixing of the three forms of knowledge and also help a knowledge process model to access information from other knowledge models without having to distinguish between knowledge. All these links will be set up as referential hypertext information links.

With respect to tutoring knowledge, every teaching goal which the system will try to attain with a particular domain hyperframe, should be linked to those attribute slots of the domain knowledge hyperframe. It may not necessarily be a one-to-one correspondence. In those cases, where a domain knowledge attribute slot holds several values, the corresponding teaching goal has a number attached to it that indicates the number of values the student must get right before moving on to the next goal. For example, in Figure 6.3, the student must name at least 10 European countries, 7 European Rivers, 5 European mountains and 5 European seas. Also, each and every goal should point to a set of teaching strategies that are deemed appropriate for tutoring with the goal. It should also include links to any annotated hypertext information nodes that contain additional information about the goal that cannot be included in the domain hyperframe. All these links will need to have been set up by the knowledge models designer.

With respect to student knowledge, all the links will be computed, as before, during the course of interaction. From each and every attribute slot that has been filled with a value, there should be a link to the corresponding attribute slot in the domain hyperframe containing this knowledge. This would provide an explicit indication of the current knowledge state of the student and also an "orthogonal" projection of the student overlay model onto the domain knowledge. The names of these links also carry a measurement of the mastery of the knowledge contained in the attribute slot of the student hyperframe. Then, from each and every attribute slot that has been filled with a value, there should also be a link to the teaching goal that is satisfied by filling the attribute slot in the student hyperframe. This link should be bidirectional to enable the system to check in the teaching goals hyperframe which teaching goals have been met and thus allow issue tutoring for those that are yet to be satisfied, or if all have been satisfied, progress on to the next domain hyperframe. Finally, there should be a link to the teaching strategy that proved to be the best for a student-user for meeting a goal, along with links to those teaching strategies that have been applied with the goal.

The resulting system is a large collection of **instruction knowledge units** which hold specific local domain knowledge, student diagnostic knowledge and teaching goals. Associated with each instruction unit will be a set of domain-independent teaching strategies provided as a set of general rules, for tutoring with the unit's knowledge. Associated teaching strategies, for instance, coaching, question/answering, etc., that are used to tutor with the unit's knowledge, are represented as production rules and are triggered through an expert system. Every instruction knowledge unit has access to the bugs library of common bugs or misconceptions in the field. These are also represented as production rules. Figure 6.8 is an example of such an instruction knowledge unit. The unit in this case is about Europe as a continent.

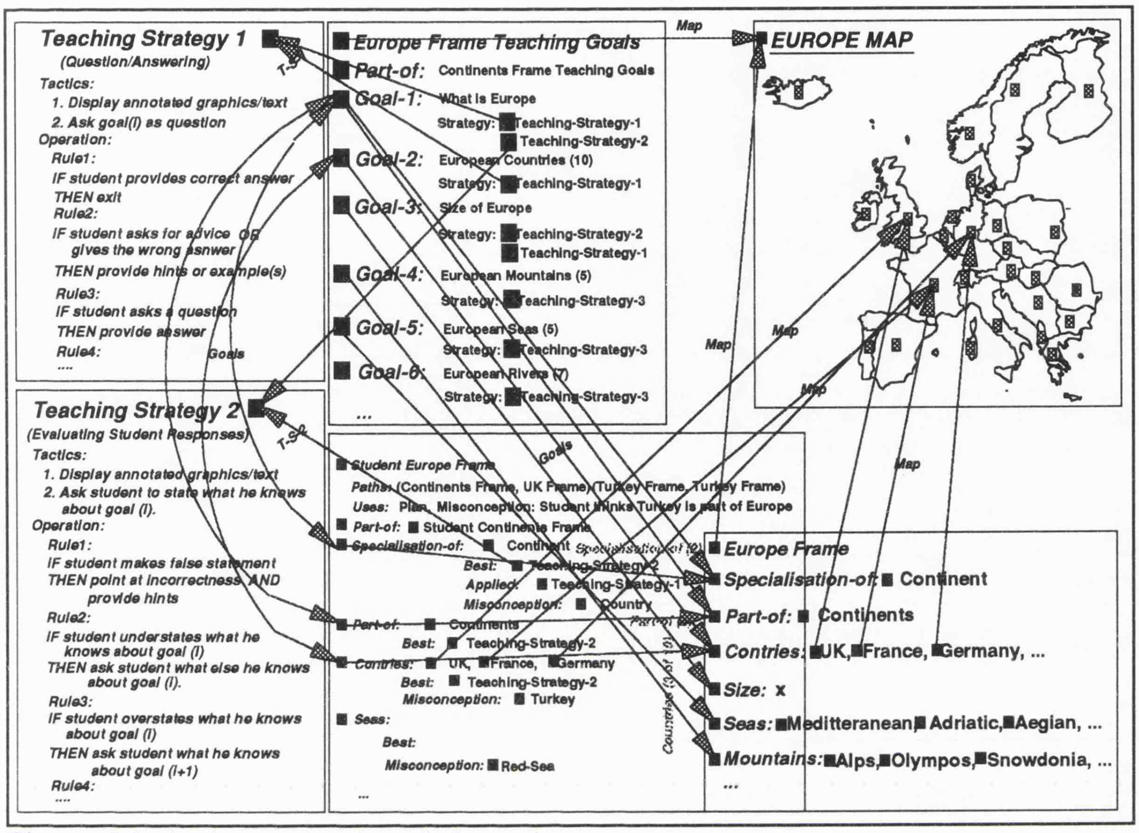

Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(i) as question
Operation:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR gives the wrong answer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
Goals
T-S
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
1. Display annotated graphics/text
2. Ask student to state what he knows about goal (i).
Operation:
Rule1:
IF student makes false statement
THEN point at incorrectness AND provide hints
Rule2:
IF student understates what he knows about goal (i)
THEN ask student what else he knows about goal (i).
Rule3:
IF student overstates what he knows about goal (i)
THEN ask student what he knows about goal (i+1)
Rule4:
Europe Frame Teaching Goals
Part-of: Continents Frame Teaching Goals
Goal-1: What is Europe
Strategy: Teaching-Strategy-1
Teaching-Strategy-2
Goal-2: European Countries (10)
Strategy: Teaching-Strategy-1
Goal-3: Size of Europe
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
Goal-4: European Mountains (5)
Strategy: Teaching-Strategy-3
Goal-5: European Seas (5)
Strategy: Teaching-Strategy-3
Goal-6: European Rivers (7)
Strategy: Teaching-Strategy-3
...
Map
EUROPE MAP
Map
Map
Map
Map
Student Europe Frame
Paths: (Continents Frame, UK Frame)(Turkey Frame, Turkey Frame)
Uses: Plan, Misconception: Student thinks Turkey is part of Europe
Part-of: Student Continents Frame
Specialisation-of: Continent
Best: Teaching-Strategy-2
Applied: Teaching-Strategy-1
Misconception: Country
Part-of: Continents
Best: Teaching-Strategy-2
Contries: UK, France, Germany
Best: Teaching-Strategy-2
Misconception: Turkey
Seas:
Best:
Misconception: Red-Sea
...
Goals
Specialisation-of (2)
Part-of (2)
Countries (3 of 10)
Europe Frame
Specialisation-of: Continent
Part-of: Continents
Contries: UK, France, Germany, ...
Size: x
Seas: Mediterranean, Adriatic, Aegian, ...
Mountains: Alps, Olympos, Snowdonia, ...
...

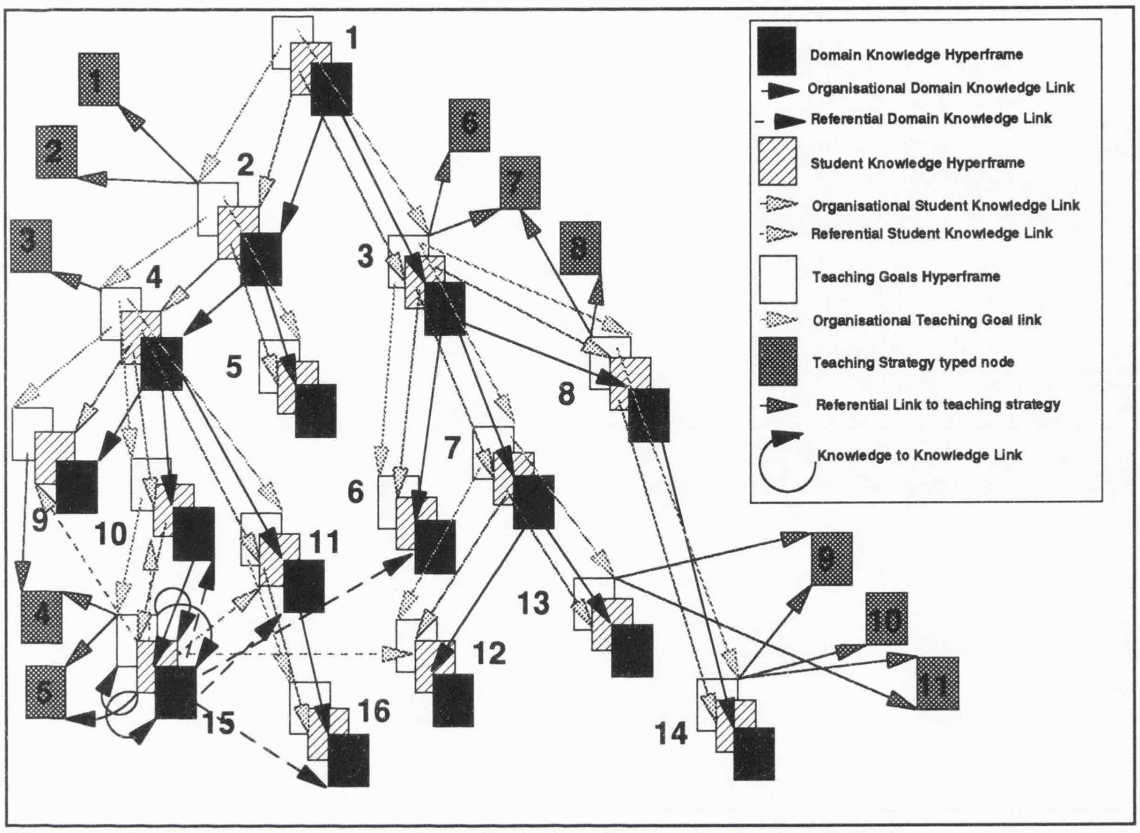
Figure 6.8: An instruction knowledge unit

Every teaching goal in the teaching goals hyperframe is linked to those attribute slots of the domain knowledge hyperframe that contain relevant domain knowledge, for instance, Goal-1 is linked to the "Specialisation-of" and "Part-of" slots, Goal-2 is linked to the "Countries" slot, etc. In the case of Goal-1, where the corresponding domain knowledge attribute slot has several values, there is the number 10 associated with the goal, which designates that to satisfy this goal the student must name at least 10 European countries. Consequently, this number overwrites the overlay statistic which in this case, is not any longer between -2 and 2 but a number out of 10. From within each and every goal slot there are links to teaching strategies that are appropriate for tutoring with the goal. The teaching goals hyperframe in our case has a link to an annotated typed hypertext information node that contains the map of Europe. Finally, there are bidirectional links to those slots in the student hyperframe that have been filled with values. This would enable the system to check which goals still remain to be satisfied with the current domain knowledge hyperframe.

From each and every attribute slot in the student hyperframe that has been filled with a value, there is a link to the corresponding attribute slot in the domain hyperframe containing this knowledge, for instance, the "countries" link to the domain hyperframe. The name of the link also carries an overlay statistic that indicates the mastery of the concept by the student. This link is necessary for overlay modelling. Also from each and every attribute slot filled with a value, there is a link to the teaching goal that is satisfied, for example, the link between Goal-1 and the "Specialisation-of" and "Part-of" slots. Finally, there are links, first, to the best teaching strategy for filling an attribute slot and, second, to those teaching strategies that have been applied with a goal.

Figure 6.9 illustrates how all the different data structures for the system would look when they are linked with hypertext information links. The instruction knowledge unit 15 in this Figure is an example of a unit with its explicit hypertext information links between the different knowledge parts of the same unit (i.e. the student knowledge, the domain knowledge, and the tutoring knowledge), and where each knowledge hyperframe of the unit is explicitly linked hierarchically and non-hierarchically, as is the case with domain and student knowledge hyperframes, to other hyperframes higher or lower in its hierarchy.

With each instruction knowledge unit there is an associated **substance node** which contains all the user tools such as such graphical browsers, graphical slots to be filled with annotated graphical information nodes and hypertext icons that act as buttons to generate tutoring actions with the instruction knowledge unit.



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Domain Knowledge Hyperframe
Organisational Domain Knowledge Link
Referential Domain Knowledge Link
Student Knowledge Hyperframe
Organisational Student Knowledge Link
Referential Student Knowledge Link
Teaching Goals Hyperframe
Organisational Teaching Goal link
Teaching Strategy typed node
Referential Link to teaching strategy
Knowledge to Knowledge Link

Figure 6.9: A hypertext view of the Tutoring System's data structure

As explained before, the system is not intended to be coupled to any particular design methodology and consequently may be developed in a variety of forms, each tailored to the specific needs or interests of its users. The hypertext nodes may be implemented as hypercards which can be used to encode either the domain knowledge hyperframes, the student knowledge frames, the teaching goals frames, the production rules of the various teaching strategies, or the bug library. The hypertext links may be implemented as **link icons** that link **component cards** together by **subcomponent links** in a hierarchical structure or in a non-hierarchical structure. The icon links represent links to other cards which the system may either suggest to the student to follow up or pull them up for the student in order to attain a certain goal. Although the user will eventually be given enough freedom to follow links, the system provides an implicit default structure which will try to take the student-user through.

6.1.5 Generative Behaviour

The third requirement for the development of a full-scale didactic operation states that in addition to hierarchical knowledge decompositions in the three knowledge structures, that would allow for a variety of hierarchical paths to be followed through, there will also be non-hierarchical explicit and direct paths between different parts of the same knowledge structure that would enable the system to follow as a result of some breakdown in the tutoring process, for instance, the need to pursue a remedial path.

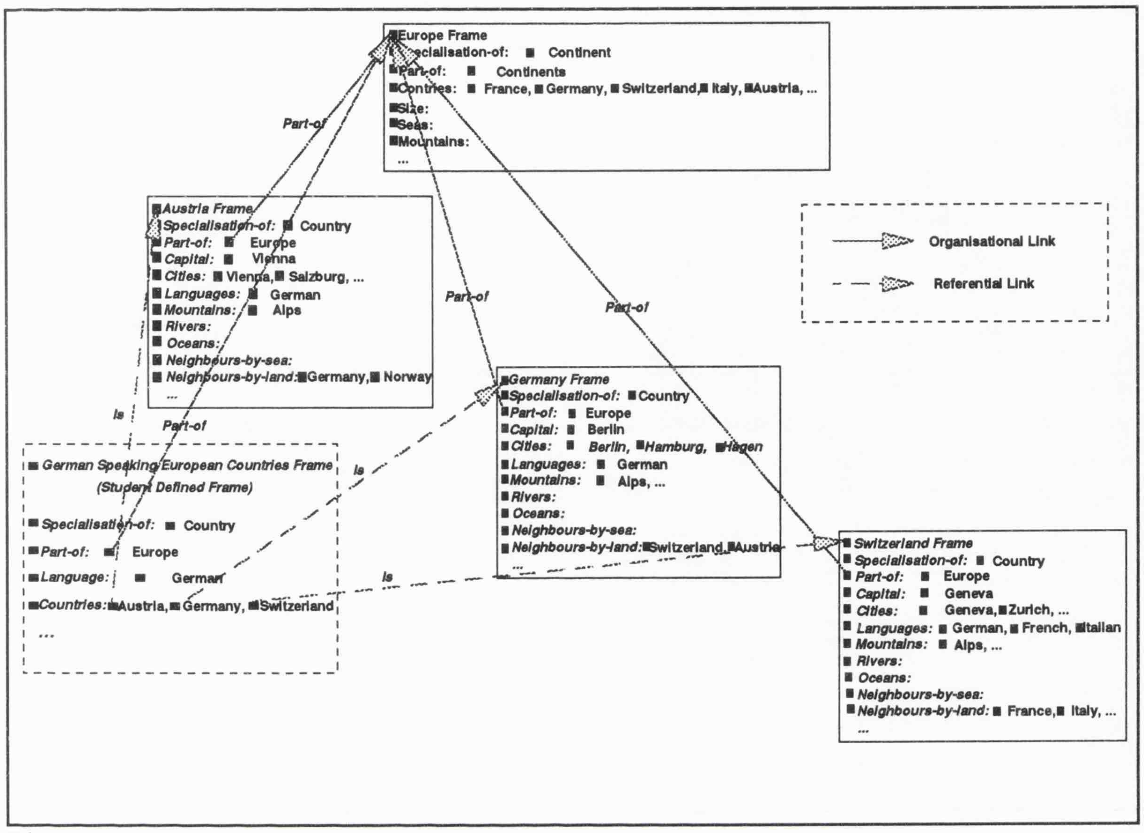
This third requirement states that these paths will either be statically preset by the instructional designer or dynamically computed by the system. With respect to the former, the tutoring system's data structure supports both hierarchical and non-hierarchical explicit paths through the network. This is achieved through the use of organisational information links for hierarchical paths and referential and keyword information links and annotations for non-hierarchical paths. Hypertext information links are by default explicit. A hypertext information link between two nodes has to be explicitly established either by the user or computed by the hypertext engine. Hypertext information links may be made visually explicit to the user with the use of graphical browsers or link icons.

The latter (i.e. links computed and established by the system during the course of interaction), imposes a fourth requirement for the development of a full-scale didactic operation: following a student request, or as a result of the outcome of the student diagnostic process model the tutoring system must pursue hypertext information links in its three knowledge representations, in order to generate either alternative paths or additional domain knowledge from a different viewpoint. This is necessary because of the wide range of outcomes from the student diagnostic process model, all of which cannot be precisely anticipated by the instructional designer, because the tutoring system may not have access to a complete domain knowledge representation and because information is required from a different viewpoint. A tutoring system with generative behaviour would result in the instructional designer not having to anticipate all possible paths a user or the tutoring system may follow during interaction, a domain knowledge structure that may not necessarily be complete and also provide alternative viewpoints on the domain knowledge.

This last requirement stresses the need for explicit and direct links to other knowledge parts anywhere within a knowledge representation and within the tutoring system's overall data structure (see Figure 6.15) which would otherwise need to be inferred by the system. The tutoring system when generating additional domain knowledge would link these to its existing knowledge representations for further use.

The system can generate additional domain knowledge hyperframes from its domain knowledge representation during the course of interaction, either if such a need arises or as a response to a student-user request. In either case, the system is in control of the generating process so it can tailor the area of emphasis to suit the individual student. Figure 6.10 is an example of such generative behaviour.

In the example in Figure 6.10, the student-user asked the system for all the German-Speaking European Countries and the system responded by searching its network of Domain knowledge hyperframes for countries whose languages attribute slot includes German, produced a "German Speaking European Countries" hyperframe and placed Austria, Germany, and Switzerland as values in the "countries" attribute slot in the hyperframe. The syntax and semantics of the hyperframe are determined by the system


Europe Frame
Specialisation-of: ■ Continent
Part-of: ■ Continents
Contries: ■ France, ■ Germany, ■ Switzerland, ■ Italy, ■ Austria, ...
Size:
Seas:
Mountains:
...
Austria Frame
Specialisation-of: ■ Country
Part-of: ■ Europe
Capital: ■ Vienna
Cities: ■ Vienna, ■ Salzburg, ...
Languages: ■ German
Mountains: ■ Alps
Rivers:
Oceans:
Neighbours-by-sea:
Neighbours-by-land: ■ Germany, ■ Norway
...
Organisational Link
Referential Link
Part-of
Part-of
Part-of
Part-of
Is
Is
Is
Germany Frame
Specialisation-of: ■ Country
Part-of: ■ Europe
Capital: ■ Berlin
Cities: ■ Berlin, ■ Hamburg, ■ Hagen
Languages: ■ German
Mountains: ■ Alps, ...
Rivers:
Oceans:
Neighbours-by-sea:
Neighbours-by-land: ■ Switzerland, ■ Austria
...
German Speaking European Countries Frame
(Student Defined Frame)
Specialisation-of: ■ Country
Part-of: ■ Europe
Language: ■ German
Countries: ■ Austria, ■ Germany, ■ Switzerland
...
Switzerland Frame
Specialisation-of: ■ Country
Part-of: ■ Europe
Capital: ■ Geneva
Cities: ■ Geneva, ■ Zurich, ...
Languages: ■ German, ■ French, ■ Italian
Mountains: ■ Alps, ...
Rivers:
Oceans:
Neighbours-by-sea:
Neighbours-by-land: ■ France, ■ Italy, ...
...
Figure 6.10: Example of Generative Behaviour

in accordance with the rest of the hyperframes.

The system will also generate the local student hyperframe corresponding to the domain knowledge hyperframe it creates and also the teaching goals hyperframe for this domain hyperframe in which to include teaching strategies by searching for the best overall teaching strategy for the student-user in the whole of the student model. The same diagnostic routines applied with the rest of the hyperframes can now be applied with this generated hyperframe.

The system's generative behaviour has many advantages. First, it increases the range of issues on which the system can offer tutoring. Second, it solves partly the problem of designing additional necessary instructional material during the course of interaction. Third, it eliminates the need to prestore all possible hyperframe combinations, especially those arising from different viewpoints. Fourth, it has virtually unlimited resource material, as much as the system can produce combinations of. Finally, it allows the system and the student-user to generate as many hyperframes as needed in order to attain the educational objectives of the system.

With respect to the last advantage, the system may use its generative abilities to generate tasks for the student, as part of its testing of the student-user's understanding of the subject material after the system has completed tutoring with a group of instruction knowledge units, for instance, the Europe Branch on the domain tree. Or it may use the hypertext engine to issue a search for all those European countries whose first language is German, and ask the student to name them. Of course, the difficulty of the task needs to be directly related to the classification of the student-user.

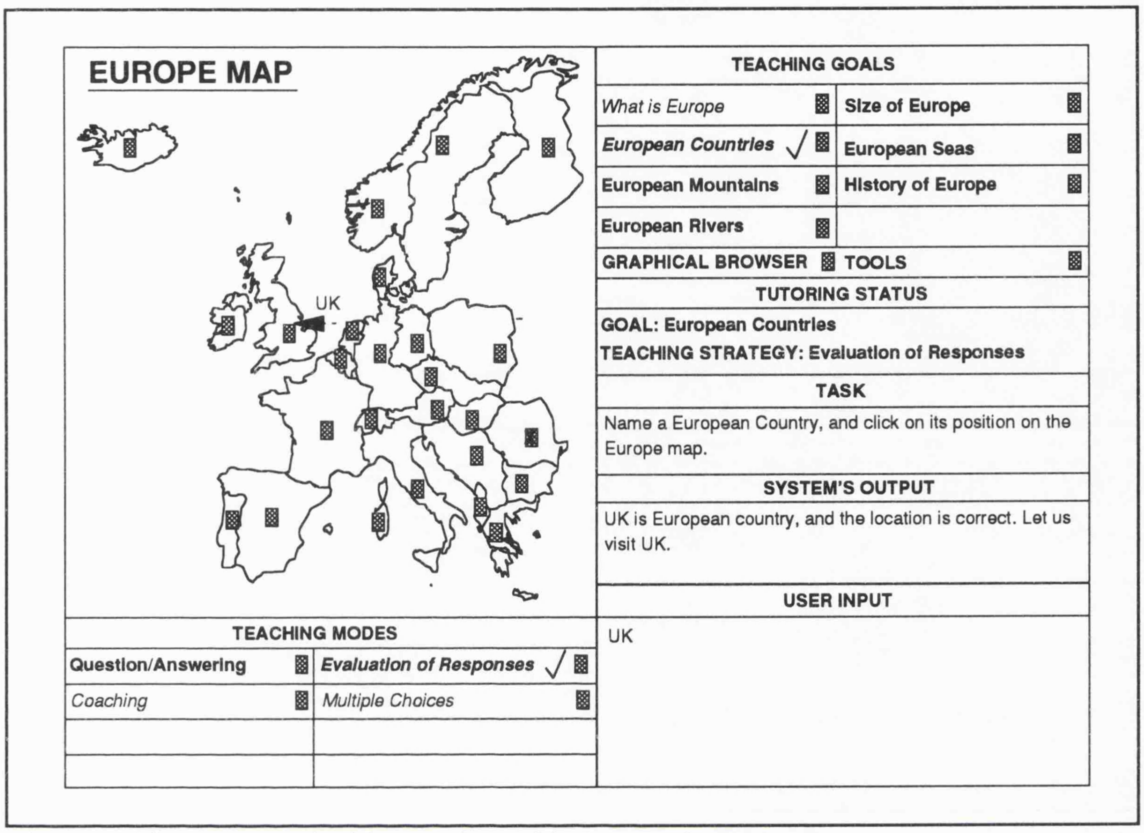
6.1.6 User Interface

During instruction delivery, the system takes the student-user through different sequences of instruction knowledge units where the different paths underlie the structure of the domain knowledge structure. In addition, the system allows experienced users to drive their way through the system. Which instruction knowledge unit or group of instruction knowledge units to visit next is determined by the pedagogical process model.

With every instruction knowledge unit that the user visits or is taken to, the user communicates with the tutoring system via a user interface which is a combination of a Hypertext Interface (HI), a Graphical User Interface (GUI) and a Restricted Natural Language Processor (RNLP). The system provides a **graphical browser** whose purpose is to collect hyperframes specified by the user, then link these hyperframes to a source hyperframe and finally display this network graphically. Hyperframes are normally represented as boxes and hypertext links as directed lines between these boxes. Hypertext browsers usually come with a set of editing tools that allow the user to rapidly access, modify, and extend the depicted network structure. The system has a **searcher** whose purpose is to find and collect hyperframes meeting certain conjunctive or disjunctive specifications of field information. The system has a **collection tool** whose purpose is to look for hyperframes of a specific type that emanate certain kinds of hypertext information links. **Cluster tools** identify similar sets of hyperframes according to some metric provided by the user. The **link follower** traverses a network along links of a specified type, displaying (if requested) encountered hyperframes in the order that have been encountered and allowing the user to select among choices where the network branches.

The user interface performs a set of management tasks: it displays "annotated" hyperframe text or graphics from the domain hyperframe and places any link icons in their place, all goals the user has to attain during interaction within the unit, all available teaching strategies, the teaching goal selected by the tutor process model, the teaching strategy selected by the tutor model with which to attain the goal, creates the relevant graphical browser to and from the domain hyperframe of the instruction knowledge unit, and prompts the user for input. Figures 6.11 and 6.12 depict protocols of interaction with a student-user.

Although the system is following a certain plan of action, within an instruction knowledge unit the student-user is allowed, always with the approval of the system, to change the mode of interaction: switch between different teaching strategies, change to a different



TEACHING GOALS

What is Europe ▨	**Size of Europe** ▨
European Countries ✓ ▨	**European Seas** ▨
European Mountains ▨	**History of Europe** ▨
European Rivers ▨	
GRAPHICAL BROWSER ▨	**TOOLS** ▨

TUTORING STATUS

GOAL: European Countries
TEACHING STRATEGY: Evaluation of Responses

TASK

Name a European Country, and click on its position on the Europe map.

SYSTEM'S OUTPUT

UK is European country, and the location is correct. Let us visit UK.

USER INPUT

UK

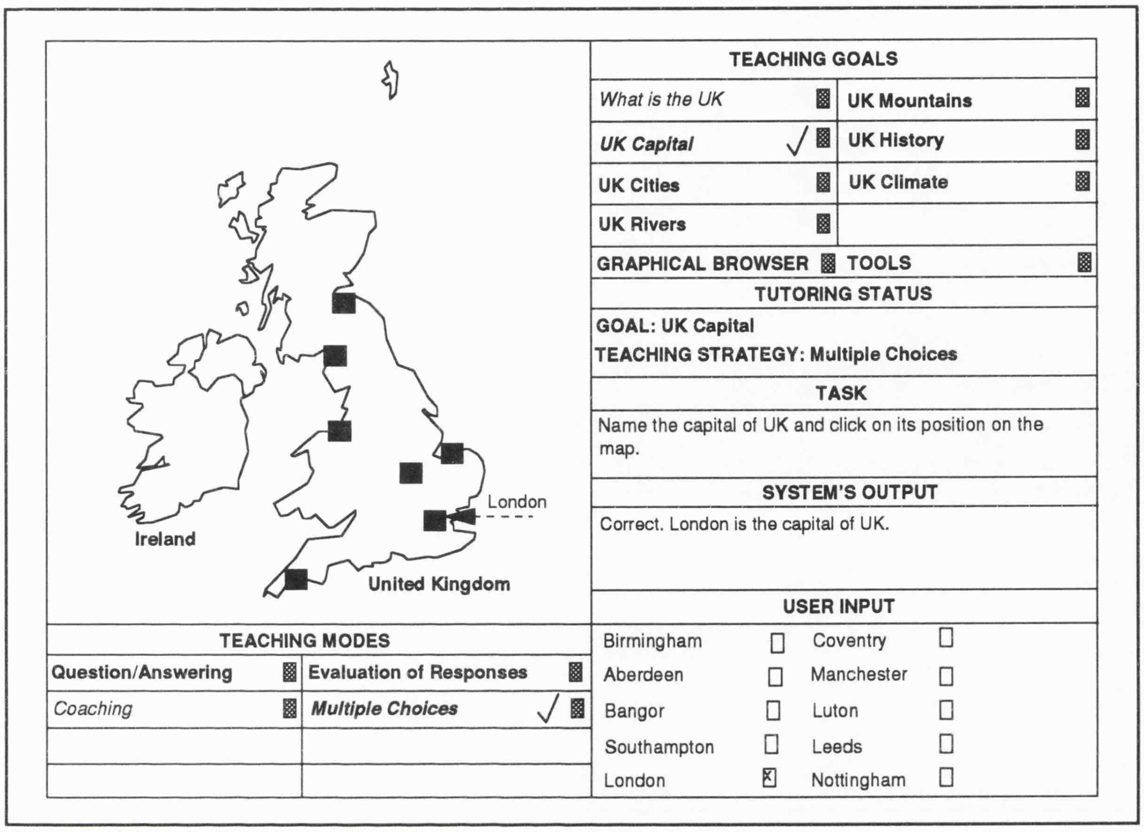
TEACHING MODES

Question/Answering ▨	***Evaluation of Responses*** ✓ ▨
Coaching ▨	*Multiple Choices* ▨

Figure 6.11: A protocol of interaction with the tutoring system

teaching goal, visit another unit with the help of the graphical browser, retrieve information using the searcher and follow this information, follow hypertext information links using the link follower, use the RNLP to pose a task or question to the system, generate a hyperframe.

The more experienced a user becomes (i.e. the range of missing conceptions in their overlay model is increasingly reduced, misconceptions have been cleared out and the student-user has been exposed to most of the material preset by the instructional designer), the more freedom the system allows to the student to explore this environment. That is when the student may choose to test the system's generative behaviour and discover any hidden curriculums. Obviously, the student-user does not have to terminate the interaction with the system after the system has taken him through its entire material. The student may continue exploring, in which case the system takes a more silent and passive role.



TEACHING GOALS	
What is the UK ▨	UK Mountains ▨
UK Capital ✓ ▨	UK History ▨
UK Cities ▨	UK Climate ▨
UK Rivers ▨	
GRAPHICAL BROWSER ▨	TOOLS ▨

TUTORING STATUS

GOAL: UK Capital
TEACHING STRATEGY: Multiple Choices

TASK

Name the capital of UK and click on its position on the map.

SYSTEM'S OUTPUT

Correct. London is the capital of UK.

USER INPUT

Birmingham	☐	Coventry	☐
Aberdeen	☐	Manchester	☐
Bangor	☐	Luton	☐
Southampton	☐	Leeds	☐
London	☒	Nottingham	☐

TEACHING MODES	
Question/Answering ▨	Evaluation of Responses ▨
Coaching ▨	***Multiple Choices*** ✓ ▨

Figure 6.12: A Protocol of interaction with the tutoring system

6.1.7 The Domain Knowledge Process Model: Expertise

The inherent hierarchy of the domain knowledge model does not designate the relative importance of the tutorial topics but is merely one way of ordering domain knowledge from a particular viewpoint, in our case that of physical boundaries (i.e. country borders). A different viewpoint may result in a different top-level domain hyperframe. At the global level, the domain knowledge process model is responsible for material sequencing, that is for retrieving the next domain hyperframe. For example, if the system is providing tutoring with the instruction knowledge unit on Europe and the system will proceed to provide tutoring on the UK, then the process model is responsible for accessing the UK hyperframe for this purpose.

At the local level (i.e. within the context of an instruction knowledge unit), the process model is responsible for the content of tutorial interventions. For example, if within the context of the instruction knowledge unit on the UK, the system is trying to satisfy the local teaching goal, "Capital of the UK" then the domain process model is responsible for retrieving the correct answer for this question.

6.1.8 The Student Knowledge Process Model: Diagnostics

The student knowledge model depicts the relative strengths (e.g. with topics, with teaching strategies, etc.) and weaknesses of the student-user (e.g. misconceptions and missing concepts). At the global level, the student knowledge process model is responsible for providing a means-ends analysis of the user. This involves examining the student overlay model for missing concepts and for misconceptions in order to classify the user-learner, for instance, as a novice, or advance beginner or competent, etc. The resulting classification serves as an alternative terminating condition and also provides a yardstick for designating the decree of freedom to explore that will be granted to the student-user.

At the local level, the process model is responsible for integrating acquired student knowledge in the corresponding student hyperframe by filling the attribute slots with values, best and applied teaching strategies, the tutoring path and the reason that lead to this domain hyperframe (e.g. didactic plan, remedial plan). In addition to merely integrating information in the student hyperframe, the process model is also responsible for diagnosing misconceptions and thus signalling appropriate remedial action. For example, if the student gave Turkey as a reply to the query "Name ten European countries", then the process model may instruct the system to provide tutoring on Turkey or on Asia, in order to clear away the misconception. Misconceptions are permanently recorded in the student hyperframe of the instruction knowledge unit in which they occur.

The reason for this, is that a series of related diagnosed misconceptions may result from missing concepts and thus the process model may signal additional remedial action. For instance, if the student repeatedly gives the names of African countries as Asian countries then the process model may instruct the system to provide remedial action by providing tutoring on Africa or Asia, or diagnose which missing concepts (if any) in the student Africa or Asia hyperframe may be the cause of these misconceptions. The process model assesses the effectiveness of the user of applying a particular teaching strategy with a specific teaching goal against a set of independent set of criteria, for instance 2s scored, time taken to reply, chosen by the student, etc.

Diagnosis of misconceptions is performed as follows. When the student provides an answer to a question, then the process model checks whether this is an acceptable value by looking in the set of values in the corresponding domain hyperframe attribute slot. If it is, then the process model adds the user input in the corresponding student hyperframe attribute slot. If not then the process model checks this value against the bugs library to see if this is a known misconception.

Since the bugs library has been represented as a set production rules, where each and every misconception is a production rule, the process model uses data-driven forward chaining to traverse the rules, with the data being the user input. When the forward chainer finds a rule that describes the misconception then the action suggested by the rule is the output of the process model. If this would involve visiting another instruction knowledge unit, then a referential link is created from the current student hyperframe to the student hyperframe of that unit and the name of the rule depicting the misconception is given as the name of the link.

6.1.9 The Tutor Knowledge Process Model: Didactics

The tutoring knowledge model comprises of the pedagogical principles that underlie the tutorial interaction with the tutoring system. At the local level, the process model is responsible for the tutoring tactics. This involves selecting the next teaching goal to be satisfied from the teaching goals hyperframe and applying an appropriate teaching strategy for attaining this goal. Once a teaching strategy has been applied, the process model uses the user input as the data for a data-driven forward chainer that traverses the rule-based comprising the teaching strategy, in order to interpret the user input.

At the global level, didactic decisions are taken solely by the pedagogical process model which is responsible for the system's pedagogical strategy that is carried through the system's didactic operation. The pedagogical process model is also responsible for controlling the flow of interaction between the three knowledge process models in order to support the tutoring system's didactic operation both at the local level and at the global level.

The role of the pedagogical process model at the local level is to coordinate the interaction of the three knowledge process models in order to satisfy the local teaching goals. It involves instructing the tutor knowledge process model to select the next teaching goal and an appropriate teaching strategy, and the domain expert knowledge process model to provide the relevant knowledge for tutoring with the goal. It also involves instructing the student knowledge process model to check the user input for correctness, and instructing the tutor model to break away from the tutorial plan and visit a particular instruction knowledge unit, if the student knowledge process model calls for remedial action.

At the global level, by working in synergy with the three knowledge process models, the pedagogical process model coordinates material sequencing. This results in advancing the user to the next instruction knowledge unit, selecting the best teaching strategy overall for the current user and performing means-ends analysis at regular intervals by evaluating the learner's overall knowledge status and consequently to classifying the learner as an end-user.

The user classification helps the pedagogical process model decide on learner control. The flow of interaction is by default under the continuous control of the pedagogical process model. However, as the user becomes more experienced, the system eases the amount of control it exercises on the user-learner and allows more freedom to the student to navigate through the hyperspace by pursuing links, initiating searches, involves the system's generative behaviour, etc. A more precise examination of the functional role of the pedagogical process model will be unravelled in the context of the rest of the didactic operation.

6.2 THE DIDACTIC PLAN OF ACTION

The organisation and hierarchical structure of the domain knowledge defines a default, nevertheless, explicit didactic plan of action for the didactic operation of this tutoring system. This default plan involves taking the student through a succession of instruction knowledge units by following the organisational links of the domain knowledge. Traversal of this hierarchical tree is arranged by the pedagogical process model working in synergy with the domain expert knowledge process model.

The pedagogical process model defines the way the domain knowledge tree is to be traversed, and the domain expert knowledge process model specifies the domain hyperframe to be retrieved. If the domain hyperframe to be retrieved is not the top level one, then the domain expert knowledge process model retrieves the name of the domain hyperframe from the current domain hyperframe in which it appears as an attribute slot value. The hypertext engine then retrieves the hypertext node in which this domain hyperframe is stored. The pedagogical process model then advances the student to the instruction knowledge unit part of which is the retrieved domain hyperframe.

Traversal of the domain tree may be performed in a number of ways. It may be traversed, breadth first, which in terms of our domain of discourse means that the domain expert knowledge process model will first take the student through the top level instruction knowledge unit (i.e. the Continents unit), then through each and every instruction knowledge unit in the first level, first the Europe unit, then the America unit, then the Australia unit, then the Asia unit and finally the Africa unit. Then it will take the student to the second level and offer tutoring first with European countries (e.g. take the student to the UK unit, then the France unit, etc.), then with American countries (e.g. take the student through the USA unit, then the Canada unit, etc.). Alternatively, the hierarchical tree of units defined by the domain tree may be traversed depth-first which means that the domain expert knowledge process model will first take the student through the top level instruction knowledge unit (i.e. the Continents unit), then to the Europe unit on the second level, then through the UK unit on the third level, eventually to go back to the top level unit and visit the second leftmost leaf of the domain tree (i.e. the America Unit). These "default" global level operations define the teaching curriculum and the student's learning path through it.

In the context of each instruction knowledge unit that the domain expert knowledge process model takes the student to, the pedagogical process model generates a didactic episode in which the goals in the teaching goals hyperframe must be fulfilled. Once within the instruction knowledge unit, the first task of the tutor knowledge process model is to retrieve the first available teaching goal from the teaching goals hyperframe. Goal selection is achieved with the assistance of the hypertext engine. The hypertext engine goes through the attribute slots that depict the teaching goals one by one and retrieves the first attribute slot from which there is no bidirectional link to the student hyperframe. Absence of a bidirectional link, as was previously explained, means that the goal has not been attempted yet by the student. Once the hypertext engine delivers an attribute slot, the tutor knowledge process model will retrieve the context of the slot (i.e. the teaching goal). The second task of the tutor knowledge process model is to retrieve the names of the teaching strategies that are appropriate for tutoring with the goal. The hypertext engine will follow the links from the attribute slot to the teaching strategies and retrieve them. For example, if the instruction knowledge unit is the one on Europe (see Figure 6.8) and the unit is visited for the first time, that is there are no bidirectional links to and from any of the teaching goals to the student hyperframe, then the tutor knowledge process model will retrieve the first available teaching goal (i.e. "What is Europe") along with teaching strategies 1 and 2.

The pedagogical process model then calls the hypertext engine to follow the information links from the teaching goal to the corresponding domain hyperframe attribute slots. Once the hypertext engine retrieves these slots, the domain expert knowledge process model then retrieves the context of these attribute slots. Therefore, for the goal, "What is Europe" the domain expert knowledge process model retrieves, "part-of: continents" and "specialisation-of: continent".

The pedagogical process model then calls the student knowledge process model to retrieve the best teaching strategy, so far, for tutoring the student with this type of goal and use it for tutoring with the current goal, if this is a member of the set of strategies that have been retrieved with the selected teaching goal. If this is not a member of the set of teaching strategies that have been retrieved with the goal or if an overall or attribute-specific "best" teaching strategy has not been recorded yet, then the first of the teaching strategies associated with the goal will be used for tutoring with the goal. Once the strategy has been selected then, if, for instance, the teaching strategy is "Question/Answering" then the teaching goal is incorporated in a question which is posed to the student to answer it. If the teaching strategy is "Evaluation of Student Response" then the student is asked to state what he knows about the goal.

When the student provides an answer, the pedagogical process model then calls the student knowledge process model to check the user input for correctness. The student knowledge process model compares the user input against the domain expert knowledge retrieved by the domain expert knowledge process model. If it matches the domain expert knowledge then the student knowledge process model records the user input in the corresponding student hyperframe attribute slot. If the student hyperframe does not exist, the hypertext engine is called to create it, as explained earlier on in this chapter. The student knowledge process model then calls the hypertext engine to create a bidirectional referential link from the student attribute slot that has just been filled, to the corresponding attribute slot in the teaching goals hyperframe, holding the teaching goal. This will designate to the tutor knowledge process model that this teaching goal has been satisfied with the current instruction knowledge unit, in order not to be selected again when the pedagogical process model calls for the next teaching goal. It also indicates to the student process model which teaching goals have been satisfied so far.

It then calls the hypertext engine to create a referential link from the value that has just been added in the attribute slot in the student hyperframe, to the corresponding attribute slot in the domain hyperframe. If a domain attribute slot corresponding to the selected teaching goal contains several values of which a certain number, indicated by the teaching goal, are required for mastery of the goal, then the name of this referential link carries the number scored so far by the student. This link will help the student knowledge process model match the student model against the domain model and thus calculate how close the student overlay model is to the domain model.

For example, if teaching strategy 1 is selected for tutoring with the selected goal then the question, "What is Europe?" is posed to the student. If the student answers that "Europe is a continent" which is correct then the student knowledge process model, will insert "continent" in the "specialisation-of" slot of the student hyperframe and "continents" in the "part-of" slot. It will then call the hypertext engine to create bidirectional referential links from both of these slots to the corresponding attribute slots in the teaching goals hyperframe and name both of these links Goal-1 and also to the corresponding attribute slots in the domain hyperframe and give as names to these links the name of the attribute slot.

Attribute slot values in the domain hyperframe that have their own hyperframes are linked to these with information links. Therefore, if an attribute slot in the student hyperframe has been filled with such a value, then a copy of the information link in the domain hyperframe is created by the hypertext engine, from the student hyperframe attribute slot value to the corresponding student hyperframe. In addition, an overlay statistic is added to the name of this computed link to indicate mastery of the concept-to-concept relationship. The value of the statistic is between the range of -2 to 2 where 2 indicates correctness, 1 indicates correctness after assistance, -1 indicates incorrectness, -2 indicates incorrectness after assistance and 0 indicates that the user has not provided the system with input about it. Similarly, the best teaching strategy for satisfying the goal has to be recorded in the attribute slot of the student hyperframe and a link created by the hypertext engine to this "best" teaching strategy. In our example, if applying teaching strategy 1 ended in success with tutoring with "What is Europe" (i.e. the student scored 2s) then the name of this teaching strategy is added in the attribute slot of the student hyperframe, and the hypertext engine creates a link from this value to the corresponding node holding the strategy. The name of the strategy is also recorded as a best teaching strategy for tutoring with this kind of goal.

If the student knowledge process model diagnoses a misconception in the student answer, that is the student answer was not in the values retrieved from the domain hyperframe and when the student knowledge process model checked the misconceptions rules in the bugs library the expert system came up with a rule that described the misconception, then it calls for the remedial action, suggested by the misconception, to be taken. In this case, the pedagogical process model breaks away from the "default" plan of action and assumes a remedial plan. This involves taking the student to an instruction knowledge unit that was not part of the didactic plan. For example, if the teaching goal selected by the tutor knowledge process model was "European countries (10)", the teaching strategy selected was Evaluation of Student Response, the student was asked to "Name ten European Countries", and the student stated "Turkey" as one of these, then when the student knowledge process model checks this against the values retrieved from the corresponding attribute slot in the domain hyperframe, then this is a clear case of misconception.

The student knowledge process model records Turkey as a misconception in the appropriate attribute slot of the student hyperframe, displays the diagnostic message, "Turkey is not part of Europe", calls for remedial action which involves taking the student to the instruction knowledge unit for Turkey. The pedagogical process model calls the domain expert knowledge process model to specify the name of the domain hyperframe to visit and calls the hypertext engine to retrieve this domain hyperframe and also create a referential link from Turkey in the student hyperframe attribute slot to the Student Turkey hyperframe. The hypertext engine names this referential link with the name of the rule that proved the misconception. Once the instruction knowledge unit is retrieved, tutoring proceeds as before.

In addition to the diagnostic message delivered, the system according to how has the user been classified during the last means-ends analysis, may tell him which continent Turkey is part of. For instance, if the user is a novice then the student knowledge process model calls the domain expert knowledge process model to retrieve the continent Turkey is part of and thus display instead, "Turkey is not part of Europe, but of Asia".

The pedagogical process model may also break away from its didactic plan of action, by pursuing non-organisational information links. For instance, if the system is following a depth first traversal and is tutoring about Greece as part of the "European countries" teaching goal in the Europe unit, and there is the Neighbours-by-land keyword link from the Greece hyperframe to the Turkey hyperframe, then although Turkey is not part of Europe and thus not currently part of the didactic plan, the pedagogical process model may instruct the domain expert knowledge process model to take the student to the Turkey hyperframe and issue tutoring. The instructional designer may provide a variety of circumstances during which this may happen.

At the global level, the didactic operation must perform another two tasks: first, to signal the end of tutoring, and thus leave the student to explore the environment at his own pace, and second, to classify the user according to his performance. With respect to the former, tutoring will inevitably terminate when all, or the majority of, teaching goals in all teaching frames have been successfully satisfied. However, because of the generative behaviour of the system, discovery learning may thereafter continue, as there are virtually unlimited number of viewpoints the user may explore with the system. For instance, the user may search for all German-speaking countries, all countries crossed by Alps, all countries that get wet by the Atlantic Ocean, visit all European countries which he did not visit, etc.

With respect to the latter, the user is classified as a learner according to the current status of missing concepts from his overlay model and the number and nature of his misconceptions. In terms of the missing concepts, the pedagogical process model calls the hypertext engine to retrieve the bidirectional links from all the domain hyperframes attribute slots to the student hyperframes attribute slots. Missing bidirectional links are interpreted as missing concepts. In those cases where the student is asked to give only a fraction of the values and thereby engage in tutoring with the units of these values only, for example, European countries (10), the name of the bidirectional link from the domain hyperframe attribute slot to the corresponding student hyperframe attribute slot in the unit, designates how many of these values the student has named and which domain hyperframes from this group of domain hyperframes he is expected to have visited, so the rest can be ignored.

In the above example, the student is expected to have named at least ten European countries and thus engaged in tutoring with only these. Absence of bidirectional links from the rest of European countries is not taken as missing concepts. According to how many missing concepts the student has and how many misconceptions and the nature of these misconceptions, he will be classified as a novice, advanced beginner, competent, proficient, expert, master, etc. This classification will then be used by the system to make decisions such as how many tries to allow the user, for example, at a question before giving a hint or even the correct answer, increase the number of European countries required in the Europe unit from 10 to 20, how much explanation to provide, etc., and finally the amount of freedom to explore the environment, for instance, change the teaching strategy at will, change the teaching goal, engage the system's generative ability, visit an instruction unit which is not part of the current didactic plan, etc.

6.3 THE PEDAGOGICAL CONTEXT

The "default" pedagogical context for the application of the system's didactic operation is that defined by the plan of action. It is **plan-based** since the pedagogical process model manipulates the sequences of experiences, that is the didactic episodes, through which the student is expected to acquire the target expertise. The teaching goals predominate during interaction. The pedagogical process model plans the interaction both at the local level which involves attaining all, or the majority of, the teaching goals in the current instruction knowledge unit, and at the global level which involves proceeding with instruction with the next instruction knowledge unit in this hierarchal planning until the student model has no or very few missing concepts. Within this plan the student has as much freedom as the learner performance evaluation allows to the student.

However, the student knowledge process model plays a central role with tutoring as it does not just monitor the unfolding of the didactic plan and fill the student data bank, but it may trigger such interventions as a result of student diagnosis of misconceptions, that may cause the pedagogical process model to call the didactic operation to deviate from this default plan and take a remedial action. This results in the pedagogical process model monitoring the unfolding of an **opportunistic plan.** This would involve advancing the student to instruction knowledge units that are not part of the didactic plan of action and engage him in tutoring. The pedagogical process model will pursue this "remedial path" and once the student misconception or subsequent misconceptions have been cleared away, the pedagogical process model calls the didactic operation to resume its plan.

Pursuing an opportunistic plan can also be the result of the student engaging the system's generative ability in an information-rich and highly structured environment. Pursuing opportunistic plans may also be the result of switching between different teaching strategies in the context of an instruction knowledge unit. Initially, the pedagogical process model will issue tutoring in a strictly plan-based context and as the user moves up the classification scale, the pedagogical process model will give the student more freedom to explore opportunistic plans. Once the pedagogical process model has achieved its target goals and the student model has no, or very few, missing concepts then if the user wishes to interact with the system, the pedagogical context of the didactic operation will be exclusively opportunistic. In such a case, the pedagogical process model will provide tutorial help only if the student asks for it or when the student knowledge process model diagnoses a misconception as a result of this request.

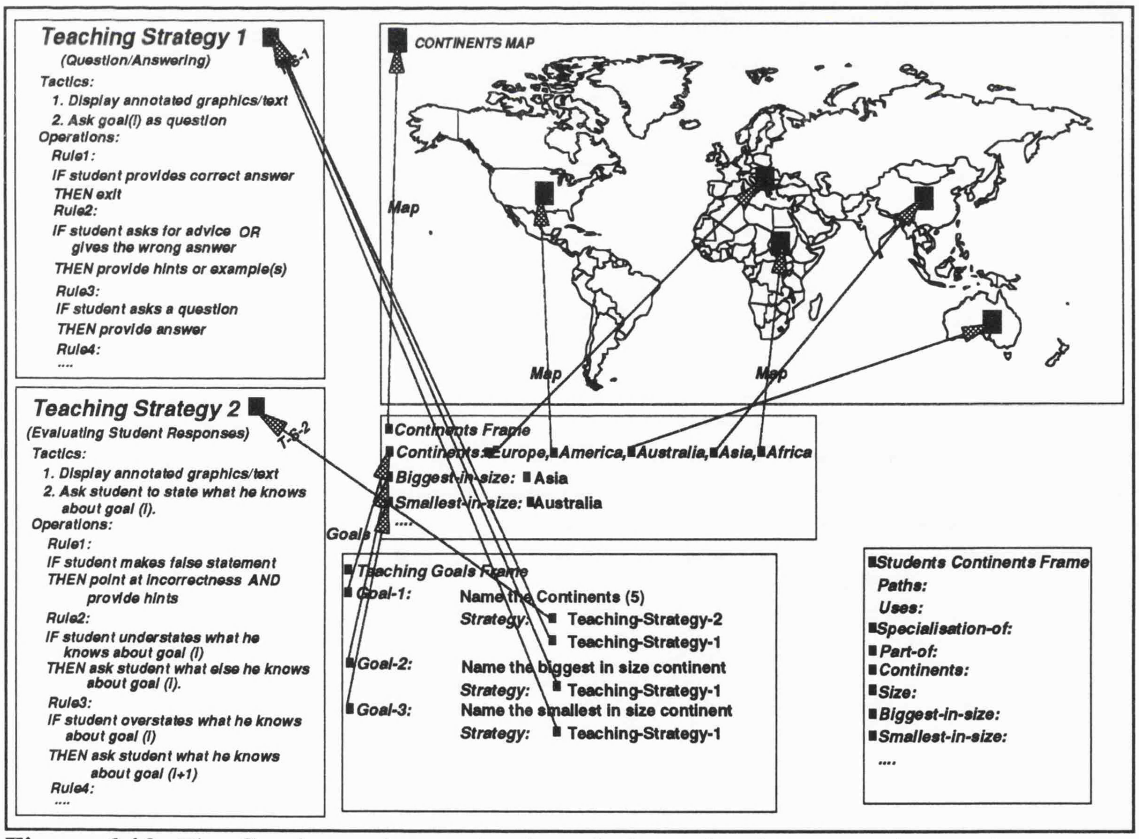
6.4 THE TARGET LEVEL

By organising a didactic episode with each and every instruction knowledge unit that the pedagogical process model takes the student to, the objective of the didactic operation is to modify the knowledge state of the student-user either directly, through communication of knowledge, or indirectly, through practice. The student hyperframe attribute slots are filled with factual knowledge, and the overlay statistics that are part of the names of the links from the knowledge in these attribute slots to other student hyperframes, provide an indication of the level of mastery of the concept-to-concept relationship or an indication of the misconception encountered. This suggests that the target level of the didactic operation is epistemic.

Nevertheless, although the target level of the didactic operation is epistemic, the contents of the student hyperframe also includes behavioural and individual aspects of the actual user-learner, at least those which are necessary to support a full-scale didactic operation at the epistemic level: Pathways to from an instruction unit (e.g. Europe to France), use of the instruction unit (e.g. "plan", "misconception: student thinks Turkey is part of Europe"), "best" teaching strategy and "applied" teaching strategies, explanation requests and non-organisational links to and from a student hyperframe are expositions of a behavioural aspect of a student hyperframe, and the student model overall. These denote user performance within an instruction knowledge unit without directly addressing domain knowledge.

6.5 AN INTERACTION PROTOCOL WITH THE TUTORING SYSTEM

The first tutoring task of the pedagogical process model is to take the student to the top level instruction knowledge unit (i.e. the Continents unit), and start tutoring within it. The hypertext engine creates an empty semi-structured hypertext information node, names it "Student Continents Frame" as in Figure 6.13, and creates the "paths" and "uses" attribute slots. Then the pedagogical process model calls the domain expert knowledge process model to supply the names of the remaining attribute slots, free of values, for the student hyperframe. The hypertext engine then creates attribute slots in the student hyperframe with these names. Figure 6.13 shows the Continents instruction knowledge unit prior to any interaction taking place within it.



Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(I) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
....

Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
1. Display annotated graphics/text
2. Ask student to state what he knows about goal (I).
Operations:
Rule1:
IF student makes false statement
THEN point at incorrectness AND provide hints
Rule2:
IF student understates what he knows about goal (I)
THEN ask student what else he knows about goal (I).
Rule3:
IF student overstates what he knows about goal (I)
THEN ask student what he knows about goal (I+1)
Rule4:
....

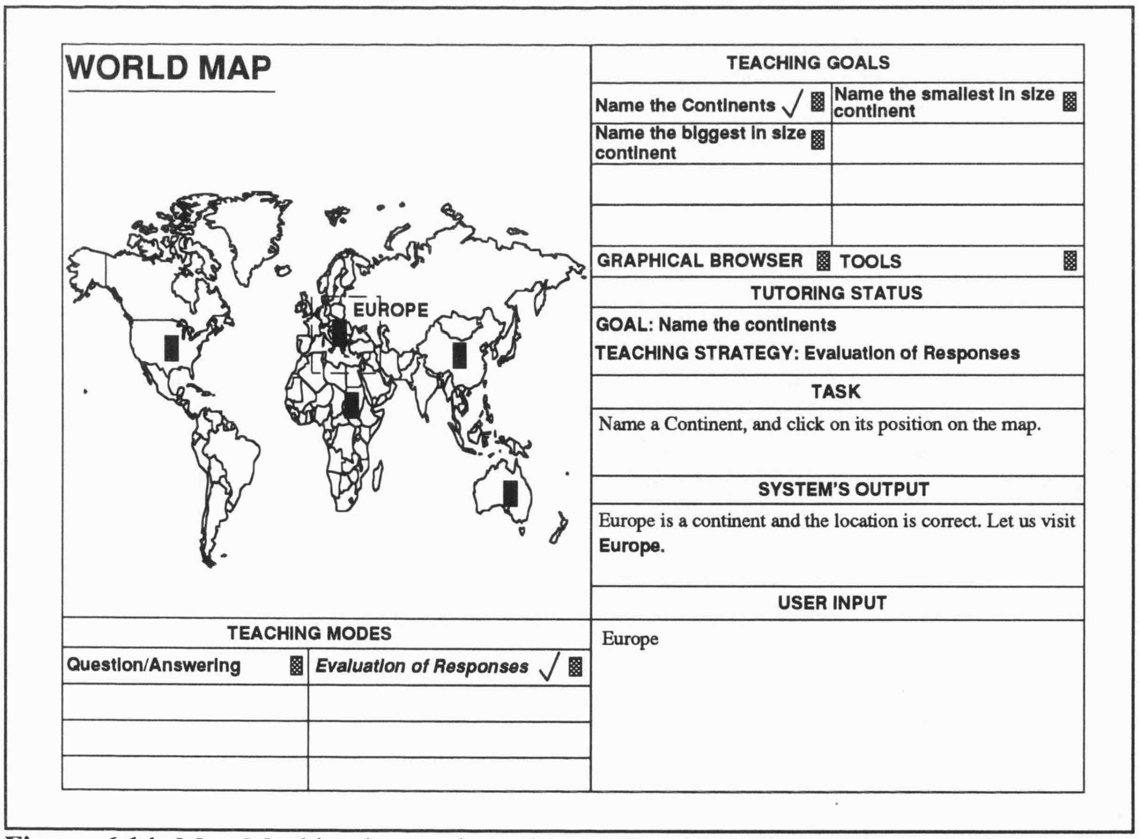
■Continents Frame
■Continents: ■Europe, ■America, ■Australia, ■Asia, ■Africa
■Biggest-in-size: ■ Asia
■Smallest-in-size: ■Australia
....

■Teaching Goals Frame
■Goal-1: Name the Continents (5)
Strategy: ■ Teaching-Strategy-2
■ Teaching-Strategy-1
■Goal-2: Name the biggest in size continent
Strategy: ■ Teaching-Strategy-1
■Goal-3: Name the smallest in size continent
Strategy: ■ Teaching-Strategy-1

■Students Continents Frame
Paths:
Uses:
■Specialisation-of:
■Part-of:
■Continents:
■Size:
■Biggest-in-size:
■Smallest-in-size:
....

Figure 6.13: The Continents instruction knowledge unit prior to any interaction

The next task of the pedagogical process model is to update the user interface. For this purpose, it calls the tutor knowledge process model to provide to the user interface for display the teaching goals that have to be attained with the current instruction knowledge unit, the teaching strategies that are appropriate for tutoring with these goals, the first goal to be attained, "Continents", and the teaching strategy that is to be applied to attain the goal. It then calls the hypertext engine to provide to the user interface for display any annotated graphics or text to the domain hyperframe. Then, according to the context of the teaching strategy, it prompts the user for input. In Figure 6.14 below, the teaching strategy that has been selected by the tutor knowledge process model and retrieved by the hypertext engine is that of the "Evaluation of student response". With this strategy the user is asked to state what he knows about the goal, so the prompt, "name a continent, and click on its position on the World Map". The user in the case, names Europe and clicks correctly with a pointing device on its position on the map. The user input is correct thus the system's output is appropriately given.



TEACHING GOALS

Name the Continents ✓ ▨	Name the smallest in size continent ▨
Name the biggest in size continent ▨	

TUTORING STATUS

GOAL: Name the continents
TEACHING STRATEGY: Evaluation of Responses

TASK

Name a Continent, and click on its position on the map.

SYSTEM'S OUTPUT

Europe is a continent and the location is correct. Let us visit **Europe.**

USER INPUT

Europe

TEACHING MODES

Question/Answering ▨	*Evaluation of Responses* ✓ ▨

Figure 6.14: Man-Machine interaction with the Continents instruction knowledge unit

The pedagogical process model co-ordinates a number of actions, before it arrives at the system's outcome. First, it calls the hypertext engine to follow the links from the teaching goal to the corresponding attribute slots in the domain hyperframe, and then calls the domain expert knowledge process model to retrieve the contents of these attribute slots. Then it calls the student knowledge process model to compare the user input with the knowledge retrieved from the domain hyperframe. In our case, the user input is correct therefore, the student knowledge process model fills the corresponding attribute slots in the student hyperframe. The attribute slot "Continents" in the student hyperframe is filled with the "Europe". The student knowledge process model then calls the pedagogical process model to continue with the plan of action.

However, before continuing with the plan of action, the hypertext engine is called by the pedagogical process model to create a referential bi-directional link from the teaching goal that has been attained in the teaching goals hyperframe, to the corresponding attribute slot in the student hyperframe (see Figure 6.15). As explained before, this is to aid the tutor knowledge process model to choose the next goal now or in a future interaction with this instruction knowledge unit. It also creates a referential link from the student attribute slot that has been filled to the corresponding domain hyperframe attribute slots. This is to help assess the current state of the student overlay model, that is how much of the domain knowledge has been acquired by the student and consequently classify the user accordingly.

Finally, if there is a domain hyperframe for the value which the student attribute slot has been filled with, the hypertext engine creates the student hyperframe for it. In our case, the hypertext engine creates a semi-structured hypertext information node which it names

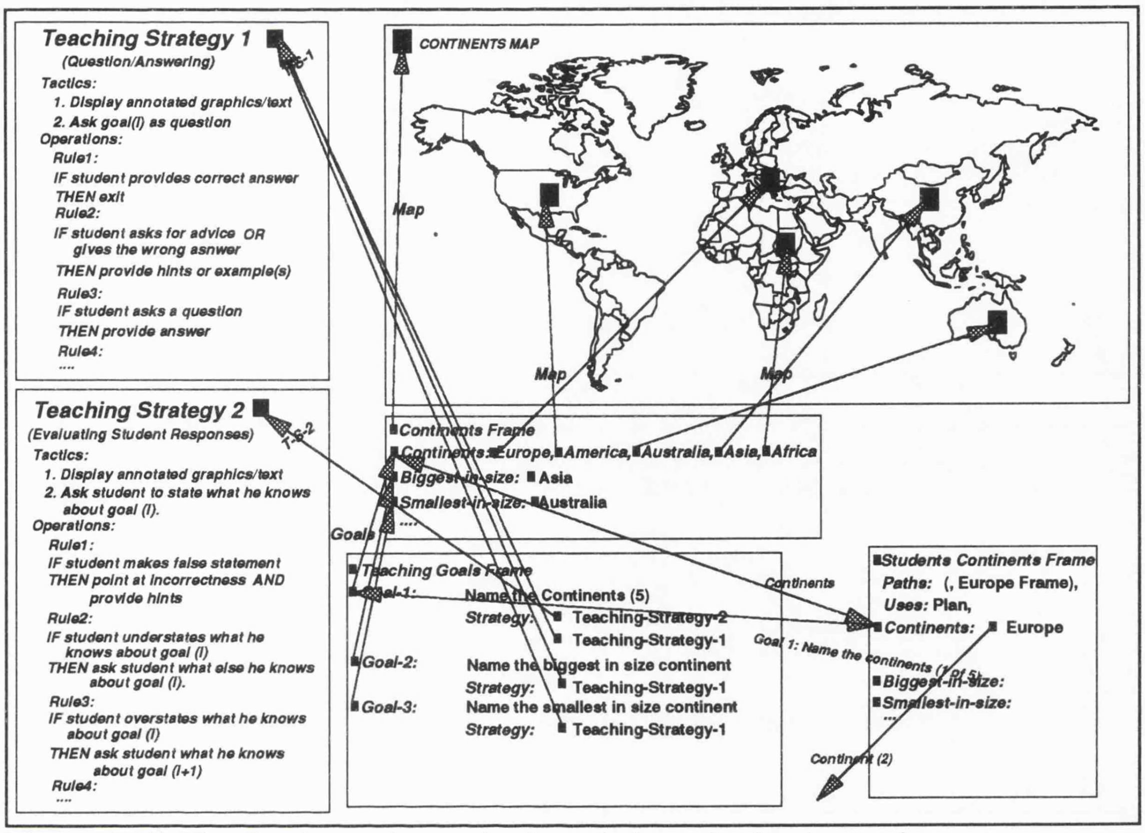

Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(i) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR
gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
....
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
1. Display annotated graphics/text
2. Ask student to state what he knows about goal (i).
Operations:
Rule1:
IF student makes false statement
THEN point at incorrectness AND provide hints
Rule2:
IF student understates what he knows about goal (i)
THEN ask student what else he knows about goal (i).
Rule3:
IF student overstates what he knows about goal (i)
THEN ask student what he knows about goal (i+1)
Rule4:
....
CONTINENTS MAP
Map
Map
Map
Continents Frame
Continents: Europe, America, Australia, Asia, Africa
Biggest-in-size: Asia
Smallest-in-size: Australia
....
Goals
Teaching Goals Frame
Goal-1: Name the Continents (5)
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
Goal-2: Name the biggest in size continent
Strategy: Teaching-Strategy-1
Goal-3: Name the smallest in size continent
Strategy: Teaching-Strategy-1
Continents
Goal 1: Name the continents (1 of 5)
Students Continents Frame
Paths: (, Europe Frame),
Uses: Plan,
Continents: Europe
Biggest-in-size:
Smallest-in-size:
...
Continent (2)
T-S-1
T-S-2

Figure 6.15: The Continents instruction knowledge unit after interaction

"Student Europe Frame" and also creates a referential link to it from the value "Europe" in the corresponding student continents hyperframe attribute slot. Figure 6.16 represents the current state of the student knowledge model.

Let us assume that the pedagogical process model instructed a depth-first domain tree traversal. In this case, the pedagogical process model calls the hypertext engine to follow an organisational hypertext information link from the continents unit to the next level in this hierarchical tree. Since the user has answered Europe, the hypertext engine follows the link to the Europe Hyperframe which the pedagogical process model calls the domain expert knowledge process model to retrieve.

Before leaving the Continents unit, the student knowledge process model fills the "path" attribute slot in the student continents hyperframe with "(, Europe Frame) and the "uses"

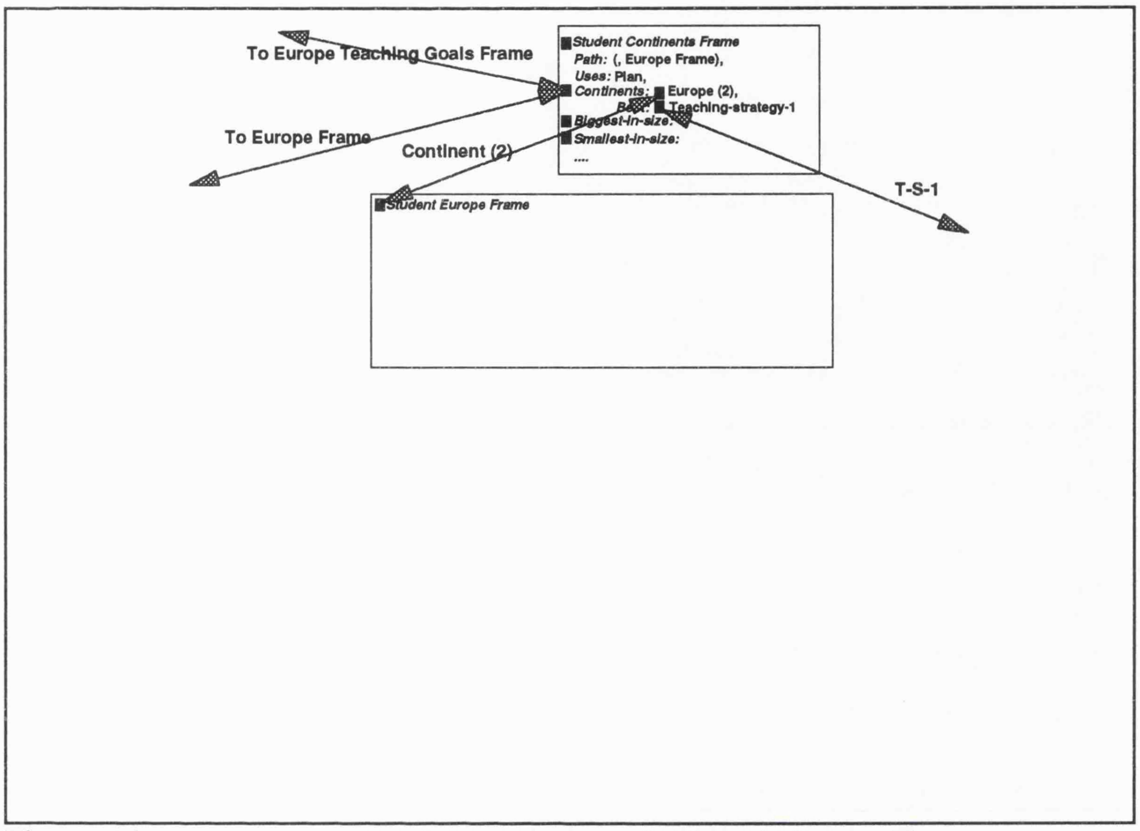
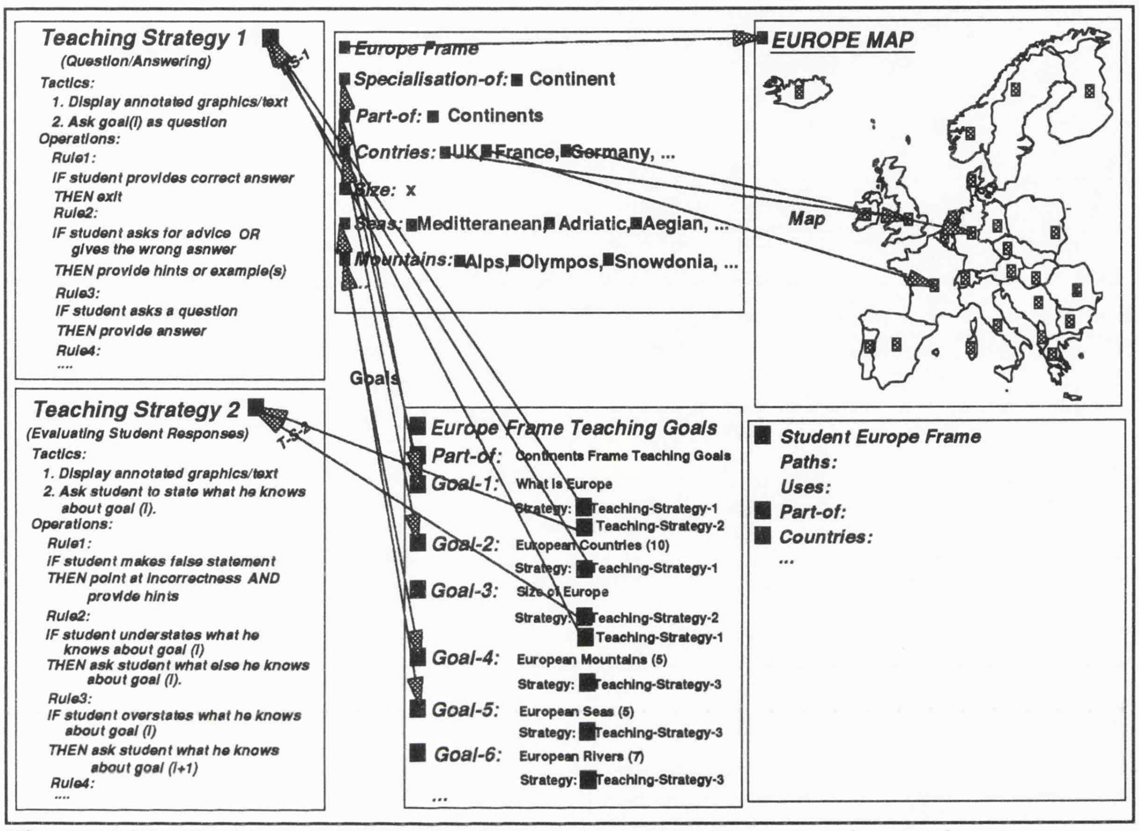


Figure 6.16: The current state of the student overlay knowledge model

attribute slot with "plan" to denote that the didactic operation is advancing the user from the Continents frame to the Europe frame as part of the didactic plan. In both Figures 6.15 and 16 the hypertext information links within the student model carry an overlay statistic that represents the level of mastery of a concept or a relationship. The bidirectional link from the teaching goal to the student attribute slot also carries an indication of the level of achievement of the student. For instance, 1 of 5 suggests that the student has named correctly one continent so far.

Once in the Europe Unit, the pedagogical process model calls the hypertext engine to create the "paths" and "uses" attribute slots in the student hyperframe. Then the pedagogical process model calls the domain expert knowledge process model to supply the names of the remaining attribute slots for the student Europe hyperframe. The hypertext engine then creates attribute slots with these names in the student Europe hyperframe. Figure 6.17 shows the Europe instruction knowledge unit prior to any interaction taking place within it.



```
Teaching Strategy 1
  (Question/Answering)
Tactics:
  1. Display annotated graphics/text
  2. Ask goal(I) as question
Operations:
  Rule1:
  IF student provides correct answer
  THEN exit
  Rule2:
  IF student asks for advice  OR
     gives the wrong asnwer
  THEN provide hints or example(s)
  Rule3:
  IF student asks a question
  THEN provide answer
  Rule4:
  ....
```

```
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
  1. Display annotated graphics/text
  2. Ask student to state what he knows
     about goal (I).
Operations:
  Rule1:
  IF student makes false statement
  THEN point at incorrectness  AND
         provide hints
  Rule2:
  IF student understates what he
     knows about goal (I)
  THEN ask student what else he knows
         about goal (I).
  Rule3:
  IF student overstates what he knows
     about goal (I)
  THEN ask student what he knows
         about goal (I+1)
  Rule4:
  ....
```

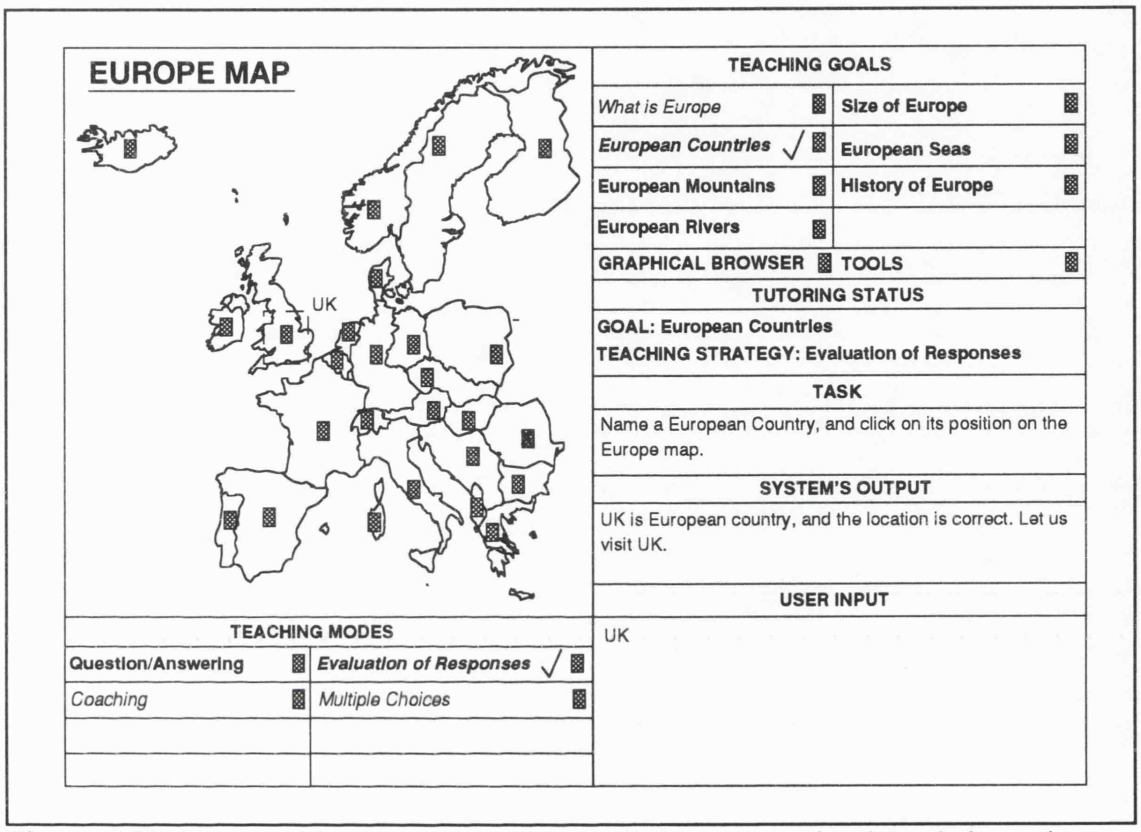
```
■ Europe Frame
■ Specialisation-of: ■ Continent
■ Part-of: ■ Continents
■ Contries: ■UK,■France,■Germany, ...
■ Size: x
■ Seas: ■Mediterranean,■Adriatic,■Aegian, ...
■ Mountains: ■Alps,■Olympos,■Snowdonia, ...
  ...
```

```
■ Europe Frame Teaching Goals
■ Part-of:  Continents Frame Teaching Goals
■ Goal-1:   What is Europe
            Strategy: ■Teaching-Strategy-1
                      ■Teaching-Strategy-2
■ Goal-2:   European Countries (10)
            Strategy: ■Teaching-Strategy-1
■ Goal-3:   Size of Europe
            Strategy: ■Teaching-Strategy-2
                      ■Teaching-Strategy-1
■ Goal-4:   European Mountains (5)
            Strategy: ■Teaching-Strategy-3
■ Goal-5:   European Seas (5)
            Strategy: ■Teaching-Strategy-3
■ Goal-6:   European Rivers (7)
            Strategy: ■Teaching-Strategy-3
  ...
```

```
■ Student Europe Frame
  Paths:
  Uses:
■ Part-of:
■ Countries:
  ...
```

Figure 6.17: The Europe instruction knowledge unit prior to any interaction

The next task of the pedagogical process model is to update the user interface. For this purpose, it calls the tutor knowledge process model to display the teaching goals that have to be attained with the current instruction knowledge unit, the teaching strategies that are appropriate for tutoring with these goals, the first goal to be attained, "European countries", and the teaching strategy that is to be applied to attain this goal. It then calls the hypertext engine to provide to the user interface for display any annotated graphics or text to the domain hyperframe. Then, according to the context of the teaching strategy, it prompts the user for input. In Figure 6.18 below the teaching strategy that has been selected by the tutor knowledge process model and retrieved by the hypertext engine is that of the "Evaluation of student response". With this strategy the user is asked to state what he knows about the goal, so the prompt, "Name a European country, and click on its position on the Map of Europe". The user in the case, names the UK and clicks correctly with a pointing device on its position on the map. The user input is correct thus the system's output is appropriately given.



TEACHING GOALS

What is Europe ▨	**Size of Europe** ▨
European Countries ✓ ▨	**European Seas** ▨
European Mountains ▨	**History of Europe** ▨
European Rivers ▨	
GRAPHICAL BROWSER ▨	**TOOLS** ▨

TUTORING STATUS

GOAL: European Countries
TEACHING STRATEGY: Evaluation of Responses

TASK

Name a European Country, and click on its position on the Europe map.

SYSTEM'S OUTPUT

UK is European country, and the location is correct. Let us visit UK.

USER INPUT

UK

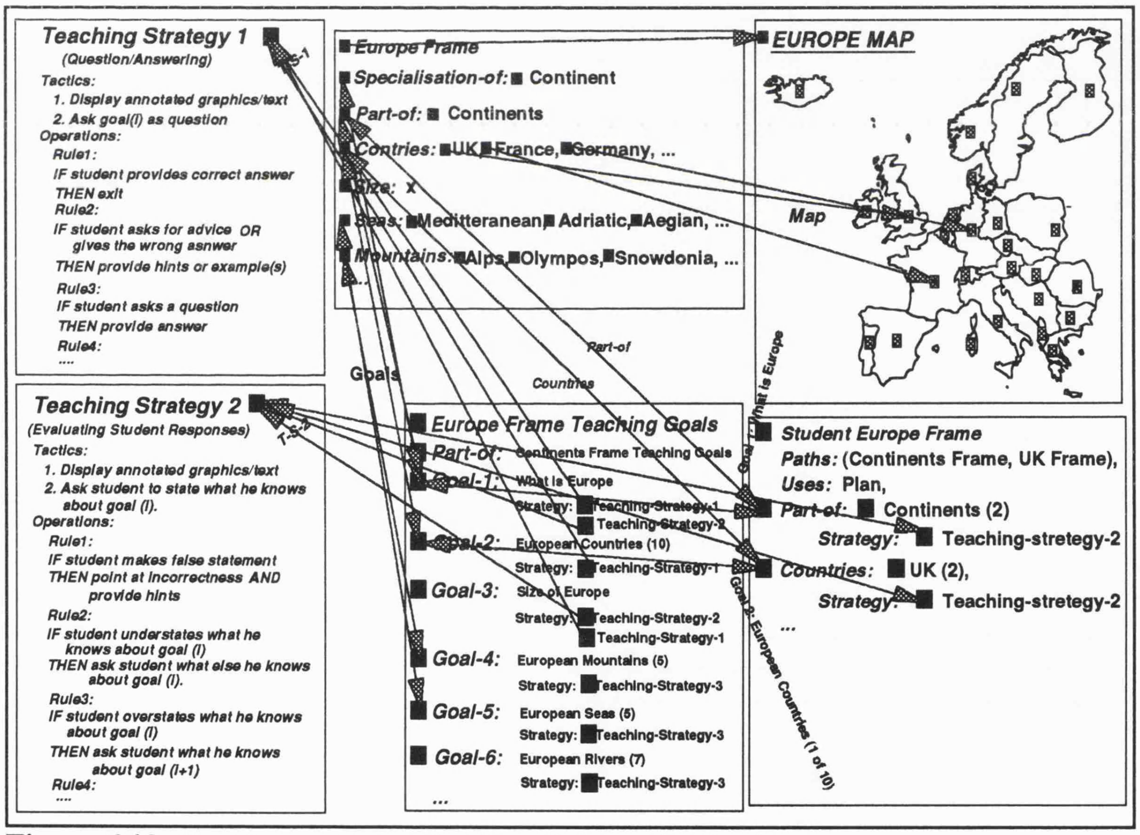
TEACHING MODES

Question/Answering ▨	***Evaluation of Responses*** ✓ ▨
Coaching ▨	*Multiple Choices* ▨

Figure 6.18: Man-machine interaction with the Europe instruction knowledge unit

As before, the pedagogical process model co-ordinates a number of activities, in order to arrive at the system's outcome. First, it calls the hypertext engine to follow the links from the teaching goal to the corresponding attribute slots in the domain hyperframe, and then calls the domain expert knowledge process model to retrieve the contents of these attribute slots. Then it calls the student knowledge process model to compare the user input with the knowledge retrieved from the domain hyperframe. In this case, the user input is correct, and therefore, the student knowledge process model fills the corresponding attribute slots in the student hyperframe, that is the attribute slot "Countries" in the student hyperframe is filled with "UK". The student knowledge process model then calls the pedagogical process model to continue with the plan of action.

However, before continuing with the plan of action, the hypertext engine, as before, is called by the pedagogical process model to create a referential bi-directional link from the teaching goal that has been attained in the Europe teaching goals hyperframe, to the corresponding attribute slot in the student hyperframe (see Figure 6.19). This is to assist the tutor knowledge process model to choose the next goal now or in a future interaction with this instruction knowledge unit. It also creates a referential link from the student attribute slot that has been filled with knowledge, to the corresponding domain hyperframe attribute slots. This is to help assess the current state of the student overlay model, that is how much of the domain knowledge has been acquired by the student and thus classify the user accordingly.



Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(I) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR
gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
....
T-S-1
Europe Frame
Specialisation-of: Continent
Part-of: Continents
Contries: UK, France, Germany, ...
Size: X
Seas: Mediterranean, Adriatic, Aegian, ...
Mountains: Alps, Olympos, Snowdonia, ...
...
EUROPE MAP
Map
Goals
Part-of
Countries
Goal 1: What is Europe
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
1. Display annotated graphics/text
2. Ask student to state what he knows about goal (I).
Operations:
Rule1:
IF student makes false statement
THEN point at incorrectness AND provide hints
Rule2:
IF student understates what he knows about goal (I)
THEN ask student what else he knows about goal (I).
Rule3:
IF student overstates what he knows about goal (I)
THEN ask student what he knows about goal (I+1)
Rule4:
....
T-S-2
Europe Frame Teaching Goals
Part-of: Continents Frame Teaching Goals
Goal-1: What is Europe
Strategy: Teaching-Strategy-1
Teaching-Strategy-2
Goal-2: European Countries (10)
Strategy: Teaching-Strategy-1
Goal-3: Size of Europe
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
Goal-4: European Mountains (5)
Strategy: Teaching-Strategy-3
Goal-5: European Seas (5)
Strategy: Teaching-Strategy-3
Goal-6: European Rivers (7)
Strategy: Teaching-Strategy-3
...
Goal 2: European Countries (1 of 10)
Student Europe Frame
Paths: (Continents Frame, UK Frame),
Uses: Plan,
Part-of: Continents (2)
Strategy: Teaching-stretegy-2
Countries: UK (2),
Strategy: Teaching-stretegy-2
...

Figure 6.19: The Europe instruction knowledge unit after interaction

Finally, if there is a domain hyperframe for the value which the student attribute slot has been filled with, the hypertext engine creates the student hyperframe for it. In our case, the hypertext engine creates a semi-structured hypertext information node which it names

"Student UK Frame" and also creates a referential link to it from the value "UK" in the corresponding student Europe hyperframe attribute slot. Figure 6.20 represents the current state of the student knowledge model.

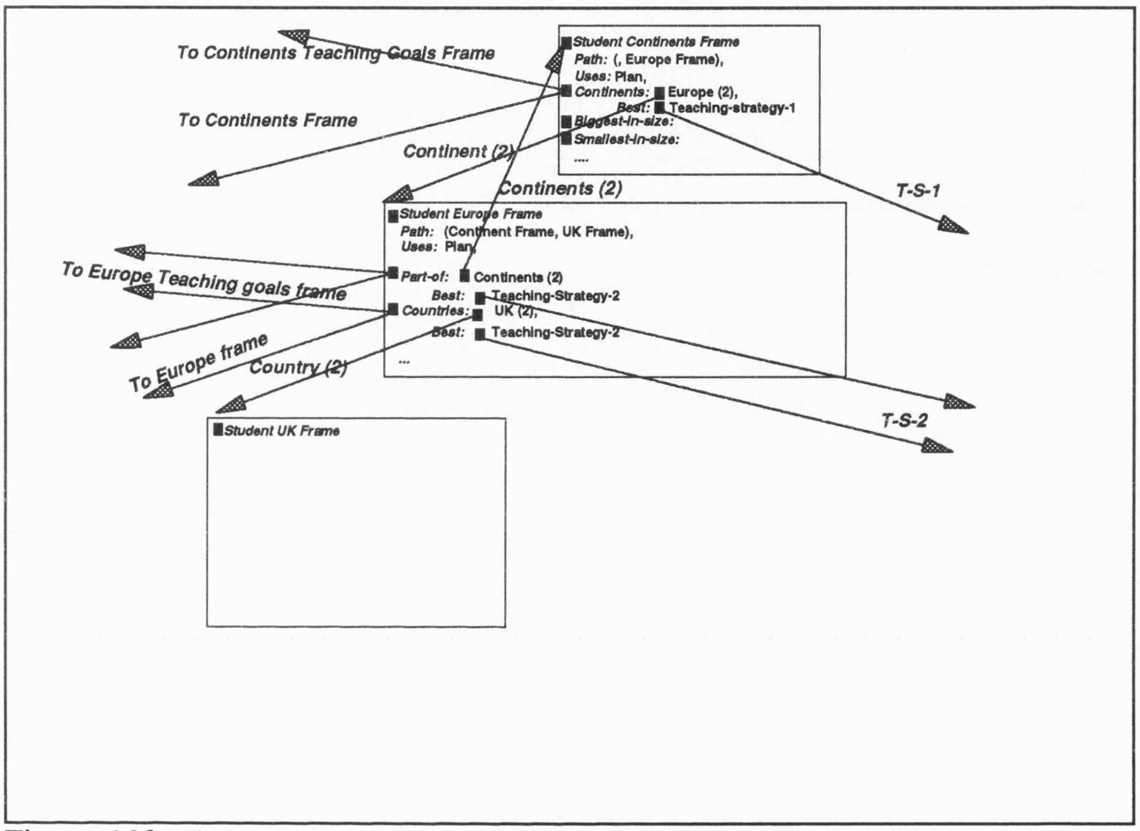
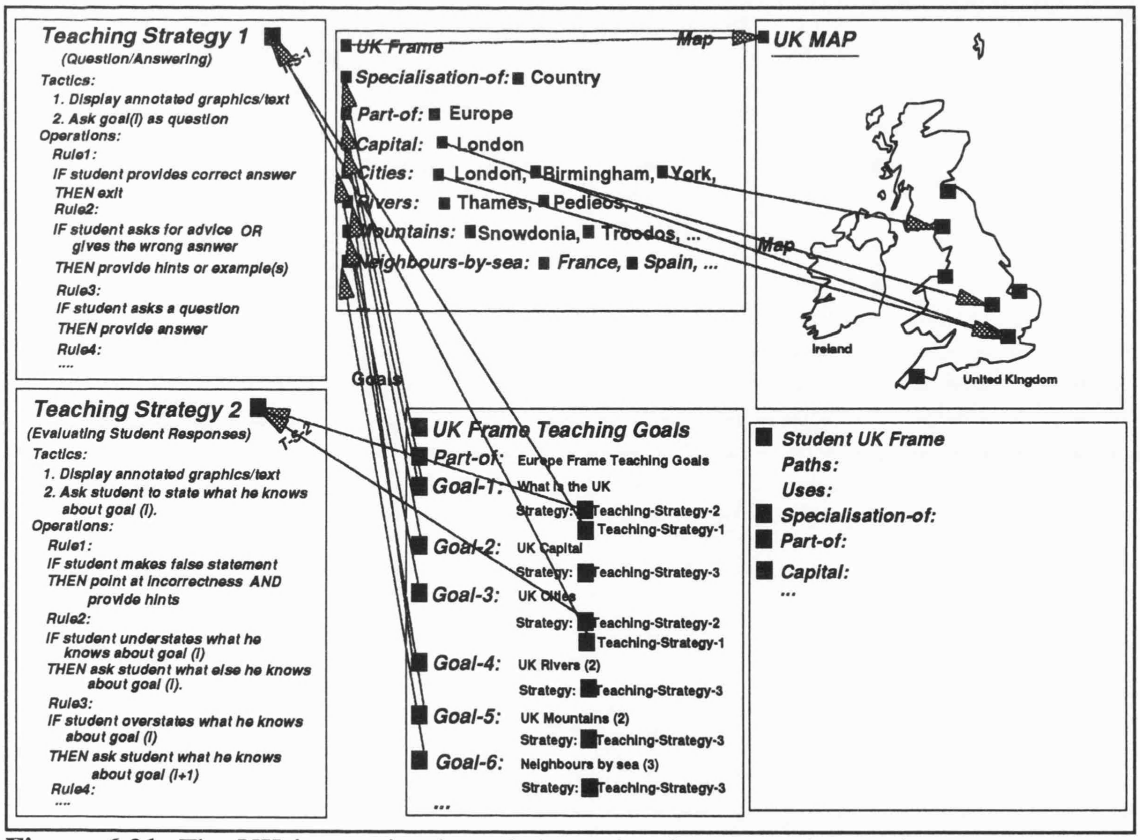


Figure 6.20: The current state of the student overlay knowledge model

The pedagogical process model following its depth-first domain tree traversal calls the hypertext engine to follow an organisational hypertext information link from the Europe unit to the next level in this hierarchical tree. Since the user has answered UK, the hypertext engine follows the organisational link to the UK Hyperframe which the pedagogical process model calls the domain expert knowledge process model to retrieve it.

Before leaving the Europe unit, the student knowledge process model fills the "path" attribute slot in the Student Europe hyperframe with "(Continents Frame, UK Frame)" and the "uses" attribute slot with "plan" to denote that the didactic operation is advancing the user to the UK frame as part of the didactic plan. In both Figures 6.19 and 6.20 the hypertext information links within the student model carry an overlay statistic that represents the level of mastery of a concept or a relationship. The bidirectional link from the teaching goal to the student attribute slot that has been filled also carries an indication of the level of achievement of the student. For example, 1 of 10 suggests that the user has named correctly one European country so far.

Once in the UK Unit, the pedagogical process model calls the hypertext engine to create the "paths" and "uses" attribute slots in the student hyperframe. Then the pedagogical process model calls the domain expert knowledge process model to supply the names of the remaining attribute slots for the student UK hyperframe. The hypertext engine then creates attribute slots with these names in the student UK hyperframe. Figure 6.21 shows the UK instruction knowledge unit prior to any interaction taking place within it.

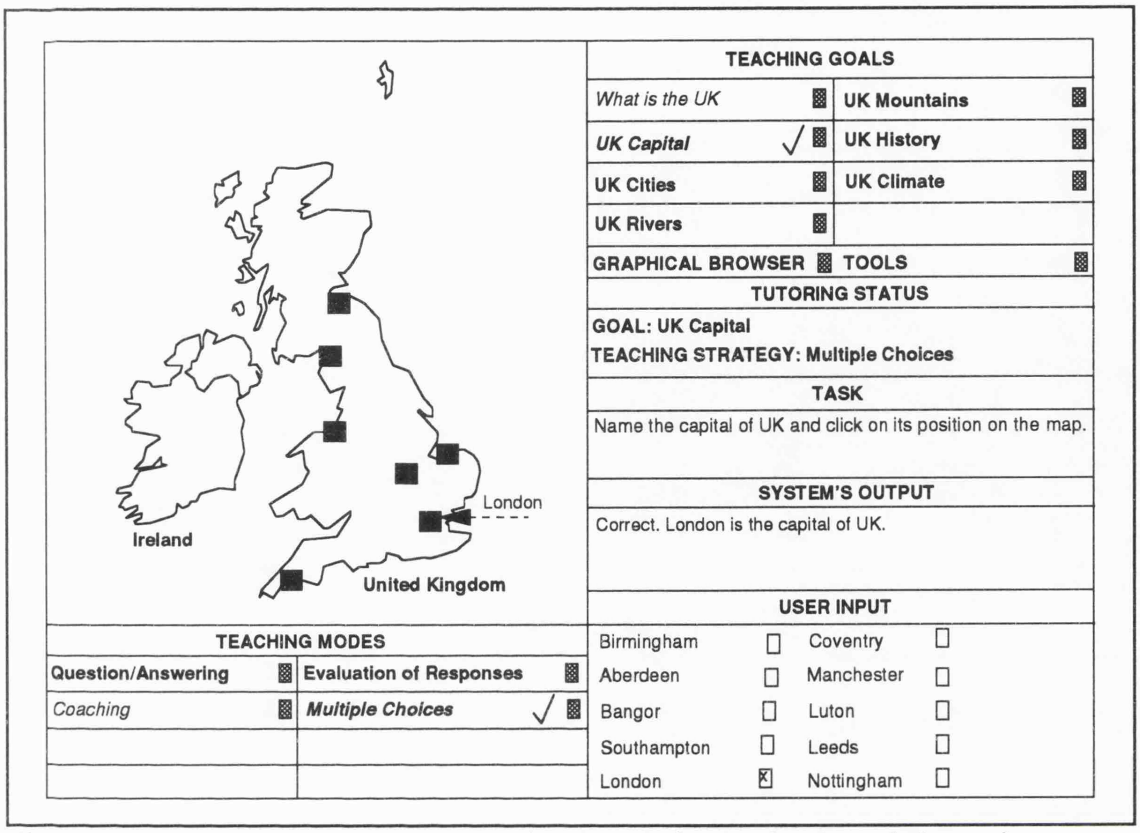


Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(i) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
....
T-S-1
UK Frame
Map
Specialisation-of: Country
Part-of: Europe
Capital: London
Cities: London, Birmingham, York,
Rivers: Thames, Pedieos, ...
Mountains: Snowdonia, Troodos, ...
Neighbours-by-sea: France, Spain, ...
UK MAP
Map
Ireland
United Kingdom
Goals
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
1. Display annotated graphics/text
2. Ask student to state what he knows about goal (i).
Operations:
Rule1:
IF student makes false statement
THEN point at incorrectness AND provide hints
Rule2:
IF student understates what he knows about goal (i)
THEN ask student what else he knows about goal (i).
Rule3:
IF student overstates what he knows about goal (i)
THEN ask student what he knows about goal (i+1)
Rule4:
....
T-S-2
UK Frame Teaching Goals
Part-of: Europe Frame Teaching Goals
Goal-1: What is the UK
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
Goal-2: UK Capital
Strategy: Teaching-Strategy-3
Goal-3: UK Cities
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
Goal-4: UK Rivers (2)
Strategy: Teaching-Strategy-3
Goal-5: UK Mountains (2)
Strategy: Teaching-Strategy-3
Goal-6: Neighbours by sea (3)
Strategy: Teaching-Strategy-3
...
Student UK Frame
Paths:
Uses:
Specialisation-of:
Part-of:
Capital:
...

Figure 6.21: The UK instruction knowledge unit prior to any interaction

The next task of the pedagogical process model is to update the user interface. For this purpose, it calls the tutor knowledge process model to display the teaching goals that have to be attained with the current instruction knowledge unit, the teaching strategies that are appropriate for tutoring with these goals, the first goal to be attained, "UK Capital", and the teaching strategy that is to be applied to attain this goal. It then calls the hypertext engine to provide to the user interface for display any annotated to the domain hyperframe graphics or text. Then, according to the context of the teaching strategy, it prompts the user for input. In Figure 6.22 below, the teaching strategy that has been selected by the tutor knowledge process model and retrieved by the hypertext engine is that of the "Multiple Choice". With this strategy the user is asked to choose from a list of options the capital of the UK, by clicking with a pointing device on what he thinks is the correct answer and then "click on its position on the UK Map". The user, in this case, chooses London and clicks correctly with a pointing device on its position on the map. The user input is correct thus the system's output is appropriately given.

As before, the pedagogical process model co-ordinates a number of activities, in order to arrive at the system's outcome. First, it calls the hypertext engine to follow the links from the teaching goal to the corresponding attribute slots in the domain hyperframe, and then calls the domain expert knowledge process model to retrieve the contents of these attribute slots. Then it calls the student knowledge process model to compare the user input with the knowledge retrieved from the domain hyperframe. In this case, the user input is correct, and therefore, the student knowledge process model fills the corresponding attribute slots in the student hyperframe, that is the attribute slot "UK Capital" in the student hyperframe is filled with "London". The student knowledge process model then calls the pedagogical process model to continue with the plan of action. With this teaching



TEACHING GOALS	
What is the UK ▨	UK Mountains ▨
UK Capital ✓ ▨	UK History ▨
UK Cities ▨	UK Climate ▨
UK Rivers ▨	
GRAPHICAL BROWSER ▨	TOOLS ▨

TUTORING STATUS

GOAL: UK Capital
TEACHING STRATEGY: Multiple Choices

TASK

Name the capital of UK and click on its position on the map.

SYSTEM'S OUTPUT

Correct. London is the capital of UK.

USER INPUT			
Birmingham	☐	Coventry	☐
Aberdeen	☐	Manchester	☐
Bangor	☐	Luton	☐
Southampton	☐	Leeds	☐
London	☒	Nottingham	☐

TEACHING MODES	
Question/Answering ▨	Evaluation of Responses ▨
Coaching ▨	*Multiple Choices* ✓ ▨

Figure 6.22: Man-machine interaction with the UK instruction knowledge unit

strategy the Domain Expert knowledge process model also retrieves additional names of UK cities which it provides for the user interface to display. The Domain Expert Knowledge process model retrieves the names of these cities from its "Cities" attribute slot part of which is also London.

Before continuing with the plan of action, the hypertext engine, as before, is called by the pedagogical process model to create a referential bi-directional link from the teaching goal that has been attained in the UK teaching goals hyperframe, to the corresponding attribute slot in the student UK hyperframe (see Figure 6.23). This is to assist the tutor knowledge process model to choose the next goal now or in a future interaction with this instruction knowledge unit. It also creates a referential link from the student attribute slot that has been filled with knowledge, to the corresponding domain hyperframe attribute slots. This is to help assess the current state of the student overlay model, that is how much of the domain knowledge has been acquired by the student and thus classify the user accordingly.

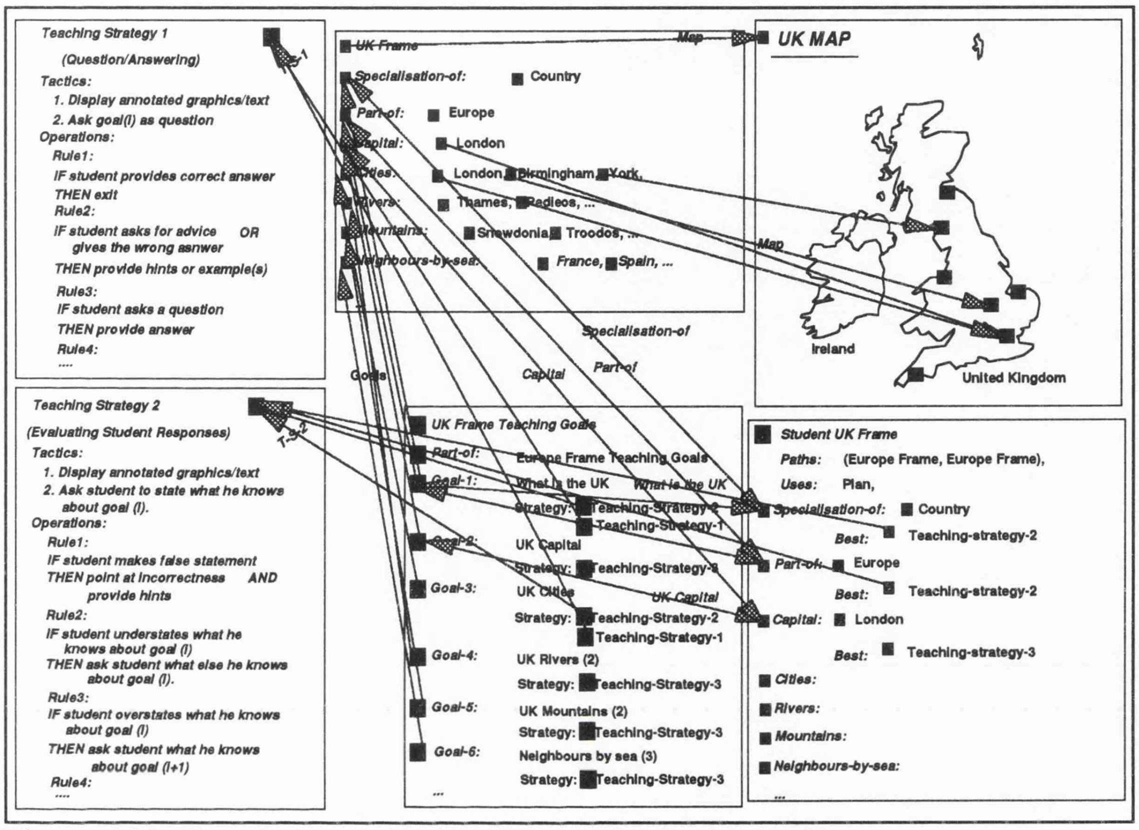

Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(i) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
....
T-S-1
UK Frame
Map
UK MAP
Specialisation-of:
Country
Part-of:
Europe
Capital:
London
Cities:
London, Birmingham, York,
Rivers:
Thames, Redleos, ...
Mountains:
Snowdonia, Troodos, ...
Neighbours-by-sea:
France, Spain, ...
Map
Ireland
United Kingdom
Specialisation-of
Capital
Part-of
Goals
Teaching Strategy 2
(Evaluating Student Responses)
Tactics:
1. Display annotated graphics/text
2. Ask student to state what he knows about goal (i).
Operations:
Rule1:
IF student makes false statement
THEN point at incorrectness AND provide hints
Rule2:
IF student understates what he knows about goal (i)
THEN ask student what else he knows about goal (i).
Rule3:
IF student overstates what he knows about goal (i)
THEN ask student what he knows about goal (i+1)
Rule4:
....
T-S-2
UK Frame Teaching Goals
Part-of: Europe Frame Teaching Goals
Goal-1: What is the UK
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
What is the UK
Goal-2: UK Capital
Strategy: Teaching-Strategy-3
Goal-3: UK Cities
UK Capital
Strategy: Teaching-Strategy-2
Teaching-Strategy-1
Goal-4: UK Rivers (2)
Strategy: Teaching-Strategy-3
Goal-5: UK Mountains (2)
Strategy: Teaching-Strategy-3
Goal-6: Neighbours by sea (3)
Strategy: Teaching-Strategy-3
...
Student UK Frame
Paths: (Europe Frame, Europe Frame),
Uses: Plan,
Specialisation-of: Country
Best: Teaching-strategy-2
Part-of: Europe
Best: Teaching-strategy-2
Capital: London
Best: Teaching-strategy-3
Cities:
Rivers:
Mountains:
Neighbours-by-sea:
...

Figure 6.23: The UK instruction knowledge unit after interaction

Finally, if there is a domain hyperframe for the value which the student attribute slot has been filled with, the hypertext engine creates the student hyperframe for it. Let us assume that the individual country hyperframes are at the bottom level of the domain tree. Figure 6.24 represents the current state of the student knowledge model.

The pedagogical process model following its depth-first domain tree traversal calls the domain expert knowledge process model to retrieve the Europe hyperframe so it can take the user back to the Europe unit. Before leaving the UK unit, the student knowledge process model fills the "path" attribute slot in the student Europe hyperframe with "(Europe Frame, Europe Frame)" and the "uses" with "plan" to denote that the didactic operation is taking the user back to the Europe frame as part of the didactic plan. In both

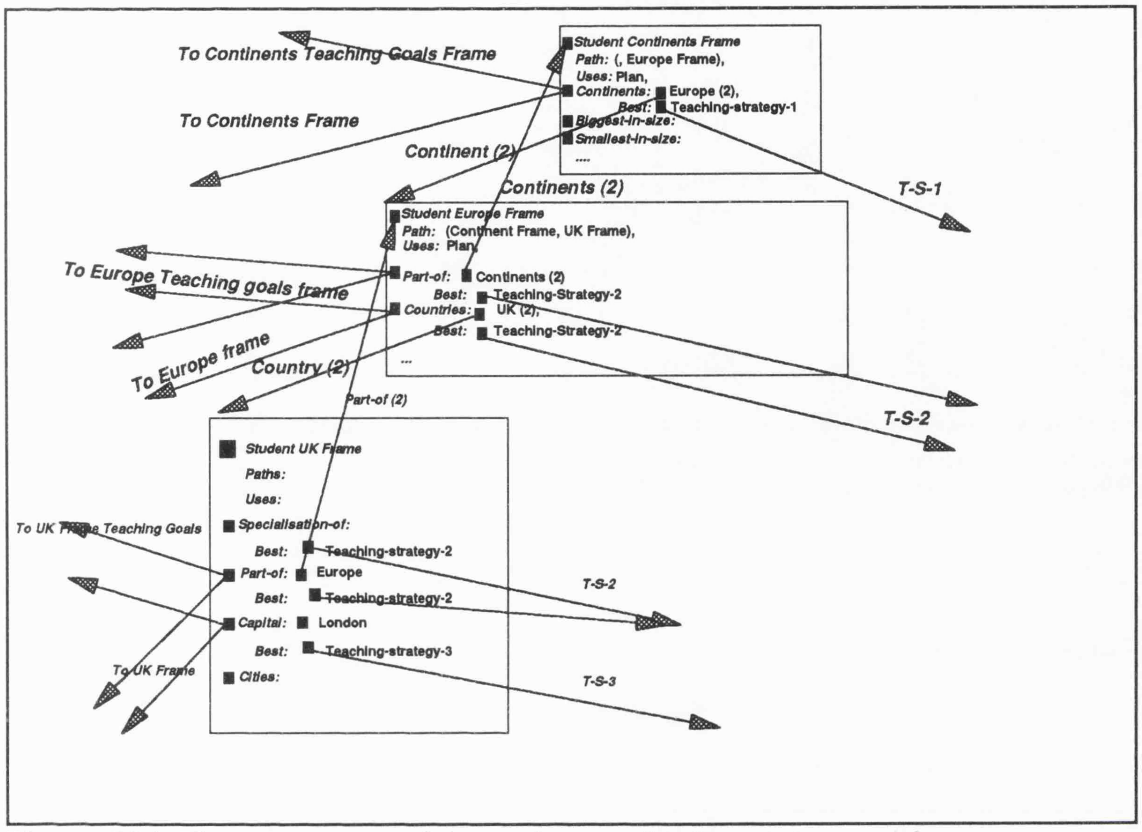
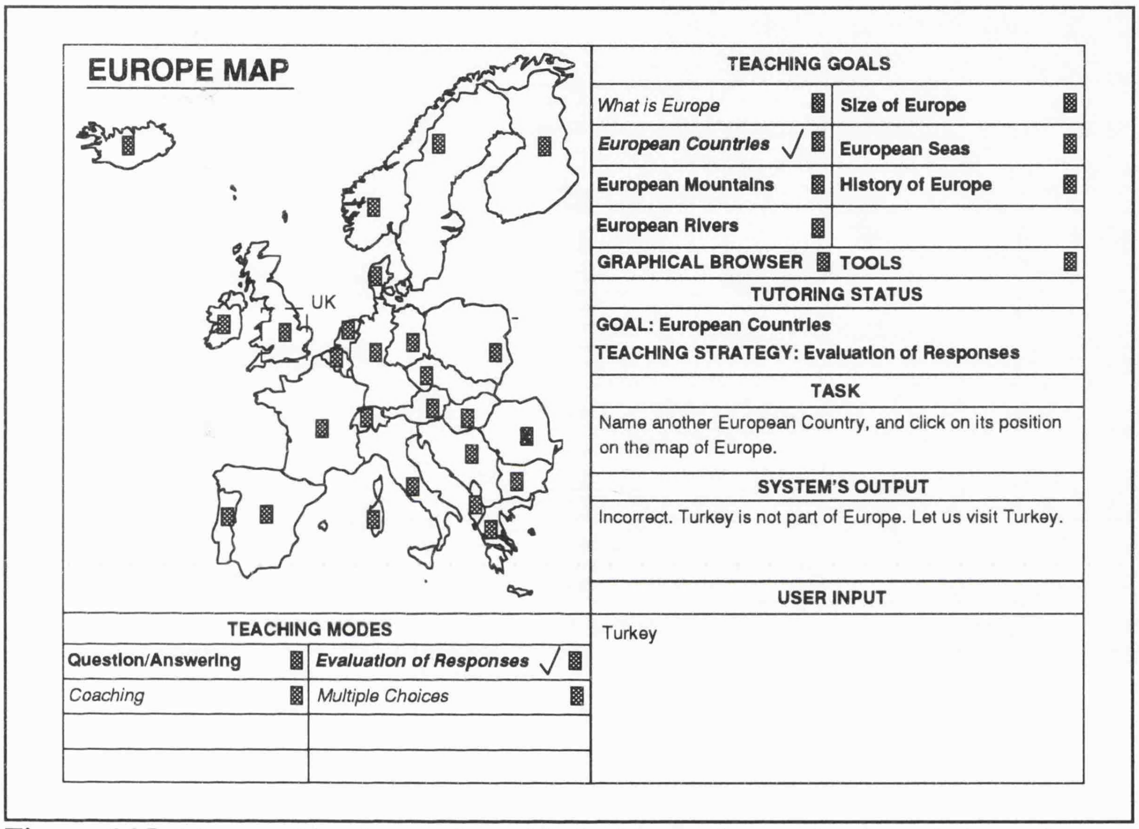


Figure 6.24: The current state of the student overlay knowledge model

Figures 6.23 and 6.24 the hypertext information links within the student model carry an overlay statistic that represents the level of mastery of a concept or a relationship.

Once back in the Europe Hyperframe the pedagogical process model updates the user interface. For this purpose, it calls the tutor knowledge process model to display the teaching goals that have to be attained with the current instruction knowledge unit, the teaching strategies that are appropriate for tutoring with these goals, the goal to be attained, "European countries", and the teaching strategy that is to be applied to attain this goal. It then calls the hypertext engine to provide to the user interface for display any annotated graphics or text to the domain hyperframe. Then, according to the context of the teaching strategy, it prompts the user for input. In Figure 6.25 below, the teaching strategy that has been selected by the tutor knowledge process model and retrieved by the hypertext engine is that of the "Evaluation of student response". With this strategy the user is asked to state what he knows about the goal, so the prompt, "Name another European country, and click on its position on the Map of Europe". The user in the case, answers Turkey, which is incorrect thus the system's output is appropriately given. See Figure 6.25.



TEACHING GOALS

What is Europe ▨	**Size of Europe** ▨
European Countries ✓ ▨	**European Seas** ▨
European Mountains ▨	**History of Europe** ▨
European Rivers ▨	
GRAPHICAL BROWSER ▨	**TOOLS** ▨

TUTORING STATUS

GOAL: European Countries
TEACHING STRATEGY: Evaluation of Responses

TASK

Name another European Country, and click on its position on the map of Europe.

SYSTEM'S OUTPUT

Incorrect. Turkey is not part of Europe. Let us visit Turkey.

USER INPUT

Turkey

TEACHING MODES

Question/Answering ▨	***Evaluation of Responses*** ✓ ▨
Coaching ▨	*Multiple Choices* ▨

Figure 6.25: Man-machine interaction with the Europe instruction knowledge unit

As before, the pedagogical process model co-ordinates a number of activities, in order to arrive at the system's outcome. First, it calls the hypertext engine to follow the links from the teaching goal to the corresponding attribute slots in the domain hyperframe, and then calls the domain expert knowledge process model to retrieve the contents of these attribute slots. Then it calls the student knowledge process model to compare the user input with the knowledge retrieved from the domain hyperframe. In this case, the user input may be incorrect since is not included in the "countries" attribute slot of the domain hyperframe.

In this case, the pedagogical process model calls the hypertext engine and the domain expert knowledge process model to check which continent Turkey is part of. The hypertext engine issues a keyword search for a hyperframe called "Turkey Frame". If we assume that there is one, the hypertext engine will follow links to this hyperframe and the domain expert knowledge process model will retrieve the contents of the part-of attribute slot. In this case, the domain expert knowledge process model will retrieve that Turkey is part of Asia. This information will confirm that the student knowledge process model diagnosed a misconception.

In this case, the student knowledge process model will place Turkey in the "countries" attribute slot of the student Europe hyperframe but marked as a misconception. The student knowledge process model using the user input, Turkey, and Europe as its data will call the expert system to traverse the bugs library and select a mal-rule that describes the nature of the misconception and that also provides details of the remedial action to be taken. In this case, it will select "mal-rule 58: Student thinks Turkey is part of Europe". The student knowledge process model calls for tutoring with the instruction knowledge unit on Turkey. Therefore, the pedagogical process model will temporarily break away from its didactic plan of action and pursue tutoring with the Turkey unit.

To achieve this, the pedagogical process model, will call the hypertext engine to create a referential link from the misconception value "Turkey" in the "countries" attribute slot of the student Europe frame to the student Turkey frame. The name of the referential link will be the name of the rule that proves the misconception, "Student thinks Turkey is part of Europe". Also an overlay statistic of -2 will be attached to this link to denote a misconception. Figure 6.26 represents the current state of the student overlay knowledge model after the diagnosis and representation of the misconception. Once this opportunistic tutoring with the Turkey unit is over the didactic operation will resume its default didactic plan of action.

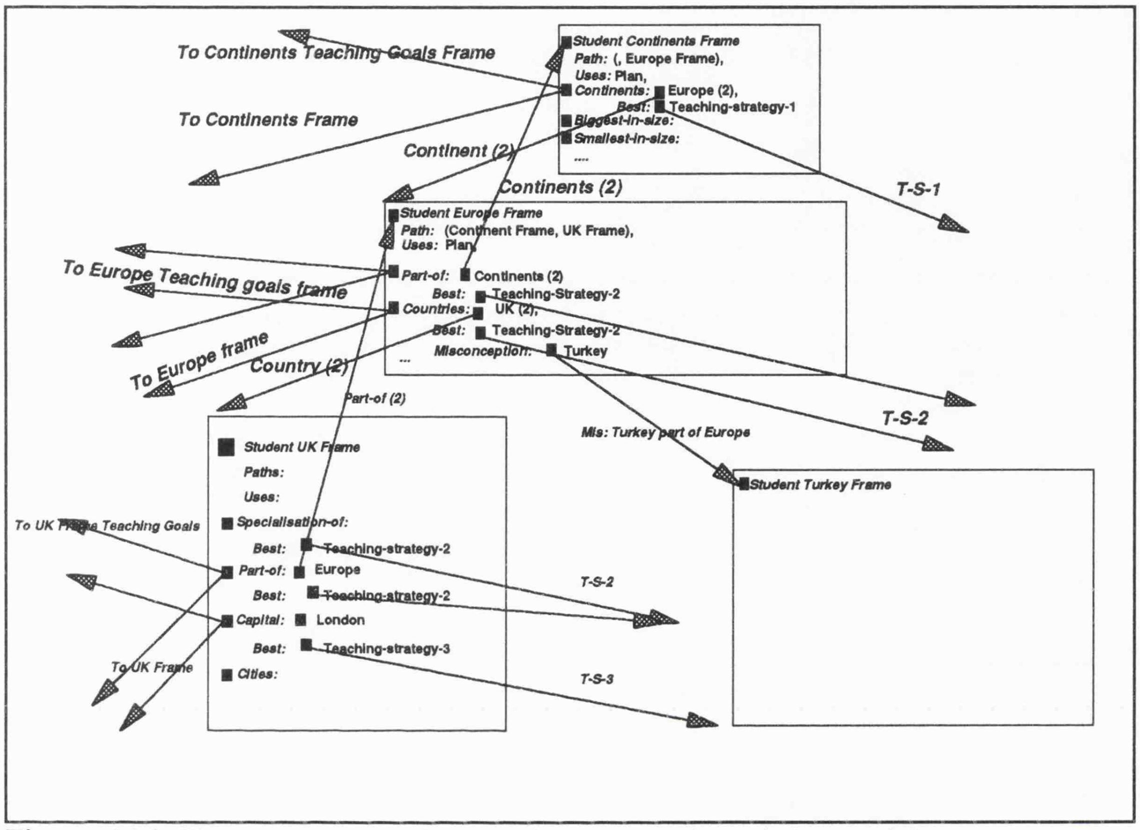


Figure 6.26: The current state of the student overlay knowledge model

6.5.1 An interaction protocol involving the tutoring system's generative behaviour

Let us assume that at some stage of the man-machine interaction, the system's generative ability has been invoked as a result of either the student being classified such that the tutoring system allows him to explore the environment or the system is testing the user after the completion of tutoring with a group of units, for instance, Europe and its constituents which results to Figure 6.10.

The next task of the pedagogical process model is to create the rest of the instruction unit for the "German Speaking European Countries Frame". It calls the hypertext engine to create a semi-structured hypertext information node which it names "German Speaking

European Countries Frame Teaching Goals". It then makes this hyperframe part of the Europe Frame Teaching Goals hyperframe by creating a "Part-of" attribute slot in it and placing Europe as the value and by creating an organisational information link to the Europe Frame Teaching Goals. It then creates an attribute slot called "Goal-1" in which it places "German Speaking European Countries (3)" as the value/goal and links the attribute slot with a referential link to the "Countries" attribute slot in the generated domain hyperframe.

The pedagogical process model then calls the student knowledge process model to retrieve the name of the overall best teaching strategy for the student. It then calls the hypertext engine to include the name of this teaching strategy in the Goal-1 attribute slot of the teaching goals frame, and also create a referential link from it to the hypertext node that contains the teaching strategy. The pedagogical process model also retrieves from the Europe map the maps of the three countries and annotates these to the generated domain hyperframe as a single map. Finally it creates, the corresponding student hyperframe, free of values. The resulting instruction knowledge unit on German speaking European Countries is given in Figure 6.27 below.

The next task of the pedagogical process model is to update the user interface. For this purpose, it calls the tutor knowledge process model to provide to the user interface for display the teaching goals that have to be attained with the current instruction knowledge unit, the teaching strategies that are appropriate for tutoring with these goals, the first (and only) goal to be attained, "German Speaking European countries", and the teaching strategy that is to be applied to attain this goal. It then calls the hypertext engine to provide to the user interface for display the annotated to the domain hyperframe graphics

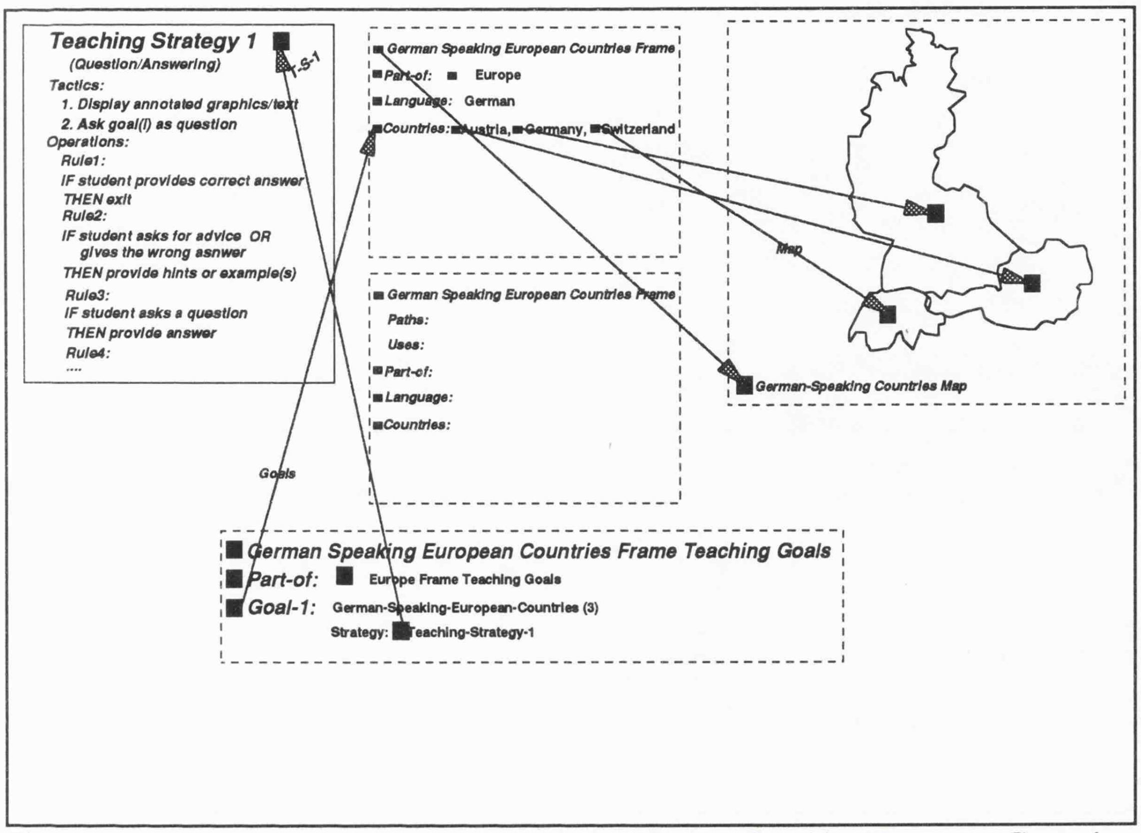
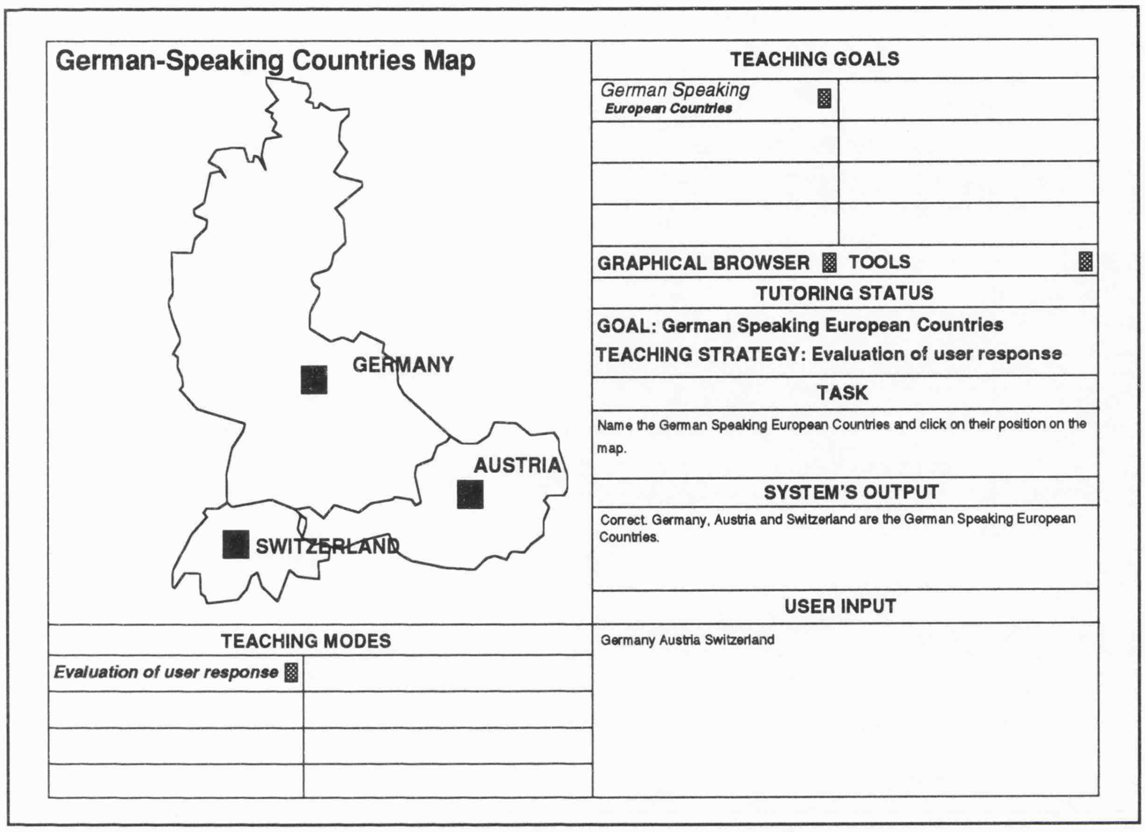

Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(i) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
....
T-S-1
German Speaking European Countries Frame
Part-of: Europe
Language: German
Countries: Austria, Germany, Switzerland
German Speaking European Countries Frame
Paths:
Uses:
Part-of:
Language:
Countries:
Map
German-Speaking Countries Map
Goals
German Speaking European Countries Frame Teaching Goals
Part-of: Europe Frame Teaching Goals
Goal-1: German-Speaking-European-Countries (3)
Strategy: Teaching-Strategy-1

Figure 6.27: The instruction knowledge unit for German Speaking European Countries

and text. Then, according to the context of the teaching strategy, it prompts the user for input. In Figure 6.28 below the teaching strategy that has been selected by the tutor knowledge process model and retrieved by the hypertext engine is that of the "Evaluation of student response". With this strategy the user is asked to state what he knows about the goal, so the prompt, "Name the German Speaking European countries", and click on their position on the Map". The user in the case, names correctly the three countries and clicks correctly with a pointing device on their position on the map.

As before, the pedagogical process model co-ordinates a number of activities, in order to arrive at the system's outcome. First, it calls the hypertext engine to follow the links from the teaching goal to the corresponding attribute slots in the domain hyperframe, and then calls the domain expert knowledge process model to retrieve the contents of these attribute slots. Then it calls the student knowledge process model to compare the user input with



German-Speaking Countries Map

TEACHING GOALS	
German Speaking European Countries ▨	

GRAPHICAL BROWSER ▨ TOOLS ▨

TUTORING STATUS

GOAL: German Speaking European Countries
TEACHING STRATEGY: Evaluation of user response

TASK

Name the German Speaking European Countries and click on their position on the map.

SYSTEM'S OUTPUT

Correct. Germany, Austria and Switzerland are the German Speaking European Countries.

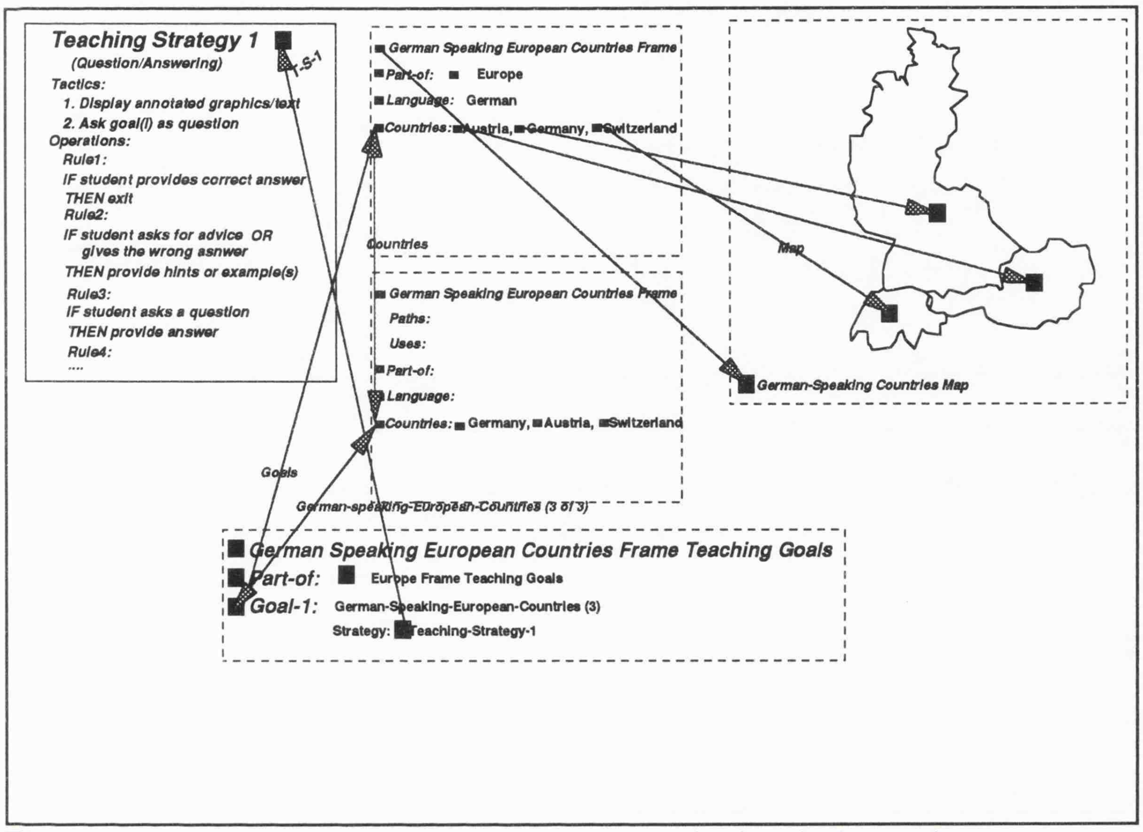
USER INPUT

Germany Austria Switzerland

TEACHING MODES	
Evaluation of user response ▨	

Figure 6.28: Man-machine interaction with the generated unit

the knowledge retrieved from the domain hyperframe. In this case, the user input is correct, and therefore, the student knowledge process model fills the corresponding attribute slots in the student hyperframe. The attribute slot "Countries" in the student hyperframe is filled with "Germany, Austria, Switzerland". The hypertext engine, as before, is called by the pedagogical process model to create a referential bi-directional link from the teaching goal that has been attained in the German Speaking European Countries teaching goals hyperframe, to the corresponding attribute slot in the student hyperframe (see Figure 6.29). It also creates a referential link from the student attribute slot that has been filled with knowledge, to the corresponding domain hyperframe attribute slots. Finally, since there are domain hyperframes for the three values which the student attribute slot has been filled with, the hypertext engine creates referential links to these from the corresponding values in the student hyperframe "countries" attribute slot. Before leaving the unit, the student knowledge process model fills the "path" and uses attribute



Teaching Strategy 1
(Question/Answering)
Tactics:
1. Display annotated graphics/text
2. Ask goal(i) as question
Operations:
Rule1:
IF student provides correct answer
THEN exit
Rule2:
IF student asks for advice OR gives the wrong asnwer
THEN provide hints or example(s)
Rule3:
IF student asks a question
THEN provide answer
Rule4:
T.S-1
German Speaking European Countries Frame
Part-of: Europe
Language: German
Countries: Austria, Germany, Switzerland
Countries
German Speaking European Countries Frame
Paths:
Uses:
Part-of:
Language:
Countries: Germany, Austria, Switzerland
German-speaking-European-Countries (3 of 3)
Goals
Map
German-Speaking Countries Map
German Speaking European Countries Frame Teaching Goals
Part-of: Europe Frame Teaching Goals
Goal-1: German-Speaking-European-Countries (3)
Strategy: Teaching-Strategy-1

Figure 6.29: The generated instruction knowledge unit after the interaction

slots in the student hyperframe.

Chapter 6 showed how to use hyperframes to design a generic model of an Intelligent Tutoring System with a full-scale didactic operation. The model caters for the design of an open and scalable Decision Base that allows for a variety of system components, like domain, student and tutoring knowledge, to be combined into a single model.

CHAPTER 7: SUMMARY AND CONCLUSIONS

This chapter summarises the research presented in this thesis and discusses its contributions to knowledge. It also describes a number of consequences of the research, linking them to ideas for further research.

7.1 SUMMARY

This thesis investigates architectures embracing three knowledge models: that of the domain, the student and the tutor, that make up an Intelligent Knowledge Based Tutoring System. In particular it investigates the interrelatedness and interconnectedness of the three knowledge models.

Chapter 1 overviews these knowledge models by explaining what they are, and the purpose they serve during the course of interaction and what techniques have been used to implement them in Intelligent Tutoring Systems. The investigation reveals that the vast majority of Intelligent Tutoring Systems in the past decade have been developed as knowledge based systems. As a result, the examination of interrelatedness and interconnectedness between the three knowledge models are in the context of Intelligent Knowledge Based Tutoring Systems.

Chapter 2 presents Wenger's model of a didactic operation which provides a framework in which the interrelatedness and interconnectedness of the three knowledge models is examined. The model of didactic operations assumes firstly the existence of domain, student and tutoring knowledge which constitute, along with their equivalent process models, the system's Decision Base. Secondly, the organisation, structure and nature of the domain knowledge and the presence of a student model assumes a target level for the didactic operation either at the behavioural level, the epistemic level or the individual level or some combination of these levels. Thirdly, the underlying organisation, structure and nature of the domain and tutoring knowledge, and the target level of the didactic operation, conjecture a pedagogical context for the application of the didactic operation (i.e. the context and the nature of the man-machine interaction). Finally, the underlying organisation, structure and nature of the domain and tutoring knowledge and the diagnostic ability of the tutoring system constitute an explicit didactic plan of action which defines the flow of tutorial interaction.

This model does not explicitly state what the nature of interaction between the three components should be. It only serves to explain the behaviour of an Intelligent Tutoring System that follows a full-scale didactic operation. At this stage the thesis suggests that to continue the investigation, an evaluation strategy that examines the relationship between such a system behaviour and its architecture is required. This would unravel the requirements for interrelatedness and interconnectedness between the three knowledge models in the context of the didactic operation. This calls for a study of existing knowledge based tutoring systems in which the relationship between their behaviour and architecture with respect to the didactic operation is examined.

Chapter 3 introduces two of the very few available Knowledge Based Tutoring Systems, namely PROUST and micro-SEARCH. These are used as pilot systems in the study. The Chapter gives a detailed account of their architecture and resulting functionality.

Chapter 4 presents an evaluation of PROUST and micro-SEARCH. By attempting to answer the question: *What should the relationship between a system's behaviour and its architecture be with respect to the didactic operation?* the evaluation of the two systems highlights four requirements about the interrelatedness and interconnectedness between the three knowledge models with respect to a full-scale didactic operation:

[1]. The system incorporates all three knowledge models.

[2]. The three knowledge models be independent but need to reference information within each other.

[3]. The system may branch the student anywhere in the domain knowledge structure as part of an alternative didactic plan of action.

[4]. The system has the ability to create additional domain knowledge from its existing domain knowledge and therefore establish additional didactic plans of action.

When these requirements are translated into a Knowledge Based Tutoring Systems context, they yield an equal number of requirements for the development of an Intelligent Knowledge Based Tutoring System with a full-scale didactic operation:

[1]. The system incorporates domain, student and tutoring knowledge representations.

[2]. There are explicit and direct links within, and between related knowledge parts of, the three knowledge representations.

[3]. The links include both hierarchical and non-hierarchical links.

[4]. The system is able to generate additional domain knowledge from, and link this to, its existing domain knowledge representation.

Finally Chapter 4 discusses a number of limitations with the knowledge based systems approach to developing a tutoring system with a full-scale didactic operation:

[1]. Knowledge decomposition, representation and inferencing is exclusively hierarchical. This conflicts with the third requirement.

[2]. The reasoning mechanism requires that all necessary knowledge be encoded prior to interaction. This conflicts with the fourth requirement.

[3]. A single viewpoint of the domain knowledge is inflicted on the user since reorganising the knowledge base during the course of interaction from another viewpoint is not yet feasible. This conflicts with the fourth requirement.

[4]. Knowledge based systems lack explicit information linking since all relationships are established through reasoning. This conflicts with the second requirement.

The last limitation raises a serious problem with respect to the interconnectedness of the knowledge models. For instance, how can the relationship between the student knowledge in the student model and the equivalent in the domain model or the relationship between a teaching goal and the equivalent knowledge in the domain model be directly described? Or how does one represent non-hierarchical and thus non-inferentiable relationships established in the student's knowledge?

Explicit hierarchical and non-hierarchical information linking and consequently generative behaviour are considered to be the foremost advantages of hypertext. Nevertheless, hypertext on its own does not constitute a framework for developing an Intelligent

Tutoring System because it lacks the logical inferencing mechanisms provided by Artificial Intelligence Techniques. Recent research and development on Artificial Intelligence has focused on hybrid models that are made up of Artificial Intelligence and Hypertext. These models utilise hypertext's hierarchical and non-hierarchical information linking abilities with Artificial Intelligence's logical inferencing techniques. Although none of these models have been proposed for Intelligent Tutoring Systems Development, Chapter 5 discusses various such hybrid models and proposes a new model, Hyperframes, that integrates Minsky's Frames with Hypertext's information nodes and links, and which promises to overcome the limitations of the Knowledge Based Tutoring Systems, is introduced.

Chapter 6 shows how to use hyperframes to design a generic model of an Intelligent Tutoring System with a full-scale didactic operation. The model caters for the design of an open and scalable Decision Base that allows for the system components: domain, student and tutoring knowledge, to be combined into a single model.

7.2 CONCLUSIONS

This thesis contributes the following to the field of Intelligent Tutoring Systems:

(a) The requirements for the development of an Intelligent Knowledge Based Tutoring System with a full-scale didactic operation:

- (i) The system incorporates domain, student and tutoring knowledge representations.
- (ii) There are explicit and direct links within, and between related knowledge parts of, the three knowledge representations.

(iii) The links include both hierarchical and non-hierarchical links.

(iv) The system is able to generate additional domain knowledge from, and link this to, its existing domain knowledge representation.

(b) Hyperframes: A knowledge representation scheme that integrates Minsky's Frames with Hypertext's information nodes and links.

(c) A (scalable and open) generic model for the architecture of an Intelligent Knowledge Based Tutoring System with a full-scale didactic operation.

The last two contributions are discussed in greater detail in the following two sections.

7.2.1 Hyperframes

The concept of a "Hyperframe" is the basis for a solution to the shortcomings of a knowledge based systems approach to developing an Intelligent Tutoring System with a full-scale didactic operation. A hyperframe integrates Minsky's Frames and Hypertext's nodes and links. With this alternative knowledge representation scheme, a frame with its attribute slots and values is stored in a semi-structured hypertext information node, where the node's labelled fields are made to be the frame's attribute slots and the labelled fields' values are made to be the values of the frame's attribute slots.

Hyperframe to hyperframe relationships are installed by explicit hypertext information links. An organisational hypertext information link denotes a hierarchical relationship between two hyperframes and a referential hypertext information link denotes a non-hierarchical relationship between two hyperframes. A keyword link links two hyperframes which share the same value for an attribute slot. Any information which is relevant to a hyperframe but could not be included in the hyperframe may be annotated with an annotation link to the hyperframe as a typed or graphical hypertext information node. From within this annotated node there may be referential, keyword or annotation links to other hyperframes. The names of the hypertext information links carry semantic information.

This knowledge representation scheme allows three reasoning strategies: logical inferencing, hypertext information retrieval, or a combination of both. Logical inferencing can support inheritance and defaulting of slot values for a hyperframe and execution of any attached procedural attachments by examining other hyperframes that the given hyperframe is hierarchically linked to. The hypertext engine is responsible for inserting the value in the appropriate attribute slot and for creating any information links between the slot value and any related hyperframes. Since the system can support computed hypertext information links (and nodes) through the hypertext engine, there is no need to generate all hypertext information links (and nodes) prior to interaction. Hypertext information retrieval can support browsing through hypertext information nodes by pursuing hypertext information links (all kinds of links as opposed to hierarchically only supported by logical inferencing) and retrieval of information from these nodes for a given node (i.e. a hyperframe). The reasoning strategy may be a blend of hypertext information retrieval and logical inferencing, with the hypertext engine establishing a path of hyperframes by pursuing organisational, referential and keyword links from a given hyperframe and the logical inferencing either activating automated inferencing procedures to infer values or executing procedural attachments, otherwise known as demons.

The application of different hypertext information links settles the first shortcoming of a knowledge based systems approach in developing an Intelligent Tutoring System with a full-scale didactic operation (i.e. exclusively hierarchical knowledge decompositions, representations and inferencing with the domain knowledge). Organisational links set up inheritance hierarchies, but all other links set up non-hierarchical relationships, and thus provide material for non-hierarchical reasoning strategies. Non-hierarchical reasoning eliminates the need to perform one-way logical inference searchings through the entire tree or network.

The support for computed links and nodes which this knowledge representation scheme supports, is the answer to both the second and third shortcomings of a knowledge based systems approach (i.e. a complete knowledge base of facts from a single viewpoint, and generative behaviour). Computed links and nodes constitutes the scheme's generative ability which removes the necessity for a complete knowledge base of facts since additional information or hyperframes can be generated during the course of interaction. At the same time, the system's generative ability can be used to sustain alternative viewpoints to the domain knowledge, not by breaking the hierarchical structure and reorganising the knowledge base, but by retrieving information and creating other hyperframes.

The use of hypertext information links, which are exclusively explicit, settles the fourth limitation of a knowledge based systems approach (i.e. exclusively implicit information linking in the domain knowledge base). As a result, information links carry semantic information which also eradicates the need to perform logical reasoning in order to infer any direct, at least, relationships between related parts in a knowledge base or related knowledge bases.

7.2.2 A (scalable and open) generic model for the architecture of an Intelligent Knowledge Based Tutoring System with a full-scale didactic operation

This section describes the second contribution of the thesis which is a (scalable and open) generic model for the architecture of an Intelligent Knowledge Based Tutoring System that fully supports a didactic operation. The generic model to be described in this section has been developed to overcome the problems that arise when a knowledge based systems approach is used to develop an Intelligent Tutoring System with a full-scale didactic operation. The solution has been developed from an understanding of the nature of the requirements for the development of such an Intelligent Tutoring System. In doing so, the research has emphasised the need to design around likely problems of the knowledge based systems approach in developing intelligent tutoring systems, rather than simply responding to these problems with more expert systems code.

The Decision Base

To satisfy the first two requirements, the Decision Base is designed on the basis that all three kinds of knowledge are kept separately as three distinct knowledge bases. Hyperframes is the knowledge representation technique that is used to represent the knowledge in each and every one of these knowledge bases. The contents of the student and the tutoring knowledge base are determined by the context of the domain knowledge base.

The domain knowledge in the domain knowledge base is decomposed from a default viewpoint into different hierarchical and non-hierarchical domain knowledge units whose level of knowledge detail depends on their position in the inherent hierarchical domain structure. Each knowledge unit in this hierarchical decomposition is represented using a hyperframe. A domain hyperframe is linked with other hyperframes via different hypertext information links: Organisational (for hierarchical relationships), referential (for non-hierarchical relationships), keyword (if they share the same attribute) and annotation (for example, graphs). Links carry names that denote the relationship. The resulting network may be declarative but it does not assume any particular order for traversal.

The tutoring knowledge in the tutoring knowledge base is comprised of a set of teaching goals and strategies for each domain hyperframe. The teaching goals for each domain hyperframe are stored in a hyperframe and the teaching strategies which are rule-based implementations of known teaching strategies are stored in "typed" hypertext information nodes. The typed hypertext information nodes containing the teaching strategies do not form any kind of hierarchy nor do they contain any form of hypertext information links to any other hypertext information nodes. However, a teaching goals hyperframe is linked with other teaching goals hyperframes via the following hypertext information links: Organisational information links for hierarchical relationships and annotations from each teaching goal to those teaching strategies that are suitable for helping a student to attain the goal.

The student knowledge in the student knowledge base is created during interaction. For each domain hyperframe, a student hyperframe is built as an overlay model of the domain hyperframe including diagnosed misconceptions. The context of a student hyperframe is a subset or at its best an exact copy of the corresponding domain hyperframe. In addition to this purely epistemic knowledge, the student model also contains behaviour knowledge such as paths to and from a student hyperframe, the reasons why the path was followed, etc. Similarly, hypertext information links to and from a student hyperframe are also computed during interaction. A student hyperframe may be linked with other student hyperframes via different hypertext information links: Organisational (for established hierarchical relationships), referential (for established non-hierarchical relationships and misconceptions), keyword (for hyperframes sharing the same information) and annotations to the best teaching strategies. Link names carry either an overlay statistic or they denote a misconception if the link was set up for this purpose.

To satisfy the interconnectedness suggested by the second requirement, the three knowledge models for each domain knowledge unit are integrated in the concept of an instruction knowledge unit. The tutoring system is a collection of instruction knowledge units each of which holds domain knowledge, student knowledge and teaching goals and associated domain-independent teaching strategies suitable for tutoring with the unit's domain knowledge. Each instruction knowledge unit has access to the system's bug library of common bugs or misconceptions.

The instruction knowledge unit is implemented through the use of hypertext information links that link together the unit's three hyperframes: domain, student and teaching goals and associated teaching strategies and annotated nodes containing graphs, animations, etc. in addition to the hypertext information links that have already been used to set up the three knowledge structures independently. During instruction delivery, the system takes the student-user through different sequences of instruction knowledge units. Which instruction knowledge unit or group of instruction knowledge units to visit next is determined on the basis of the default didactic plan of action or any diagnosed misconceptions.

Within an instruction knowledge unit, every teaching goal is linked to the corresponding knowledge in the domain hyperframe which is necessary to meet the goal, to those teaching strategies that are suitable for tutoring with the goal, and to those parts of the student knowledge hyperframe that satisfy teaching goals. From established knowledge in the student hyperframe there are links to the corresponding knowledge in the domain hyperframe as part of the student overlay model, to the teaching goals that are satisfied with acquired knowledge and to that teaching strategy that proved to be the most effective for meeting a teaching goal. Should another model other than the domain, student or tutoring model be included in the instruction knowledge unit, then the principle of integrating this with existing unit models is the same as with the other models.

The use of hypertext information links between hyperframes allows for various hierarchical and non-hierarchical explicit paths to be established through the domain network. Hypertext information links are by default explicit since they carry a name which also denotes what the relationship is between the concepts it links, and can be made visually explicit to the user, for instance as link icons. The nature and use of hypertext information links satisfies the third requirement.

With respect to generative ability posed by the last requirement, pursuing the various hypertext information links in the three knowledge structures allows the system to generate alternative paths or additional domain knowledge parts as domain hyperframes, from different viewpoints. It has been demonstrated that the system is already capable of generating student hyperframes, which in effect are some form of domain knowledge. The generative ability eases the need for a complete domain knowledge base and also allows knowledge base restructuring from alternative viewpoints. A further use of hypertext information links is to link generated domain hyperframes to the existing domain knowledge base. In addition to generating a domain hyperframe, the system will also generate the equivalent student and teaching goals hyperframes and decide on suitable teaching strategies.

The generative ability increases the range of issues the system can provide tutoring on, solves the problem of having to design additional instructional material during interaction, eliminates the need for a complete domain hyperframe, has virtually unlimited resource material and allows the system and the student-user to generate as much material as necessary to accomplish the educational objectives set by the instructional designer.

The Didactic Plan of Action

The default, but explicit, didactic plan of action is what is defined by the organisational and hierarchical structure of the domain knowledge. The plan involves taking the student through instruction knowledge units by following the organisational links of the domain knowledge. The pedagogical process model working in synergy with the domain model defines how this hierarchical tree of instruction knowledge units is to be traversed, for instance breadth-first, depth-first, best-first, etc. This defines the teaching curriculum and the student's learning path through it.

In each instruction knowledge unit the pedagogical process model generates a didactic episode in which the goals in the teaching goals hyperframe must be fulfilled by applying one of many teaching strategies on the equivalent knowledge contained in the domain hyperframe. The corresponding student hyperframe is created by the hypertext engine along with any necessary hypertext information links. When a misconception is diagnosed in the user input, the pedagogical process model may break away from the "default" plan of action to assume a remedial plan of action, if this is what is the action suggested for remedying the bug. This may involve taking the student to an instruction knowledge unit that is not part of the default didactic plan. The hypertext engine creates the appropriate link to this instruction knowledge unit and tutoring proceeds in the context of it. In another instance, the pedagogical process model may break away from its default didactic plan of action, by pursuing non-organisational information links.

The didactic operation has another two tasks to perform: First, to signal the end of tutoring and thus leave the student with the choice to quit or explore this information-rich environment at his own pace and second, to classify the user as a learner according to his performance. Because of the generative ability, discovery learning may continue, since there can be many viewpoints to be exploited. With respect to classifying the user, the degree of freedom allowed to the user-learner may be defined accordingly.

The Pedagogical Context

The default pedagogical context in this program design is plan-based, since the hierarchical planning of domain knowledge and the teaching goals drive interaction. However, the student knowledge process model may trigger such interventions as a result of some student diagnosis of misconceptions which may cause the pedagogical process model to call for, and monitor an opportunistic plan of action. Pursuing opportunistic plans may result from the student directly using the system's generative ability or from switching between different teaching strategies in the context of an instruction knowledge unit. The freedom to pursue opportunistic plans will be gradually given as the student moves up the classification ladder.

The Target Level

The objective of the didactic operation is to modify the student's knowledge state through either direct communication of knowledge or practice which makes the target level of the didactic operation epistemic. Nevertheless, a student hyperframe includes behavioural and individual aspects of the user-learner, like pathways to and from an instruction unit, uses of the instruction unit, best teaching strategy, applied teaching strategies and explanation requests.

7.3 FUTURE RESEARCH AND DEVELOPMENT

The thesis has shown the possibility of using Hypertext to support Artificial Intelligence techniques in developing an Intelligent Tutoring System that supports a full-scale didactic operation. As was noted earlier, the model, being generic, can incorporate more knowledge models in order to increase the system complexity. One such component can be a multimedia element to enable electronic information in various modes like images, text, data, video and sound, simulation and animation to be combined in new interactive applications where appropriate. As was explained in Chapter 6, the thesis does not attempt to promote a particular design methodology nor propose a new one. One area for further research and development would be to examine the validity of using different conventional design methodologies to develop an Intelligent Tutoring System using the generic model.

A pragmatic reason for choosing a hybrid model to implement the generic model is that on the one hand, existing hypertext tools such as Hypercard II, Guide III or even KnowledgePro could not support the development of the generic model on their own because they are exclusively hypertext-oriented tools and they lack the logical inferencing abilities of Artificial Intelligence techniques. On the other hand, it may be a waste of time to attempt to redesign existing knowledge based systems to deal with explicit information linking when there are already tools that perform this function very efficiently. Therefore, one area for further development would be tools that facilitate logical inferencing with hypertext knowledge representations. The direction this research may take could be to develop an Intelligent Tutoring System both as a Knowledge Based Expert System and as a Hypertext System, and then perform a comparative evaluation of the two systems in order to uncover the advantages and shortcomings of both approaches which would highlight the areas for potential integration.

Although the hyperframe model contributes an alternative knowledge representation scheme, the thesis has not been concerned with studying different knowledge representations because none of the existing knowledge representation schemes cater for explicit hierarchical and non-hierarchical information linking. Logical inferencing with all knowledge representation schemes involves traversing a hierarchical tree or network in order to establish a relationship. Any attempts to endow existing knowledge representations schemes with explicit information linking abilities would be a waste of time because they would result in re-inventing hypertext. Rather, further research and development needs to focus on endowing hypertext tools with logical inferencing facilities beyond keyword searching. The reason why Minsky's Frames have been selected for integration with hypertext instead of some other knowledge representation technique is because of the similarity of their representation with semi-structured hypertext information nodes.

The knowledge of the domain of discourse, in the context of which the generic model is discussed in Chapter 6, is declarative, therefore suggesting that the generic model may be applicable only for expository tutors. Further research be undertaken on the application of the generic model to procedural knowledge. Procedural knowledge can be subdivided into two subcategories: flat and hierarchical. Hierarchical representations allow for subgoaling, so for example, if the goal is to win a game of chess, a subgoal may be to take the opponent's Queen, a subgoal of which is to make a certain piece move. This procedural knowledge may then be represented as a hierarchical tree where each branch of the tree is thought of as a skill which the user-learner has to acquire and which may, as in the above example, be decomposed into subskills. Flat representations can also be thought of as a hierarchical tree of a single level, where each and every task on this level, although independent from the other tasks, contributes towards acquiring a certain skill (e.g. arithmetic subtraction skills). With procedural knowledge the pedagogical process model may be more strict regarding tree traversal because of the order of skill and subskill acquisition.

Furthermore, providing the full context of the rule bases that denote the bugs library and the teaching strategies that the system may use or the precise conditions for the didactic and opportunistic plans that the system may pursue is beyond the scope of this thesis. Nevertheless, investigating different plan-based or opportunistic strategies in relation with the generic model may serve as an area for further research and development.

Finally, addressing the problem of the authoring of instructional material for either the proposed system or for any of the systems discussed is beyond the scope of this thesis. However, investigating authoring either in the context of the proposed system or in more general terms may serve as an area for further research and development. As Nielsen

[1990a] argues that unlike conventional Knowledge Based Systems, one of the greatest advantages of current hypertext tools is the ease of authoring hypertext material.

BIBLIOGRAPHY

Alessi S.M. and Trollip S.R. (1985): *Computer-Based Instruction: Methods and Development*, USA: Prentice-Hall International.

Anderson J.R. (1988): The Expert Module, in *Foundations of Intelligent Tutoring Systems*, Polson M.C. and Richardson J.J. (eds.), USA: Lawrence Erlbaum Associates.

Anderson J.R. (1989): A Theory of the Origins of Human Knowledge, *Artificial Intelligence* 40(1-3).

Anderson J.R., Boyle C.F., Corbett A.T. and Lewis M.W. (1990): Cognitive Modelling and Intelligent Tutoring, in *Artificial Intelligence and Learning Environments*, Clancey W.J. and Soloway E. (eds.), USA: MIT Press.

Anderson J.R., Boyle C.F. and Reiser B.J. (1985): Intelligent Tutoring Systems, *Science* 228(4698).

Anderson J.R., Boyle C.F. and Yost G. (1985): The Geometry Tutor, in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Altos, USA: Morgan Kaufmann.

Anderson J.R. and Reiser B.J. (1985): The Lisp Tutor, *Byte* 10(4).

Angelides M.C. and Doukidis G.I. (1990): In there a place in OR for Intelligent Tutoring Systems?, *Journal of the Operational Research Society* 41(6). "Reprinted in *Artificial Intelligence in Operational Research*, G.I. Doukidis and R.J. Paul (eds.), UK: McMillan 1992".

Baker M. (1990): Arguing with the Tutor, in *Guided Discovery Tutoring: A Framework for ICAI Research*, Elsom-Cook M. (ed.), UK: Paul Chapman Publishing.

Begeman M.L. and Conklin J. (1988) The Right Tool for the Right Job, *Byte* 13(10).

Begoray J.A. (1990): An Introduction to hypermedia issues, systems and application areas, *International Journal of Man-Machine Studies* 33(2).

Bielawski L. and Lewand R. (1991): *Intelligent Systems Design: Integrating Expert Systems, Hypermedia and Database Technologies*, USA: John Wiley and Sons Ltd.

Bonar J.G. (1991): Interface Architectures for Intelligent Tutoring Systems, in *Intelligent Tutoring Systems: Evolutions in Design*, Burns H.L., Parlett J.W. and Redfield C.L. (eds.), USA: Lawrence Erlbaum Associates.

Brecht B. and Jones M. (1988): Student Models: The Genetic Graph Approach, *International Journal of Man-Machine Studies* 28(5).

Brown J.S. (1985): Process versus product: A perspective on tools for communal and informal electronic learning, *Journal of Educational Computing Research* 1(2).

Brown J.S. (1990): Towards a New Epistemology for Learning, in *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, Frasson C. and Gauthier G. (eds.), USA: Ablex Publishing.

Brown J.S. and Burton R.R. (1975): Multiple representation of knowledge for tutorial reasoning, in *Representation and Understanding: Studies in Cognitive Science*, Bobrow D. and Collins A. (eds.), USA: Academic Press.

Brown J.S. and Burton R.R. (1978): A paradigmatic example of an artificially intelligent instructional system, *International Journal of Man-Machine Studies* 10(3).

Brown J.S., Burton R.R. and Bell A.G. (1975): SOPHIE: a step towards a reactive learning environment, *International Journal of Man-Machine Studies* 7(5).

Brown J.S., Burton R.R. and deKleer J. (1982): Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), USA: Academic Press.

Brown J.S., Collins A. and Duguid P. (1991): Situated Cognition and the Culture of Learning, in *Artificial Intelligence and Education Volume 2: Principles and Case Studies*, Yazdani M. and Lawler R.W. (eds.), USA: Ablex Publishing Corporation.

Brown J.S. and VanLehn K. (1980): Repair theory: A generative theory of bugs in procedural skills, *Cognitive Science* 4.

Bumbaca F. (1988): Intelligent Computer Assisted Instruction: A Theoretical Framework, *International Journal of Man-Machine Studies* 29(3).

Burns H.L. and Capps C.G. (1988) Foundations of Intelligent Tutoring Systems: An Introduction, in *Foundations of Intelligent Tutoring Systems*, Polson M.C. and Richardson J.J. (eds.), USA: Lawrence Erlbaum Associates.

Burns H.L. and Parlett J.W. (1991): The Evolution of Intelligent Tutoring Systems: Dimensions of Design, in *Intelligent Tutoring Systems: Evolutions in Design*, Burns H.L., Parlett J.W. and Redfield C.L. (eds.), USA: Lawrence Erlbaum Associates.

Burton R.R. (1982): Diagnosing bugs in a simple procedural skill, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), USA: Academic Press.

Burton R.R. (1988): The Environment Module of Intelligent Tutoring Systems, in *Foundations of Intelligent Tutoring Systems*, Polson M.C. and Richardson J.J. (eds.), USA: Lawrence Erlbaum Associates.

Burton R.R. and Brown J.S. (1976): A tutoring and student modelling paradigm for gaming environments, in *Computer Science and Education*, Colman R. and Lorton P. Jr. (eds.), *ACM SIGCSE Bulletin* 8(1).

Burton R.R. and Brown J.S. (1982): An investigation of computer coaching for informal learning activities, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), USA: Academic Press.

Campbell B. and Goodman J.M. (1988): HAM: A General Purpose Hypertext Abstract Machine, *Communications of the Association of Computing Machinery* 31(7).

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

Carlson A. and Ram S. (1990): HyperIntelligence: The Next Frontier, *Communications of the ACM* 33(3).

Clancey W.J. (1982): Tutoring rules for guiding a case method dialogue, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), USA: Academic Press.

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

Clancey W.J. (1988): The role of qualitative models in instruction, in *Artificial Intelligence and Human learning: Intelligent Computer Assisted Instruction*, Self J. (ed.), UK: Chapman and Hall Computing.

Clancey W.J. and Soloway E. (1990): *Artificial Intelligence and Learning Environments*, USA: MIT Press.

Clegg C., Warr P., Green T., Monk A., Kemp N., Allison G. and Lansdale M. (1988): *People and Computers: How to evaluate your company's new technology*, in collaboration with Potts C., Fell R. and Cole I., Ellis Horwood Series in Computers and their Applications, UK: Ellis Horwood.

Conklin J. (1987): Hypertext: An Introduction and Survey, *IEEE Computer*.

Corbett A.T., Anderson J.R. and Patterson E.G. (1990): Student Modelling and Tutoring Flexibility in the Lisp Intelligent Tutoring System, in *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, Frasson C. and Gauthier G. (eds.), USA: Ablex Publishing.

Dede C. (1986): A review and synthesis of recent research in intelligent computer assisted instruction, *International Journal of Man-Machine Studies* 24(4).

Diaper D. and Rada R. (1991): Expertext: Hyperizing Expert Systems and Expertizing Hypertext, in *Hypermedia/Hypertext and Object-oriented Databases*, Brown H. (ed.), UNICOM Applied Information Technology 8, UK: Chapman and Hall.

Doukidis G.I., Angelides M.C. and Harlow J.L. (1988): Towards an Intelligent Tutoring System for Pascal Programming, *International Journal of Education and Computing* 4(4).

Doukidis G.I., Rogers R.A. and Angelides M.C. (1989): Developing a Pascal Tutoring Aid, *International Journal of Computers and Education* 13(4).

Doukidis G.I., Shah V.P. and Angelides M.C. (1988): *Lisp: From Foundations to Applications*, UK: Chartwell-Bratt.

Doukidis G.I. and Whitley E.A. (1988): *Developing Expert Systems*, UK: Chartwell-Bratt.

Duchastel P. and Imbeau J. (1988): Intelligent Computer-assisted Instruction (ICAI): Flexible Learning Through Better Student-Computer Interaction, *Journal of Information Technology* 3(2).

Duchastel P., Doublait S. and Imbeau J. (1988): Instructible ICAI, *Journal of Information Technology* 3(3).

Fiderio J. (1988): A Grand Vision, *Byte* 13(10).

Fink P.K. (1991): The Role of Domain Knowledge in the Design of an Intelligent Tutoring System, in *Intelligent Tutoring Systems: Evolutions in Design*, Burns H.L., Parlett J.W. and Redfield C.L. (eds.), USA: Lawrence Erlbaum Associates.

Foss C.L. (1989): Tools for reading and browsing Hypertext, *Information Processing and Management* 25(4).

Frisse M. (1988): From Text to Hypertext, *Byte* 13(10).

Gable A. and Page C.V. (1980): The use of Artificial Intelligence techniques in Computer-Assisted Instruction: An overview, *International Journal of Man-Machine Studies* 12(3).

Gaines B.R. and Linster M. (1990): Integrating a Knowledge Acquisition Tool, an Expert System Shell, and a Hypermedia System, *International Journal of Expert Systems* 3(2).

Garg P.K. and Scacchi W. (1989): ISHYS: Designing an Intelligent Software Hypertext System, *IEEE Expert.*

Goldstein I.P. (1982): The genetic graph: A representation for the evolution of procedural knowledge, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), USA: Academic Press.

Goldstein I.P. (1982): WUMPUS, in *Handbook of Artificial Intelligence* II, Barr A. and Feigenbaum E.A. (eds.), USA: Addison-Wesley.

Goldstein I.P. and Carr B. (1977): The computer as coach: an athletic paradigm for intellectual education, in *Proceedings of the National ACM Conference*, Seattle, Washington, USA.

Halasz F.G. (1988): Reflections on notecards: Seven issues for the next generation of hypermedia systems, *Communications of the ACM* 31(7).

Halff H.M. (1988): Curriculum and Instruction in Automated Tutors, in *Foundations of Intelligent Tutoring Systems*, Polson M.C. and Richardson J.J. (eds.), USA: Lawrence Erlbaum Associates.

Han B.J., Kahn P., Riley V.A., Coombs J.H. and Meyrowilz N.K. (1992): IRIS Hypermedia Services, *Association of Computing Machinery* 35(1).

Harmon P. (1987): Intelligent Job Aids: How Artificial Intelligence will change Training in the next five years, in *Artificial Intelligence and Instruction: Applications and Methods*, Kearsley G.P. (ed.), USA: Addison-Wesley.

Hartley J.R. (1973): The Design and Evaluation of an adaptive teaching system, *International Journal of Man-Machine Studies* 5(4).

Hartley J.R. and Sleeman D.H. (1973): Towards more Intelligent Teaching Systems, *International Journal of Man-Machine Studies* 5(2).

Hirschheim R. and Smithson S. (1988): A Critical Analysis of Information Systems Evaluation, in *Information Systems Assessment: Issues and Challenges*, Bjorn-Andersen N. and Davis G.R. (eds.), North Holland.

Johnson W.L. and Soloway E. (1985): PROUST, *Byte* 10(4).

Johnson W.L. and Soloway E. (1987): PROUST: An Automatic Debugger for Pascal Programs, in *Artificial Intelligence and Instruction: Application and Methods*, Kearsley G.P. (ed.), USA: Addison-Wesley.

Jonassen D.H. (1990): Semantic network elicitation: Tools for structuring hypertext, in *Hypertext: State of the Art*, McAleese R.M. and Green C. (eds.), UK: intellect ltd.

Kearsley G.P. (1987): *Artificial Intelligence and Instruction: Applications and Methods*, USA: Addison-Wesley.

Kimball R. (1982): A self-improving tutor for symbolic integration, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), USA: Academic Press.

King M. and McAulay L. (1992): Simple Expert Systems for Computer Assisted Instruction, in *Artificial Intelligence in Operational Research*, G.I. Doukidis and R.J. Paul (eds.), UK: MacMillan.

Koffman E.B. and Perry J.M. (1976): A model for generative CAI and concept selection, *International Journal of Man-Machine Studies* 8(4).

Kopec D., Brody M., Cheng Shi C. and Wood C. (1992): Towards an Intelligent Tutoring System with Application to Sexually Transmitted Diseases, in *Artificial Intelligence and Intelligent Tutoring Systems: Knowledge-based systems for teaching and learning*, Kopec D. and Thompson R.B. (eds.), Ellis Horwood Series in Artificial Intelligence, UK: Ellis Horwood.

Lawler R.W. and Yazdani M. (1987): *Artificial Intelligence and Education I: Learning Environments and Tutoring Systems*, USA: Ablex Publishing.

Leinhardt G. and Greeno J.G. (1991): The Cognitive Skill of Teaching, in *Teaching Knowledge and Intelligent Tutoring*, Goodyear P. (ed.), USA: Ablex Publishing Corporation.

Lesgold A. (1988): Toward a Theory of Curriculum for Use in Designing Intelligent

Instructional Systems, in *Learning Issues for Intelligent Tutoring Systems*, Mandl H. and Lesgold A. (eds.), GERMANY: Springer-Verlag.

Littman D. and Soloway E. (1988): Evaluating ITSs: The Cognitive Science Perspective, in *Foundations of Intelligent Tutoring Systems*, Polson M.C. and Richardson J.J. (eds.), USA: Lawrence Erlbaum Associates.

Mandl H. and Lesgold A. (1988): *Learning Issues for Intelligent Tutoring Systems*, GERMANY: Springer-Verlag.

Marchionini G. and Shneiderman B. (1988): Finding Facts vs. Browsing Knowledge in Hypertext Systems, *IEEE Computer*.

Miller J.R. (1988): The Role of Human-Computer Interaction in Intelligent Tutoring Systems, in *Foundations of Intelligent Tutoring Systems*, Polson M.C. and Richardson J.J. (eds.), USA: Lawrence Erlbaum Associates.

Minsky M. (1986): *The Society of Mind*, USA: Simon and Schuster.

Murray W.R. (1988): *Automatic Program Debugging for Intelligent Tutoring Systems*, UK: Pitman.

Nielsen J. (1990a): *Hypertext and Hypermedia*, USA: Academic Press.

Nielsen J. (1990b): The Art of Navigating through Hypertext, *Communications of the Association of Computing Machinery* 33(3).

Nwana H.S. (1990a): Intelligent Tutoring Systems: An Overview, *Artificial Intelligence Review* 4(4).

Nwana H.S. (1990b): Evaluation of an Intelligent Tutoring Systems, *Intelligent Tutoring Media* 1(3).

Nwana H.S. (1991): Mathematical Intelligent Learning Environments, *Intelligent Tutoring Media* 2(3/4).

Nwana H. and Coxhead P. (1988): Towards an intelligent tutor for a complex mathematical domain, *Expert Systems* 5(4).

O'Malley C. (1990): Interface Issues for Guided Discovery Learning Environments, in *Guided Discovery Tutoring: A Framework for ICAI Research*, Elsom-Cook M. (ed.), UK: Paul Chapman Publishing.

O'Neil H.F., Slawson D.A. and Baker E.L. (1991): Design of a Domain-Independent Problem-Solving Strategy for Intelligent Computer Assisted Instruction, in *Intelligent Tutoring Systems: Evolutions in Design*, Burns H.L., Parlett J.W. and Redfield C.L. (eds.), USA: Lawrence Erlbaum Associates.

O'Shea T. (1982): A self-improving quadratic tutor, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), Academic Press.

O'Shea T. (1989): Magnets, Martians and Microworlds: Learning with and Learning by OOPS, in *Artificial Intelligence and Education: Synthesis and Reflection*, Bierman D., Breuker J., and Sandberg J. (eds.), HOLLAND: IOS.

O'Shea T., Bornat R., du Boulay B., Eisenstad M. and Page I. (1984): Tools for creating intelligent computer tutors, in *Human and Artificial Intelligence*, Elithor and Banerjii (eds.), North Holland.

Palmer B.G. and Oldehoeft A.E. (1975): The Design of an Instructional System based on Problem Generators, *International Journal of Man-Machine Studies* 7(2).

Payne S.J. (1988): Methods and mental models in theories of cognitive skill, in *Artificial Intelligence and Human learning: Intelligent Computer Assisted Instruction*, Self J. (ed.), UK: Chapman and Hall Computing.

Powell P. (1992): Information Technology Evaluation: Is it different? *Journal of the Operational Research Society* 43(1).

Pirolli P. (1991): Computer-Aided Instructional Design Systems, in *Intelligent Tutoring Systems: Evolutions in Design*, Burns H.L., Parlett J.W. and Redfield C.L. (eds.), USA: Lawrence Erlbaum Associates.

Rada R. (1991): *Hypertext: From Text to Expertext*, USA: McGraw Hill.

Redfield C.L. and Steuck K. (1991): The Future of Intelligent Tutoring Systems, in *Intelligent Tutoring Systems: Evolutions in Design*, USA: Lawrence Erlbaum Associates.

Russell D.M., Moran T.P. and Jordan D.S. (1988): The Instructional Design Environment, in *Intelligent Tutoring Systems: Lessons Learned*, Psotka J., Massey L.D. and Mutter S.A. (eds.), USA: Lawrence Erlbaum Associates.

Schlumberger P.C. (1989): Fusing Hypertext with Artificial Intelligence, *IEEE Expert.*

Sculley J. (1989): The Relationship Between Business and Higher Education: a Perspective on the 21st Century, *Communications of the Association of Computing Machinery* 32(9).

Self J.A. (1985): Intelligent Computer Assisted Instruction, paper presented at the *ICAI Spring Seminar*, Logica Cambridge Ltd, UK.

Shneiderman B. and Kearsley G. (1989): *HYPERTEXT-HANDS-ON!: An Introduction to a New Way of Organising and Accessing Information*, USA: Addison-Wesley.

Shortliffe E.H. (1976): *Computer-based medical consultations: MYCIN*, HOLLAND: Elsevier Science Publishers.

Sleeman D.H. (1982): Assessing aspects of competence in basic algebra, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), USA: Academic Press.

Sleeman D.H. (1987): micro-SEARCH: A "shell" for Building Systems to Help Students

Solve Nondeterministic Tasks, in *Artificial Intelligence and Instruction: Application and Methods*, Kearsley G.P. (ed.), USA: Addison-Wesley.

Sleeman D.H. and Brown J.S. (1982): *Intelligent Tutoring Systems*, Computers and People Series, Gaines B. (ed.), USA: Academic Press.

Sleeman D.H. and Ward R.D. (1988): Intelligent Tutoring Systems in Training and Education: Prospects and Problems, in *Research and Development in Expert Systems V*, Kelly B. and Rector A. (eds.), British Computer Society workshop series.

Smeaton A.F. (1991): Retrieving information from hypertext: Issues and problems, *European Journal of Information Systems* 1(4).

Smith J.B. and Weiss S.F. (1988): Hypertext, *Communications of the Association of Computing Machinery* 31(7).

Smithson S. (1989): Guidelines for the user-centred evaluation of information retrieval systems, *Information Systems Department Working Paper Series* 11, London School of Economics, London, UK.

Stansfield J.C., Carr B. and Goldstein I.P (1976): Wumpus advisor I: a first implementation of a program that tutors logical and probabilistic reasoning skills, *AI Lab Memo* 381, MIT.

Stevens A.L. and Collins A.M. (1977): The goal structure of a Socratic Tutor, in *Proceedings of the National ACM Conference*, Seattle, Washington, USA.

Stevens A.L. and Collins A.M. (1979): Misconceptions in students' understanding, *International Journal of Man-Machine Studies* 11(1).

Stevens A.L., Collins A.M. and Goldin S.E. (1982): Misconceptions in students' understanding, in *Intelligent Tutoring Systems*, Sleeman D.H. and Brown J.S. (eds.), USA: Academic Press.

Streitz N.A. [1988]: Mental Models and Metaphors: Implications for the Design of Adaptive User-System Interfaces, in *Learning Issues for Intelligent Tutoring Systems*, Mandl H. and Lesgold A. (eds.), GERMANY: Springer-Verlag.

Suchman Lucy A. (1987): *Plans and situated actions: The problem of human\machine communication*, UK: Cambridge University Press.

Swartz M.L. and Yazdani M. (1992) *Intelligent Tutoring Systems for Foreign Language Learning: The Bridge to International Communication*, GERMANY: Springer-Verlag.

Symons V. (1991): A Review of Information Systems Evaluation: Content, Context and Process, *European Journal of Information Systems* 1(3).

VanLehn K. (1983): Human procedural skill acquisition: theory, model, and psychological validation, in *Proceedings of the National Conference on Artificial Intelligence*, Washington D.C., USA.

VanLehn K. (1988): Student Modelling, in *Foundations of Intelligent Tutoring Systems*, Polson M.C. and Richardson J.J. (eds.), USA: Lawrence Erlbaum Associates.

Wenger E. (1988): *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, USA: Morgan Kaufmann.

White B.Y. and Frederiksen J.R. (1985): QUEST: qualitative understanding of electrical system troubleshooting, *ACM SIGART Newsletter* 93(1).

White B.Y. and Frederiksen J.R. (1987): Qualitative models and intelligent learning environments, in *Artificial Intelligence and Education Volume one: Learning Environments and Tutoring Systems*, Lawler R.W. and Yazdani M. (eds.), USA: Ablex Publishing Corporation.

White B.Y. and Frederiksen J.R. (1990): Causal Model Progressions as a Foundation for Intelligent Learning Environments, in *Artificial Intelligence and Learning Environments*, Clancey W.J. and Soloway E. (eds.), USA: MIT Press.

Whitley E.A. (1990): *Embedding expert systems in semi-formal domains: Examining the boundaries of the knowledge base*, unpublished Ph.D. thesis, University of London, 1990.

Wilkins D.C., Clancey W.J. and Buchanan B.G. (1988): Using and Evaluating Differential Modelling in Intelligent Tutoring and Apprentice Learning Systems, in *Intelligent Tutoring Systems: Lessons Learned*, Psotka J., Massey L.D. and Mutter S.A. (eds.), USA: Lawrence Erlbaum Associates.

Williams M.D., Hollan J.D. and Stevens A.L. (1981): An overview of STEAMER: an advanced computer-assisted instruction system for propulsion engineering, *Behaviour Research Methods and Instrumentation* 13(2).

Woodhead N. (1991): *Hypertext and Hypermedia: Theory and Applications*, USA: Addison-Wesley.

Woolf B.P. (1987): Theoretical Frontiers in Building a Machine Tutor, in *Artificial Intelligence and Instruction: Applications and Methods*, Kearsley G.P. (ed.), USA: Addison-Wesley.

Woolf B.P. (1988): Representing complex knowledge in an intelligent machine tutor, in *Artificial Intelligence and Human learning: Intelligent Computer Assisted Instruction*, Self J. (ed.), UK: Chapman and Hall Computing.

Woolf B.P. (1991): Representing Acquiring and Reasoning about Tutoring Knowledge, in *Intelligent Tutoring Systems: Evolutions in Design*, Burns H.L., Parlett J.W. and Redfield C.L. (eds.), USA: Lawrence Erlbaum Associates.

Woolf B.P. and McDonald D.D. (1984): Context-dependent transitions in tutoring discourse, *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas, USA.

Yazdani M. (1986a): Intelligent Tutoring Systems Survey, *Artificial Intelligence Review*

1(1).

Yazdani M. (1986b): Intelligent Tutoring Systems: An Overview, *Expert Systems* 3(3).

Yazdani M. (1988): Expert Tutoring Systems, *Expert Systems* 5(4).

Yazdani M. (1989): Expert Tutoring Systems, in *Proceedings of the Fifth International Expert Systems Conference*, London, UK.

Yazdani M. Lawler R.W. and (1991): *Artificial Intelligence and Education II: Principles and Case Studies*, UK: Ablex Publishing.

Yob G. (1975): Hunt for Wumpus, *Creative Computing*.

APPENDIX A: GENERIC CATEGORIES OF INTELLIGENT TUTORING SYSTEMS

Intelligent Tutoring Systems have been implemented using many programming languages that run on many different sizes of computers. However, the bulk of Intelligent Tutoring Systems implementation has been done in exploratory programming environments like the various Lisp programming environments like interLISP, zetaLISP and muLISP [Doukidis, Shah and Angelides, 1988] and Expert System shells like Xi+ and Insight [Doukidis and Whitley, 1988] originally developed for Artificial Intelligence research and development.

These programming environments seek to minimise the time and effort required to go from an idea to its implementation and to minimise the difficulty of modifying the implementation as the idea changes. As a result the designer is encouraged to perform formative evaluations to actually get and use feedback by trying out early Intelligent Tutoring Systems to improve later ones. Burton [1988] argues that during the next period of Intelligent Tutoring Systems development, it will be critical to modify existing Intelligent Tutoring Systems quickly in order to respond to shortcomings discovered by their being placed in the field.

There have been many programming conventions used in developing Intelligent Tutoring Systems. Nevertheless, the Intelligent Tutoring Systems community classifies existing Intelligent Tutoring Systems in two broad categories: Intelligent Tutoring Systems developed with the Knowledge Based Systems Engineering Paradigm and those Intelligent Tutoring Systems developed with other programming conventions like procedural programming, simulations, object-oriented programming, etc. Following precedence, existing Intelligent Tutoring Systems will be presented in this fashion, starting first with the latter category.

The most popular way among Intelligent Tutoring Systems researchers for classifying existing pre-Knowledge Based Systems Intelligent Tutoring Systems in generic categories is by the approach which they follow in tutoring a certain topic: Tutorial Dialogues, Drills, Simulation and Instructional Games. In very broad terms, an Intelligent Tutoring System becomes an effective instructional tool if it cycles through the following four phases: presenting information, guiding the student, practising and assessing student learning. The four generic categories of Intelligent Tutoring Systems are discussed in relation to the cycle and then existing Intelligent Tutoring Systems classified under each generic category are presented.

A.1 Tutorial Dialogues

Tutorial Dialogues are the most basic form of an Intelligent Tutoring System. Such Intelligent Tutoring Systems tutor by carrying on a dialogue with the student. They would normally present information, ask the student questions or answer questions posed by the student and then make tutoring decisions whether to move on to new information or to engage in review, based on the student's comprehension. Most of the early Intelligent Tutoring Systems were developed along this theme of instruction. Such Intelligent Tutoring Systems do not normally engage in extended practice or assessment of learning [Alessi and Trollip, 1985].

Alessi and Trollip [1985] argue that Tutorial Dialogues are appropriate for presenting factual information, for learning rules and principles, or for learning problem-solving strategies. As such they satisfy only the first two phases of instruction: Presenting Information, and Guiding the student. Figure A.1 shows the general structure and sequence followed by a Tutorial Dialogue.

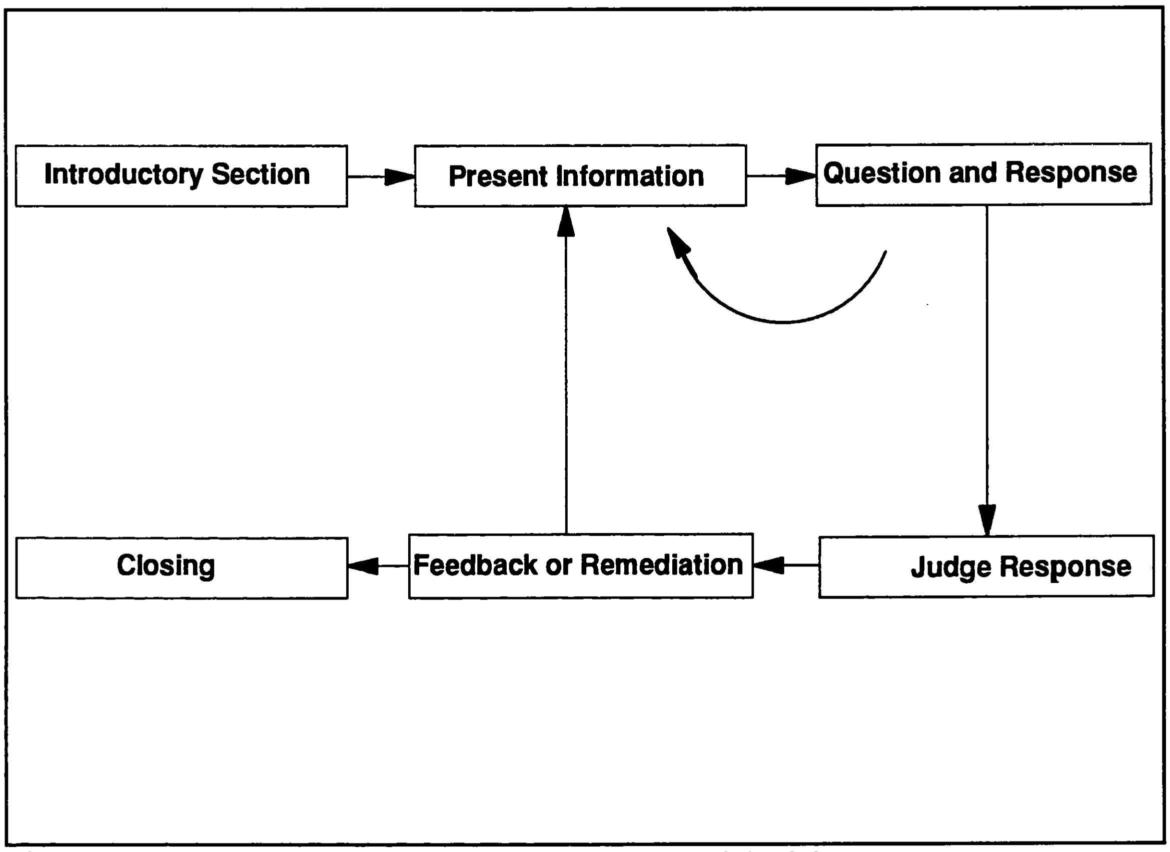


Figure A.1: The general structure and flow of a Tutorial Dialogue

An effective Tutorial Dialogue begins with an introductory section that informs the student of the purpose and nature of the tutorial. After that a form of a cycle begins: Information is presented and elaborated, questions are asked by or posed to the Intelligent Tutoring System, the Intelligent Tutoring System judges the response to assess student comprehension, the student is given feedback to improve comprehension and future performance, the Intelligent Tutoring System makes a sequencing decision to determine what information should be treated when the cycle begins again. The cycle continues until the lesson is terminated by either the student or the Intelligent Tutoring System. At this point there may be a summary and closing remarks.

SCHOLAR

SCHOLAR [Carbonell, 1970] is an Intelligent Tutoring System that can hold mixed-initiative tutorial dialogues with the students, responding to their questions by traversing a semantic network and asking them questions in order to convey the contents of the network to them interactively in a way a human teacher uses his own knowledge representation to generate tutorial sessions of explanations and questions.

SCHOLAR's knowledge of its subject matter, the geography of South America, is represented in a semantic network whose nodes stand for geographical objects and concepts organised in a partial hierarchy with their relations. Inferences can be made by propagation of inherited properties via these hierarchical links. For instance, SCHOLAR can conclude that Santiago is in South America since Santiago is in Chile which is in South America. In addition, the system can determine the semantic relation between two nodes by following their respective paths up the hierarchy until a common node is found. For instance, it can find that Santiago and Buenos Aires are both South American cities.

Typical of Tutorial Dialogues, SCHOLAR does not attempt to produce a model of the student, other than an oversimplified overlay model. SCHOLAR starts with the complete network to model the perfect student and progressively perturbs it to reflect the student's actual performance, deleting and even modifying nodes and links.

SCHOLAR's tutorial strategies are fairly primitive, consisting mainly of local topic selections. For instance, the student is allowed to ask vague questions such as "Tell me something about Peru". Since this type requires more than straight retrieval and inference, SCHOLAR must be able to determine which information is most relevant. The notion of distance between nodes in the network gives some indication of relevance and for this purpose numerical tags are attached to individual nodes and to their various attributes. These relevance tags provide some guidance to SCHOLAR in selecting topics, for instance in formulating questions, when the system is given the initiative.

The linguistic capabilities of the system's interface are restricted to a subset of English, which is limited to simple sentences. Text is generated by sentence and question templates that are filled in with information from the network. The parsing of student questions follows the same principle in reverse order, whereas the parsing of student answers is done by symbolic pattern matching against a list generated from the semantic network.

In general, with SCHOLAR there were two areas that need further improvement. First, was the extent of the inferences and diagnosis SCHOLAR was able to make within the representational framework of semantic nets, especially with incomplete knowledge. For instance, student diagnosis with SCHOLAR is exclusively by a means of a simplified overlay model with which SCHOLAR is not able to distinguish wrong answers. Second, is the lack of tutorial strategies necessary to conduct a coherent and effective dialogue. Figure A.2 shows an interaction protocol with SCHOLAR.

Carbonell [1973] proposed extensions for SCHOLAR. First, modifying the reasoning strategy to work with an open semantic network thus deal with incomplete knowledge. Second, contradictory concepts and object to be denoted, for instance, city and river, in order to facilitate negative inferences. Third, making assumptions, for example, if the language of a country is Spanish, then it cannot be French. Fourth, an inverse induction strategy, or lack-of-knowledge strategy, for instance, if Uruguay's product list does not

```
 WHAT'S CHILE?
*A COUNTRY*
 THAT'S FINE.
 SELECT AN ALTERNATIVE FROM THE LIST:
 214000.
 800300.
 49883.
 63690.
 FOR THE QUESTION:
   APPROX WHAT IS THE AREA OF CHILE?
*800300*
 YOUR ANSWER IS INCORRECT.
 THE CORRECT ANSWER IS: 214000.
 THE POPULATION OF CHILE IS APROXIMATELY 8500000. PEOPLE.
 TRUE OR FALSE?
*TRUE*
 YOU ARE CORRECT.
 USE ONE OF THE FOLLOWING:
 SUCRE
 FRENCH
 ARGENTINA
 SPANISH
 TO ANSWER THE QUESTION:
   WHAT IS THE LANGUAGE IN CHILE?
*
 YOU ARE TAKING TOO MUCH TIME ...
*SPANICH*
 YOU SOULD HAVE WRITTEN "SPANISH".
 VERY GOOD.
 THE CAPITAL IN CHILE IS SANTIAGO.
 CORRECT OR INCORRECT?
*CORRECT*
 THAT'S RIGHT.
.........
 IS IT CORRECT TO SAY THAT THE CITIES IN BRAZIL ARE
 SAO PAOLO, RIO DE JANEIRO, BRAZILIA, PORTO ALEGRE, BAHIA
 RECIFE, SANTOS, MANAOS, AND BELEM?
*YES*
 THAT'S FINE.
 THE CITIES IN BOLOVIA ARE —.
*LA PAZ, SUCRE, BOGOTA, AND ACONGAGUA*
 AT LAST! ANYWAY, LET US SEE WHAT YOU HAVE JUST WRITTEN.
 HAVEN'T YOU INCORRECTLY USED "BOGOTA" AND "ACONGAGUA"?
 ON THE OTHER HAND "LA PAZ" AND "SUCRE" IS CORRECT.
 THE COMPLETE ANSWER ALSO INCLUDES POTOSI, COCHABAMBA,
 AND SANTA CRUZ.
```

Figure A.2: An interaction protocol with SCHOLAR [Carbonell, 1970]

include oil, this does not mean that the country does not produce oil. Fifth, uncertainty tags would present levels of certainty through the inference process. Finally, a functional analysis that would be responsible for analysing the factors on which a hypothesis would depend like infer a region's climate by considering its location. While the first version relies heavily on randomness, this second version includes heuristic rules to direct this selection. An important extension is some capability to parse and discuss errors. In the new version of SCHOLAR, a separate module handles detected errors by reporting factors that distinguish the student's answer from the correct one. Figure A.3 shows an interaction protocol with the new version of SCHOLAR.

```
 SCHOLAR: The eastern and central plain of Argentina is --
  Student: Chaco.
SCHOLAR: No.
            The northern plain of Argentina is the Chaco. The
            eastern and central plain of Argentina is the Pampas.
  Student: Is the Chago in Chile?
SCHOLAR: No.
            The Chaco is in Argentina, Paraguay, and Bolivia, not
            in Chile.
  Student: Are the Andes a region in Chile?
 SCHOLAR: Well, you're almost right.
            More specifically, a region of Chile is the Southern
            Andes.
                 .
                 .
                 .
                 .
```

Figure A.3: An interaction protocol with the new version of SCHOLAR [Carbonell, 1973]

WHY

The new version of SCHOLAR led to the Socratic Method for teaching through Tutorial Dialogues. With the Socratic Method, the Intelligent Tutoring System does not teach a subject by direct exposition, but leads the student by successive questions to formulate general principles on the basis of individual cases, to examine the validity of his own hypotheses, to discover contradictions, and finally to extract correct inferences from the facts he knows. A Socratic Tutor achieves its goal through a succession of local decisions which aim at inducing the formulation of hypotheses and the testing of these hypotheses in a predictive fashion.

A typical Socratic rule has for condition the student response in relation to the tutor's knowledge, and as action what the tutor should ask or propose next. The Socratic Method leads the student to find errors or contradictions by entrapping him in the consequences of his own conclusions. If the student agrees with the overgeneral rule presented, the tutor will then come up with a known counterexample, thus forcing the student to correct the course of his reasoning by considering other factors. If the student does not agree with the over general rule, the same effect will be achieved by asking him why he disagrees.

WHY [Stevens and Collins, 1977] is an Intelligent Tutoring System for tutoring on rainfall processes using the Socratic Method. The domain knowledge about processes is stored in a hierarchy of Scripts which represent stereotypical instances of events because semantic networks cannot hold procedural knowledge. The generation of questions and the evaluations of student answers are based on these scripts. The nested script hierarchy provides an organised view of the knowledge, which the Intelligent Tutoring System can exploit thus giving sequences of events ranging from general understanding to detailed description of the process. Figure A.4 shows an interaction protocol with WHY. Again typical of Tutorial Dialogues WHY does not assess student learning by building a student model other than evaluations of responses.

WHY: Do you think the AMAZON jungle has heavy rainfall or light rainfall?
(Ask for a prediction about a particular case)
STUDENT: Heavy rainfall.

WHY: Why does the Amazon have heavy rainfall?
(Ask for prior cause)
STUDENT: Because it's near the Atlantic.

WHY: Yes, the AMAZON jungle is near the equatorial Atlantic.
How does that affect the rainfall in the Amazon jungle?
(Ask for intermediate cause)
STUDENT: The water in the current is warm.

...

WHY: Do you think that any place with mountains has heavy rainfall?
(Form a general rule for insufficient cause)
STUDENT: Yes.

WHY: Southern California has mountains.
Why doesn't Southern California have heavy rainfall?
(Pick a counter-example for an insufficient cause)

Figure A.4: An interaction protocol with WHY [Stevens and Collins, 1977]

In general with WHY, there were two areas that needed improvement. First, the global tutorial goals were ignored in the local applicability conditions of the tutorial rules. Second, the knowledge presentation was not enough to explain the mechanisms involved in the reasoning process or to diagnose and correct the student's misconceptions.

With respect to the first point, Stevens and Collins [1979] argue that diagnosing and correction would be the key solution. Diagnosis would trace surface errors to deep misconceptions and Correction would not only help remedy these but also classify different categories of errors. With respect to the second point, they propose incorporating functional analysis into WHY that would consider the various elements, which they call actors, involved in the processes, and their functions, which they call roles, in the interactions that give rise to various events. Also they argue that these actors have factors influencing the extent to which the roles of actors are manifested. Finally, functional relations hold between factors, and explain the results of processes. They argue that scripts control mainly the sequencing of major teaching episodes, functionality guides local interactions.

A.2 Drills

With Drills a selection of questions or problems is presented repeatedly until the student answers or solves them all at some predetermined level of proficiency. Alessi and Trollip [1985] argue that Drills are appropriate for the third phase of instruction only: practising. As such, Drills are not intended to tutor but merely practice with the assumption that the student is already familiar with the information to some degree. Therefore, assuming that initial information presentation and guidance have already occurred the function of Drills is merely to provide a means for practice. Figure A.5 shows the general procedure of a Drill.

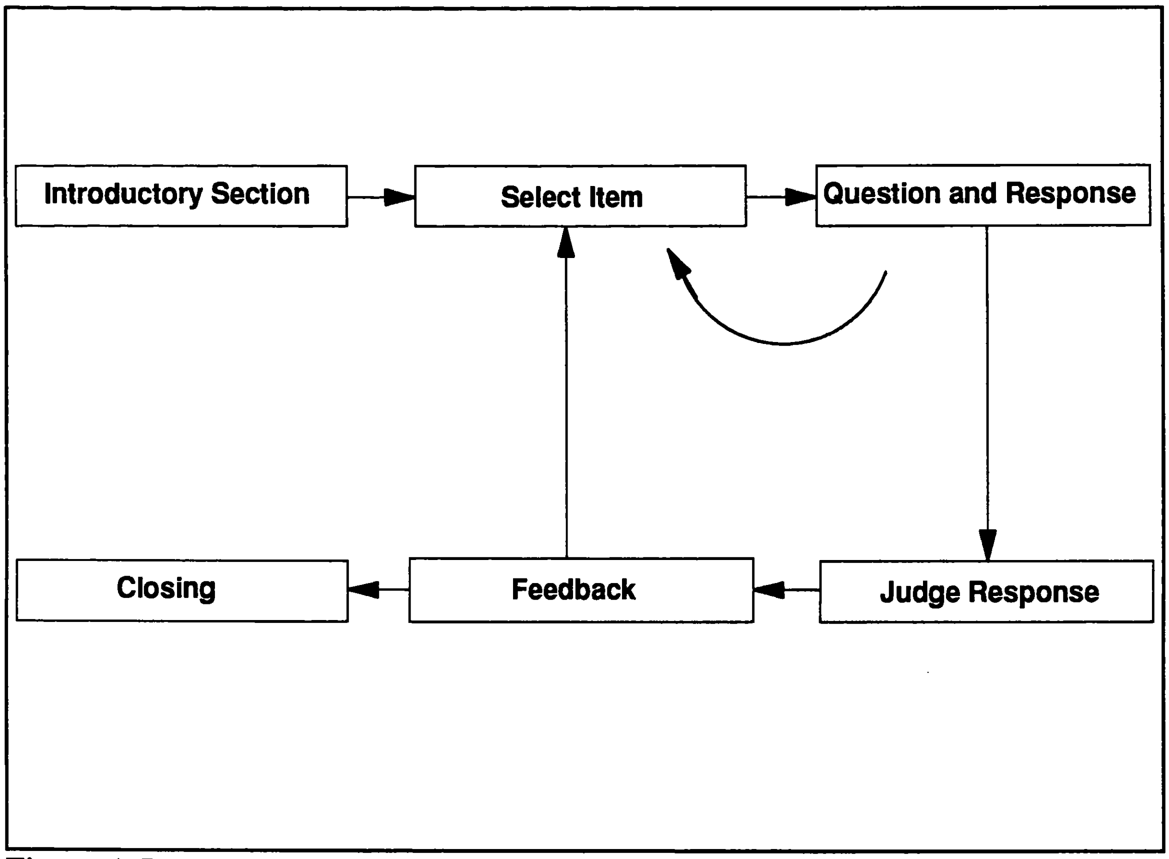


Figure A.5: The general structure and flow of a Drill

An effective Drill begins with an introductory section that informs the student of the purpose and nature of the Drill followed by a cycle that is repeated many times. Each time the cycle is repeated the following actions generally take place: An item is selected, the item is displayed, the student responds, the program judges the response, the student receives feedback about the response. Item selection may be in a specific order or random to increase variety. Termination may finish based upon a number of factors: number of items, time, level of quality of student performance.

The INTEGRATION Tutor

The INTEGRATION Tutor [Kimball, 1982] is an Intelligent Tutoring System for tutoring symbolic integration. Its main feature was in its use of an interaction between domain knowledge and a student model to guide the construction of a teaching sequence. The objective of the tutor was to communicate its judgemental knowledge while the student is solving a sequence of example problems selected by the INTEGRATION Tutor from a fixed archive.

Domain expertise is represented as a matrix that relates all problem classes to all solution methods. Each matrix element is a value indicating the probability of applying a given problem-solving approach to a given problem class. The underlying assumption is that the student uses the same classification for problems as the expert, an that the student knows the different approaches, for instance, integration by parts. The matrix representing expertise denotes an explicit characterisation of the teaching goal. The student's state of knowledge is represented as a similar matrix, whose values can readily be compared to those of the expert.

New problems are selected via differential modelling between the student model and the domain model in a breadth-first fashion rather than a focused fashion. The tutor can only give advice according to its own approach probabilities. Figure A.6 shows an interaction protocol with the INTEGRATION Tutor.

Tutor: Ok, Ruth, our integral is $\int X\log(X)dX$. What shall we do to solve it?
Student: sub
Tutor: Can you think of a substitution? (Yes, Help or Exit)
Student: yes
Tutor: Let U =
Student: log(X)
Tutor: Ok, Ruth, we will let U = log(X). Thus $X = e^{U}$ and $dX = e^{U}dU$ which we substituted into the intergrand. Our new integral is now $\int Ue^{2U}dU$. Please type Good, Bad, or Giveup:
Student: bad
Tutor: Can you think of a substitution? (Yes, Help or Exit)
Student: help
Tutor: i can't find a suitable substitution. Will exit ...
Ok, Ruth, our integral is $\int X\log(X)dX$. What shall we do to solve it?
Student: help
Tutor: i suggest you try integration by parts, What shall we do to solve it?
Student: parts
Tutor: Integration by parts! We shall use the notation $\int UdV = UV - \int VdU$. Can you think of a choice for U? (Yes, help or Exit)
Student: help
Tutor: How about letting U = log(X)

Figure A.6: An interaction protocol with the INTEGRATION Tutor [Kimball, 1982]

The simple language interface basically consists of multiple choice questions. For diagnosis, the system simply updates the probabilities in the student's matrix with methods from a probability theory with some ad hoc heuristics for generating a seed probability. A unique feature of the INTEGRATION Tutor is that its domain expertise can improve in the course of a tutorial session. If the student's approach leads to a better solution than the trace stored in the expert's archive, the system adopts the student's solution as its standard thus the label self-improving is attached to this tutor. Kimball's matrices may model the external behaviour manifested by knowledge in the domain, nevertheless, the probabilistic values are totally ignorant of the large sets of reasons and beliefs they summarise. In some ways, these reasons and beliefs are more important for real expertise than the behaviour they generate.

The QUADRATIC Tutor

The QUADRATIC Tutor [O'Shea, 1982] is an Intelligent Tutoring System on the domain of solving simple quadratic equations of the form $x^2 + c = bx$ whose answer can be obtained by clever guesses and with the help of a few rules. These rules are simple applications of the general root theorem, which states that b is the sum of the equation's two solutions and that c is their product. The tutor presents example problems on the basis of which students are to discover and master the rules. The QUADRATIC Tutor was an attempt at giving an Intelligent Tutoring System some ability to set up experiments using variations of its strategies and to adopt those that seem to produce the best results.

The QUADRATIC tutor has two tasks: it must select appropriate examples from an archive of problems, then tutor the student by providing him with comments and hints, and possibly by interrupting him if he takes too long. For these tasks, the system considers three sources of information: its task difficulty matric, its student model, and its tutorial strategies. The task difficulty matrix is used in the selection of new problems. It relates specific features in a problem to well-defined teaching goals. The student model is a set of hypotheses about the student's current mastery of each of the rules he must learn plus some combinations of them in an overlay model fashion. The teaching strategies are the core of the tutor, and they are represented as a set of production rules. Self improvement with the QUADRATIC Tutor deal is focused exclusively on the teaching strategies by experimenting with modifications of the teaching strategies.

With the representation of tutorial strategies in the form of rules rather than general didactic principles learning is empirical because it is impossible to reason about rules without knowing the principles which they embody. Thus its inability to detect interaction between experiments that have been setup to test different teaching strategies.

A.3 Simulations

Simulations imitate or replicate a phenomenon in order to tutor the student about it. Simulations do not tutor in the way a Tutorial Dialogue does by both presenting information and requiring student activity through appropriate question-answering techniques, but simply show the student something who is expected to learn by actually performing the activities to be learned in a context that is similar to the real world. In this simplified world, the student solves problems, learns procedures, comes to understand the characteristics of phenomena and how to control them, or learn what actions to take in different situations. The purpose is to help the student build a useful mental model of part of the world, and to provide an opportunity to test it safely and efficiently.

Simulations, in contrast to Tutorial Dialogues and Drills may be used with all four phases of instruction. In other words, simulations may serve for initial presentation, for guiding the learner, for practice, for assessing learning, or for any combination of these. Figure A.7 shows the flow of a simulation.

An effective simulation follows the following cycle: a scenario is presented, the student is required to react, the student reacts, the system changes in response to this action. The frequency of the simulation is dependent on the nature of the simulation. Simulations are further subdivided into four main categories: Physical, Procedural, Situational and Process.

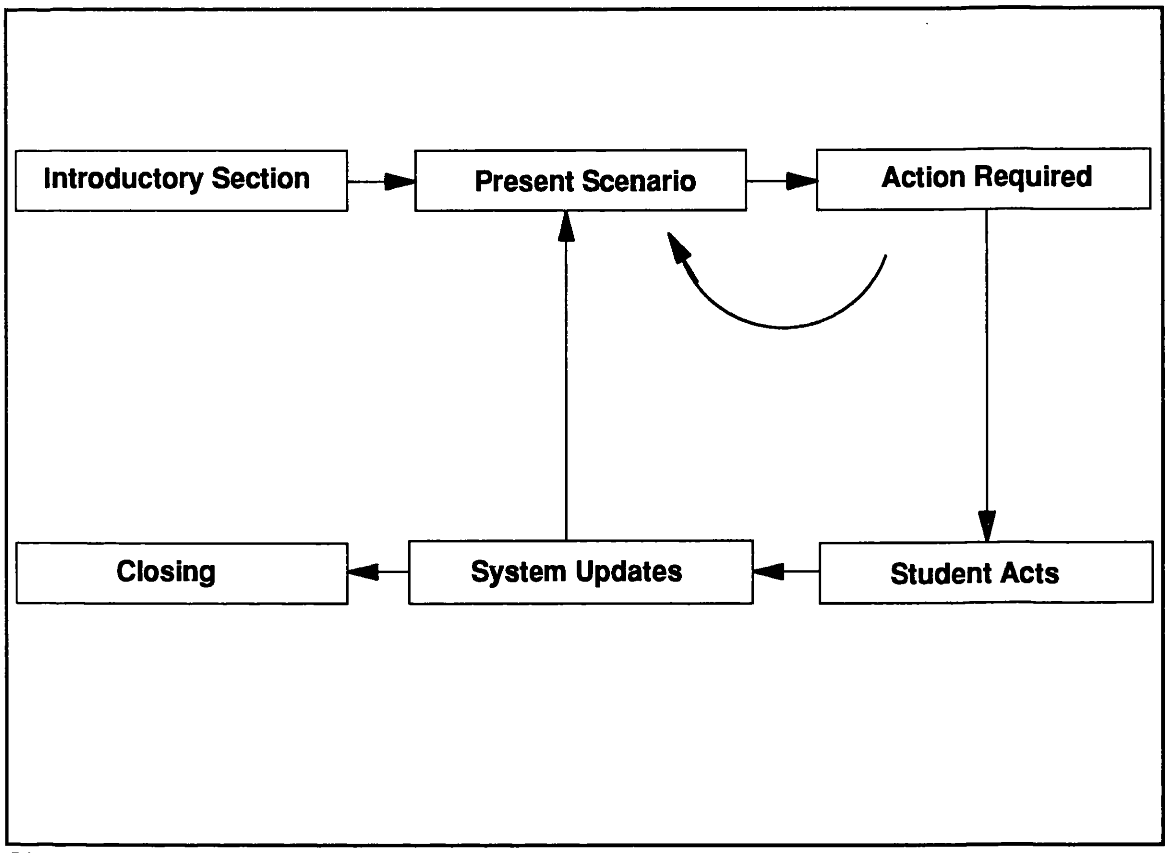


Figure A.7: The general structure and flow of Simulation

In an Intelligent Tutoring System encompassing a physical simulation, a physical object is displayed on the screen, giving the student an opportunity to use it or learn about it. Typical examples are a machine that the student must learn to operate, or some laboratory equipment to be used in an experiment. For example, in a Flight Simulator Training Program, the trainee may see simplified versions of the plane's instruments and, perhaps, the view through the cockpit window. The purpose of the simulation may be for the student to learn the relationship between instrument readings and the plane's passage through the air. The student may operate simulated controls to see how the instrument readings or the view through the window change in response to control inputs. Physical simulations are often used to depict experiments. For instance, on the screen the student may see laboratory equipment, such as the apparatus required to perform steam plant operations. In laboratory simulations like these, the student observes and uses equipment, water, instruments and various energy sources.

Alessi and Trollip [1985] argue that in most simulation-based Intelligent Tutoring Systems physical simulations play a secondary role to procedural simulations. That is, the physical simulations exists only as a vehicle for the procedural content. Thus, the student learns about how the simulated machine works, not as an end in itself, but rather as a means for acquiring the skills and actions needed to operate it. For example, the Flight Simulator Training Program that simulates the important flight instruments of an airplane, for example, is more likely to be used primarily to teach procedures of flying rather than how the instruments work.

The purpose of most procedural simulations is to teach a sequence of actions that constitute a procedure, for example, diagnosing an equipment malfunction like an electronic circuit. Many physical simulations are also procedural simulations, for not only is the physical entity imitated, but also the student's performance must imitate the actual procedures of operating or manipulating it. In fact, the primary focus of a simulation is usually procedural, and the simulation of the various physical objects is therefore necessary to meet the procedural requirements. The important feature with procedural simulations is diagnosis: the student is presented with a problem to solve, and must follow a set of procedures to determine the solution. For example, in diagnosing electronic faults the student applies the correct sequence of tests to determine the nature and composition of the circuitry, locate the fault, and then repair it.

With procedural simulations, whenever the student acts, the computer reacts, providing information or feedback about the effects the action would have in the real world. Based on this new information, the student takes successive actions and each time obtains more information. The primary characteristic of procedural simulations is that there is usually a correct or preferred sequence of steps that the student should learn to perform.

Situational simulations deal with the attitudes and behaviours of people in different situations, rather than with skilled performance. In other words, it is some form of a role playing simulation. Unlike procedural simulations, which teach sets of rules, situational simulations usually allow the student to explore the effects of different approaches to a situation, or to play different roles in it. In virtually all situational simulations, the student is an integral part of the simulation, taking one of the major roles. The other roles may be performed by students who interact with the same Intelligent Tutoring System, or by the Intelligent Tutoring System playing the role of a person. By going through the simulation a number of times, the student learns a set of behaviours that result to different approaches to various situations.

With process simulations, the user selects values of various parameters at the beginning of the simulation, and then watches the process occur without intervention. The student neither participates in the simulation as with situational simulations, nor constantly manipulates it as in physical or procedural simulations. Learning from such simulations occurs by repeating the process a number of times with different starting values and comparing the results. Alessi and Trollip [1985] argue that process simulations are accelerated or slowed-down versions of the real process. That is they alter the rate at which the process occurs in real life to one that enhances the learning process.

The METEOROLOGY Tutor

The METEOROLOGY Tutor [Brown, Burton and Bell, 1973] is a mixed-initiative Intelligent Tutoring System for tutoring on meteorology on the basis of a simulation-based representational scheme. The causal knowledge about processes is represented in finite-state automata in which sequences of events is a simulation of transitions between states. Individual automata representing different meteorological processes are captured in a process model and transition conditions are augmented with global predicates that refer explicitly to assumptions constituting the context of questions. Hence, the automata look like augmented transition networks. Factual information is contained in a semantic network, which is used to answer questions in a SCHOLAR style. Figure A.8 shows an interaction protocol with the METEOROLOGY Tutor.

Factual Question:
Question: is it true that fog is a form of precipitation?
Answer: No, fog is an example of condensation.
Questions about processes:
Question: (What happens if the air temperature decreases to 20 degrees and there are no condensation nuclei)
Answer: The (humidity of saturation) (decrease) from 24 to 17 mm Hg because (air temperature) (decrease). The (relative humidity) (increase) form 100 to 180% because (humidity of saturation) (decrease) and gradient of change) (sudden). The (water temperature) (decrease from 25 to 20 degrees because (air temperature) (decrease). The (rate of evaporation) (decrease) from 24 to 17 mm Hg because (Water temperature) (decrease). Then the (air condition) (supersaturated) because (rate of evaporation) (decrease) and gradient of change) (sudden).
Question: Suppose the pressure decrease gradually)
Answer: The (absolute humidity) (decrease) because ...

Figure A.8: An interaction protocol with the METEOROLOGY Tutor [Brown et al, 1973]

For the generation of answers to questions about processes, small units of text are associated with each state in the various automata and sentences are constructed by simple concatenation of the text elements. The simulation is a qualitative one based on state transitions. Quantitative information is introduced after some computations. Figure A.9 shows an example of an automaton used by the tutor.

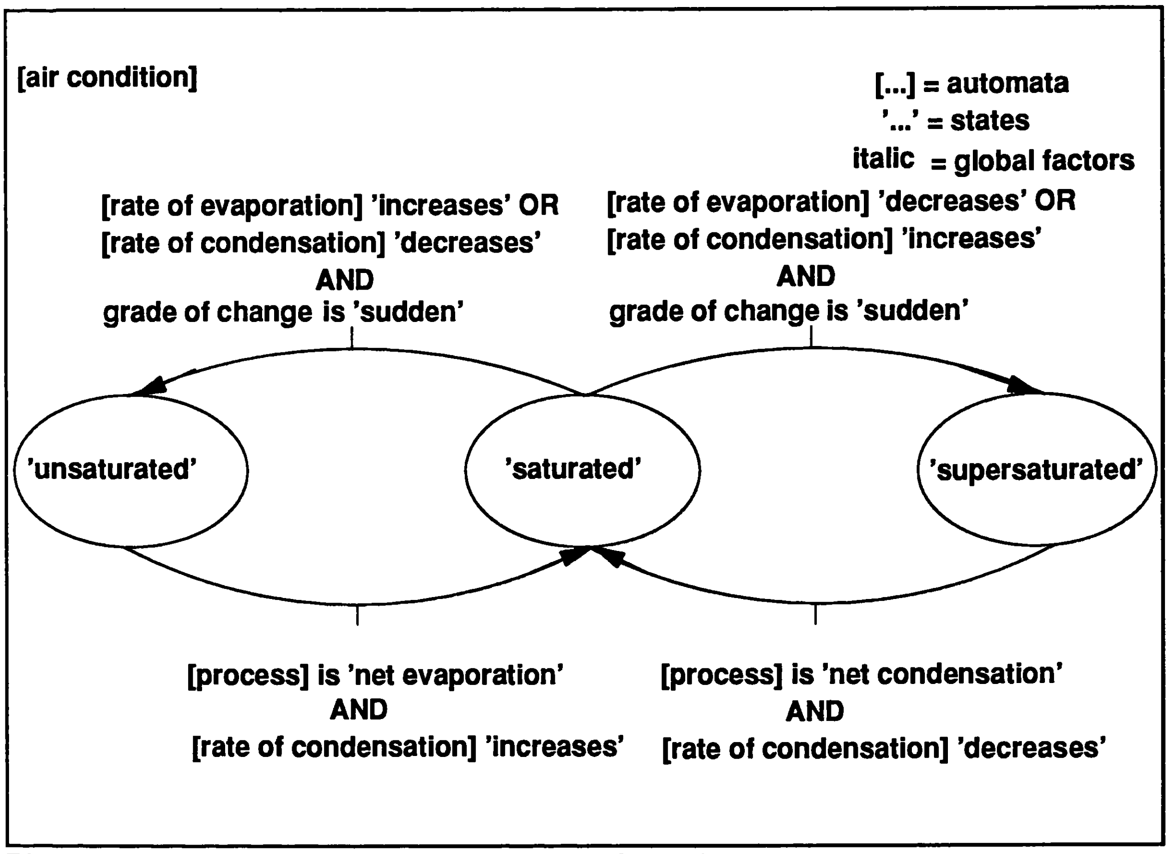

[air condition]
[...] = automata
'...' = states
italic = global factors
[rate of evaporation] 'increases' OR [rate of condensation] 'decreases' AND grade of change is 'sudden'
[rate of evaporation] 'decreases' OR [rate of condensation] 'increases' AND grade of change is 'sudden'
'unsaturated'
'saturated'
'supersaturated'
[process] is 'net evaporation' AND [rate of condensation] 'increases'
[process] is 'net condensation' AND [rate of condensation] 'decreases'

Figure A.9: An automaton for the condition of the air

These causal descriptions are based on an inference tree built dynamically by simulation of the question on the model. The state of the appropriate automaton is set to reflect the question. The enable transitions are propagated throughout the model, and recorded in the inference tree.

SOPHIE

SOPHIE (SOPHisticated Instructional Environment) [Brown, Burton and Bell, 1975] is an Intelligent Tutoring System for tutoring on troubleshooting of electronic circuits. Since the problem solving activity revolves around a model of a circuit whose components can be faulted, troubleshooting involves performing a series of measurements to propose and test hypotheses concerning the location and nature of the fault. The student has then the chance to apply his theoretical knowledge of electronic laws to understanding the causality underlying circuit mechanisms and also to acquire troubleshooting strategies. Therefore, the pedagogical purpose with SOPHIE is not to imitate a dialogue with a human teacher but to provide a reactive learning environment, a simulated laboratory, in which the student can apply his knowledge and receive informed feedback.

A major attraction the development of SOPHIE achieved was the joint application of Semantic Grammars and Semantic Augmented Transition Networks to the development of its natural language interface. Nevertheless, the introduction of semantic information into the parsing mechanisms meant, at that time, that the domain of tutoring be limited and well-defined. The internal meaning of all sentences consists of operations of operations to be performed, expressed as functions with arguments. The interface could handle incomplete, nevertheless, contextually meaningful questions. Figure A.10 shows an interaction protocol with SOPHIE.

```
Student: Now what is the outpt?
SOPHIE: (outpt => output) The output voltage Is 1.2 volts.
        If you are not sure whether a value Is correct, you can ask
        "Is that value correct?"
Student: Is that right?
SOPHIE: In a working circuit the output voltage is 30.1 volts. That value is incorrect.
Student: What is the voltage between the currect limiting transistor and the
         constant current source?
SOPHIE: The voltage between N4 and N5 is 1.28 volts.
Student: Is there any ripple in the voltage at the base of Q4?
SOPHIE: I'm sorry. I do not handle AC characteristics.
Student: What Is the base current of Q3?
SOPHIE: 12 microamps.
Student: Thru the emitter?
SOPHIE: The emitter current of Q3 Is -.26 milliamps.
Student: Q4
SOPHIE: -.26 milliamps
Student: If C2 were leaky, what would the voltage across it be?
```

Figure A.10: An interaction protocol with SOPHIE [Brown et al, 1975]

SOPHIE-I

SOPHIE-I [Brown and Burton, 1975] was an extension to SOPHIE. The difference with SOPHIE-I is in its use of multiple simulation-based representations of its domain knowledge including a mathematical model of the circuit both procedural and declarative. SOPHIE-I is not meant to function as a complete source of instruction, but rather as an automated lab around a simulated circuit, with an instructor providing intelligent critiques. It does not take any pedagogical initiatives since there are almost no pedagogical expertise. Furthermore, it does not make any attempts to build a student model, nor does it possess an explicit knowledge of active troubleshooting strategies.

Once a fault has been inserted into the circuit to debug it, the procedural model answers specific requests and queries by setting up appropriate experiments on a general circuit simulator based on the circuit's mathematical model. The procedural model has four tasks: answer hypothetical questions about the consequences of an action or event, evaluate hypotheses although the SOPHIE-I knows the correct answer, provide a list of all possible hypotheses consistent with the set of measurements performed so far, and evaluate a new measurement proposed by the student. Figure A.11 for an interaction protocol with SOPHIE-I.

For tutoring purposes, the main drawback of quantitative simulation is its inability to give a full account of the causality underlying its inferences. Causality is pedagogically important because it is the main ingredient of the kinds of explanations human students can understand. In a troubleshooting context, causality, more than information content, drives the diagnostic reasoning and the decision to perform measurements.

Student: What could be wrong?
SOPHIE: Here are some faults which explain the measurements you observed:
the beta of Q3 is low
the beta of Q4 is low
or R22 has the wrong resistance
Student: What would the output voltage be if the beta of Q3 were 8?
SOPHIE: If the beta of Q3 were 8 under the present setting, the output voltage would be 10.5 volts. In a working circuit it is 19.9 volts.
Student: Is the emitter current of Q4 correct?
SOPHIE: The emitter current of Q4 is -61 milliamps. In a working circuit the emitter current of Q4 is -49 milliamps. That value seems to be incorrect.
Student: Could R22 be low?
SOPHIE: Is it OK if I use 25 ohms for the resistance of R22?
Student: Yes
SOPHIE: That fault agrees with all of the measurements you have made.

Figure A.11: An interaction protocol with SOPHIE-I [Brown and Burton, 1975]

SOPHIE-II

SOPHIE-II was an attempt to improve the pedagogical setup in which SOPHIE-I's simulated lab was used. SOPHIE-I did not take any didactic initiatives. In SOPHIE-II a troubleshooting expert is added to the simulated lab who can demonstrate troubleshooting strategies, given a fault in the circuit in a function part of the circuitry which the student has selected. He can then watch the expert reason out the location of the fault. Figure A.12 shows an interaction protocol with SOPHIE-II.

The student is involved in the process by the questions the expert asks as it demonstrates its troubleshooting strategies. A referee makes explicit use of SOPHIE-I's simulated lab to rectify incorrect predictions made by the student. The troubleshooting expert reasons qualitatively making causally meaningful measurements and explaining its strategic decisions. SOPHIE-II pedagogical significance is that the student's personal

SOPHIE-II: How does the OUTPUT CURRENT of the CONSTANT CURRENT SOURCE compare with a working circuit?
Student: OK
SOPHIE-II: The current source Is working then. The modules now left are CL, VL, OPF, and DARL. I think i will make my next measurement on the Darlington amplifier. I want to know whether the BASE CURRENT of Q4 is OK.
Student: HELP
Referee: Let's use SOPHIE to find the answer.

>> is the base current of Q4 OK?
SOPHIE-I: The base current of Q4 is 0.0 amps, which seems to be incorrect because in a working circuit it is .031 milliamps.
Referee: So, you see, the BASE CURRENT of Q4 is LOW in the faulted circuit. Let's go on ...
SOPHIE-ii: Low input current to the Darlington amplifier wouid certainly result in lower than normaivoitage at its output under moderate current loads. I think then, the Darlington amplifier is OK, since it is following its input. Now, since we have already seen that the CCS output current is OK, perhaps either the current limiter or the voitage limiter must be drawing excessive current from the CCS. ...

Figure A.12: An interaction protocol with SOPHIE-II

experimentation alternates with observation of expert behaviour actively involving his understanding of the circuit.

SOPHIE-III

SOPHIE-III, a proposed extension to SOPHIE-I, was designed to provide a pedagogically active environment organised around a quantitative simulation, which centres on the learning needs of the student by supporting coaching and student modelling while allowing the student to take the initiative. SOPHIE-III was an effort to give SOPHIE-I's inferential abilities a humanlike reasoning flavour, the kind of causal reasoning performed by human troubleshooters. Of the SOPHIE-III project out came the BLOCKS Tutor, BUGGY, and WEST each dealing with a different problem, the designers of SOPHIE-III had to tackle. Coaching of problem-solving in WEST, student modelling in BUGGY, and troubleshooting strategies in the BLOCKS Tutor. It is worth mentioning that SOPHIE-III was never developed.

SOPHIE-III was meant to encompass a troubleshooting expert (a glass box system) accessible for tutorial purposes working on top of an electronics expert which propagates quantitative information about voltages and currents across components which are translated into qualitative information for a rule-based expert system that infers the behaviour of the circuit's modules which are in turn used to analyses the circuit in terms of the behaviour of its logical modules. The aim of the SOPHIE-III project arose from the need to understand an electronic circuit where each component has some known properties, and where the function of the whole circuit results from the structural relations between those components.

In trying to achieve this aim, SOPHIE-III's designers questioned the cognitive assumptions of the knowledge and the reasoning that pertained the representation and the representation itself. Furthermore, student modelling with the whole of the SOPHIE project did not go any further than the development of simple overlay models. This gave rise to their proposal of the ENVISION project. Nevertheless, the student modelling is still not addressed. From a pedagogical point, the emphasis was on the production of explanations, which was the major weakness of quantitative simulation but from a descriptive standpoint: student modelling was still at its best an overlay model with no concern at all of misconceptions diagnosis.

STEAMER

STEAMER [Williams, Hollan and Stevens, 1981] is an interactive, inspectable simulation based Intelligent Tutoring System for training engineers who will operate large ships, specifically the steam propulsion plants of these ships. The object of such training is to learn to perform the vast collection of procedures associated with both normal and abnormal operating conditions.

STEAMER's attraction is in its display of a running model of the propulsion plant using a language of animated icons, which allow the student to form a mental model and to learn procedures by manipulating this simulated plant. The various indicators appearing in the display are connected to an underlying quantitative model and are updated as the simulation proceeds. They amy also be set to specific values by the student, who can then observe the consequences of his manipulations. A hierarchical decomposition of the model allows the student to explore subsystems in further details. This is enabled by an object-oriented graphics editor that manipulates icons representing objects such as gauges, pipes and flows. There are facilities for connecting these objects to the variables of an underlying quantitative model and for associating procedures with objects.

STEAMER's pedagogical capabilities covering plant operating procedures, basic engineering principles, and explanations about plant functioning are the result of a tutor module that provides feedback during the execution of known procedures and a minilab for exploring the structure of specific components. However, it still shares many of SOPHIE's pedagogical limitations particularly with respect to causal explanations with quantitative simulations. Nevertheless, the graphic interface of STEAMER makes the mental model of a complex steam plant inspectable for instructional purposes which provides a realistic experience that, for example, the cost, availability, safety and training site [Angelides and Doukidis, 1990] may prohibit.

QUEST

QUEST (Qualitative Understanding of Electrical System Troubleshooting) [White and Frederiksen, 1985] is an Intelligent Tutoring System which shares with STEAMER the same practical goal of building a learning environment centred around a graphic simulation for instruction about physical devices in which the student can solve circuit problems. Graphic simulations and causal explanations of circuit behaviour play a prominent role because of the emphasis placed on supporting the student's development of executable models of electrical circuits. The goal is for the student to understand the general principles governing the behaviour of these circuits so ass to be able to predict the states of components and perform a small set of troubleshooting operations.

With QUEST explanations based on qualitative models find direct application in a learning environment centred on a simulated circuit. The simulation is basically component-oriented, but it does provide some of the advantages of process-oriented simulations by incorporating some higher-level concepts with which to guide the evaluations of component states.

A.4 Instructional Games

The purpose of both simulations and games is to provide an environment that facilitates learning or the acquisition of skills. Simulations do so by mimicking reality and games by providing the student with entertaining challenges. The purpose of Instructional Games is to tutor and as such they convey a variety of information like facts and principles, processes, the structure and dynamics of systems, skills such as problem solving, decision making or strategy formulating, social skills such as communication and attitudes and a variety of identical skills such as the nature of competition, how people cooperate, the dynamics of social systems, the role of chance, and the fact the penalties often have to be paid for just or unjust reasons.

Alessi and Trollip [1985] argue that games tend to motivate students and focus their attention on the goal of the game and enhance the learning environment because the teacher plays a less dominant role and is not the only judge of performance. Figure A.13 shows the flow of a game. A game has basically the same cyclic nature as a Simulation. The only difference is the addition of an optional input by an opponent.

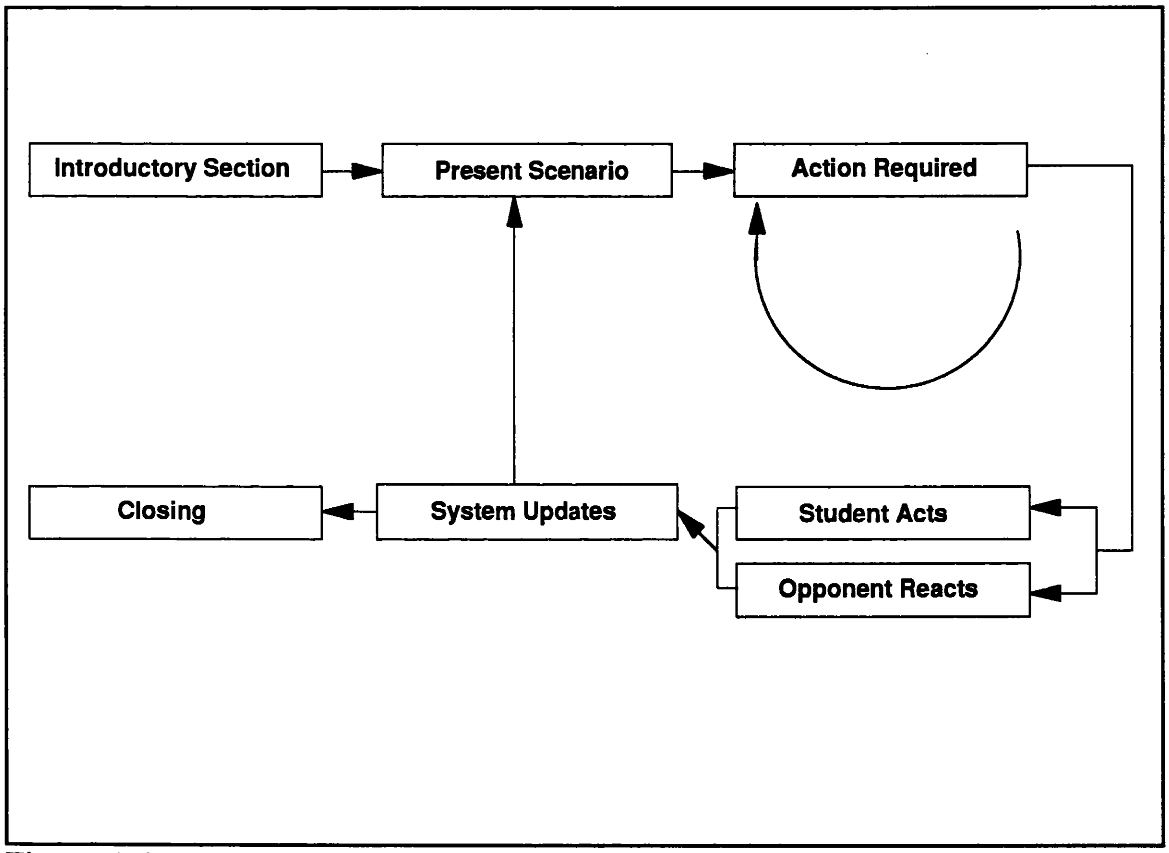


Figure A.13: The general structure and flow of an Instructional Game

WEST

WEST is an Intelligent Tutoring System for coaching with the computer game "How the WEST was won" developed for the PLATO project. The project came out of the research Brown and Burton were doing for SOPHIE-III in the mid 1970s thus keeping in place with the concept of reactive learning environments which was a central theme to the whole of the SOPHIE project. The purpose of the game is to exercise arithmetic skills. Players are involved in a race to their home town. When their turn comes, they are given three random numbers by spinners. With these numbers, they have to compose an arithmetic expression that involves two different operators and that determines the number of spaces they can move. However, getting the largest number is not always the best strategy, because of shortcuts and the possibility of bumping backward an opponent who is not in a town. As a result players are encouraged to explore different ways of combining the numbers with arithmetic operators.

With WEST Burton and Brown [1976] proposed the issues paradigm. With this paradigm, the knowledge to be conveyed is described as a set of issues, which are presented to the student as they become relevant to the game along with his moves. Finding relevant issue is achieved with differential modelling. However, because of the game's randomness, it is not possible to predict which skills will be necessary ahead of a move.

When it is the student's turn to play, the expert generates an ordered list of all the possible moves. If the student's expression does not yield the expert's optimal move, the diagnostic process starts. First, the student's move is analysed by the issue recognisers to determine which issues are used. Then, all the expert's moves that are better than the student's are also analysed by the same recognisers, to produce a list of issues the student failed to apply. Finally, these issues are evaluated in the existing differential student model to find one in which the student is known to be weak. If one such issue is found, it can be presented to the student: both abstractly, using a piece of prestored text and concretely using the expert's better move as an example. Figure A.14 shows an interaction protocol with WEST.

```
          Bob is at 54 – the computer is at 40.
   WEST: It's MY turn ... Spinners are 2 4 2 My expression is:
          2 + (4/2) which gives ME 4
          I took a SHORTCUT
          I bumped YOU
          the COMPUTER is at 54 – Bob Is at 40.
   WEST: It's your turn ... Spinners are: 1 2 2 What arithmetic
          expression did you form?
 Student: 1 + 2*2
   WEST: What number does that give you?
Student: 5
  WEST: You don't seem to be bumping very much. Bumps are hard to
          get bu they are usually a good idea. One good example
          would be the expression: (1*2) + 2, which would give you a
          SHORTCUT and a BUMP!! So you could have been at 54 while
          I would have ended up at 40. Bob Is at 45 -- the COMPUTER Is at 54.
```

Figure A.14: An interaction protocol with WEST [Burton and Brown, 1976]

In addition to helping select relevant topics, the differences between the lists of issues respectively applied by the student and the expert also provide information for updating the student model. Pedagogically WEST adopts a cautious attitude and intervenes only when there is good evidence for a weakness. WEST's expert needs a global strategy to determine optimality when ordering moves but does not need to use the issues, or even to know about them, since its moves are analysed by the same diagnostic procedures as are the student's.

WUSOR

WUSOR [Goldstein and Carr, 1977] is an Intelligent Tutoring System for coaching with the computer game WUMPUS [Yob, 1975]. WUMPUS takes the player through successive caves in a warren where the terrible Wumpus is hiding. In addition to

Wumpus, other dangers are lurking: deadly pits, and bats that grab the player and drop him in a random cave. Whenever the player reaches a new cave, he is given a list of the neighbouring caves. He also receives some warnings when applicable: a draft or a squeak reveals the presence of a pit or a bat respectively in an unspecified neighbouring cave. The Wumpus itself can be smelled up to two caves away. The player moves by selecting the neighbouring cave he wants to visit next. To win the game, he must shoot one of his five arrows into the WUMPUS lair. He looses if he falls into a pit, by walking into the WUMPUS lair or by using up his arrows without scoring any hits. In deciding which neighbouring cave to visit next, the player must exercise logical and probabilistic reasoning to draw inferences from the signals she has received so far within a time-limit.

WUSOR-I [Stanfield, Carr and Goldstein, 1976] consists of an expert and an advisor. In the expert, the domain knowledge is represented in the form of production rules (unlike WEST's black box system) which are organised into different danger specialists which cooperate to classify all possible moves and select the best one. The expert keeps a record of the information gather so far. The expertise consists of heuristic rules for approximate probabilities. The advisor intervenes each time the student has not chosen the optimal move according to the expert's classification with the appropriate explanation given. The advisor does not maintain a student model nor does it attempt to diagnose the student's state of knowledge and thus adapt its interventions accordingly. The players are asked to rate themselves, a rate which is seriously taken into account by the coach in preparing the material complexity.

WUSOR-I was updated to WUSOR-II by the incorporation of the overlay theory into its student modelling. This was a move towards a learner-oriented tutor, both for the student model and for the representation of expertise. The student knowledge is represented as an overlay model and his learning preferences are represented as a set of three registers indicating his need for repetition, his degree of forgetfulness and his receptivity to advice. Game expertise is organised into five successive phases ranging from basic rules of the game to logical and finally to probabilistic rules. The student will not be allowed to proceed to the next phase before he has mastered required skills in the current one. Figure A.15 shows an interaction protocol with WUSOR-II.

```
WUSOR: Hi Mary. There are 15 caves, 2 bats, 2 pits and 1 Wumpus.
       You are now at cave 15 with neighbours 4, 14, and 0. Brrr!
       There is a draft. Your are near a pit. What a stench! The
       Wumpus is near! What now?
Student: 4
WUSOR: You are now at cave 4 with neighbours 15, 14, and 2. Brrr!
       There is a draft. Squeak! A bat is near. What now?
Student: 14
WUSOR: Mary, it isn't necessary to take such large risks with pits.
       One of caves 2 and 14 contains a pit. Likewise, one of caves
       0 and 14 contains a pit. This is muitiple evidence of a pit in cave 14
       which makes it likely that cave 14 contains a pit. It is less likely that
       cave 0 contains a pit. Hence, MAry, we might want to expiore cave
       0 instead. Do you want to take back your move?
```

Figure A.15: An interaction protocol with WUSOR-II [Goldstein, 1982]

WUSOR-II was updated to WUSOR-III [Goldstein, 1982] by the incorporation of the genetic graph to combine the concept of overlay on a rule-based representation of domain knowledge with a learner-oriented set of links. The genetic graph represents elementary subskills as nodes connected by links representing their evolutionary relations, such as generalisation or analogy. With WUSOR-III the student's knowledge can be represented as an overlay on the nodes of the genetic graph, including correct and incorrect rules and his learning history as an overlay on the links of the genetic graph. The overlay on the links then supports pedagogical actions that view learning as a process of building upon existing knowledge. WUSOR-III was never fully implemented with much of the genetic graph remaining an idea.

BUGGY

BUGGY [Burton, 1982] is an Intelligent Tutoring System for tutoring arithmetic skills. The BUGGY project that came out of SOPHIE-III was centred around the design of a student diagnostic model: that is a model of the student's current skills that would reflect its exact composition of correct and incorrect elementary subskills. The skills in the diagnostic model were represented as a black box procedural hierarchical network which is a decomposition of that skill into subprocedures which are linked together in a lattice of subgoals. The procedural network is executable and thus it can be used to on a set of problems to model the skill that it represents. Its structure is also inspectable.

The most important feature of the procedural network is that it is possible to include in the hierarchical structure all the possible buggy variants of each subskill. It achieves this by performing a generate-and-test diagnosis. It replaces an individual subskill in the procedural network by one of its bugs, and thus attempt to reproduce a student's incorrect behaviour. If such a faulted network does obtain the same answers as the student on a sufficient set of problems, the bugs that have replaced the correct subskills in the network are then claimed to be possessed by the student. This deterministic deep-structure model explains the student's incorrect behaviour in terms of a set of exact internalised errors. The limitations of this approach to pedagogical instruction is the in-depth analysis of the domain and of actual performances by students and then the lack of explanation of the bugs. BUGGY was developed as a game. Figure A.16 shows an interaction protocol with BUGGY in the domain of addition.

```
BUGGY: Welcome to BUGGY. I have chosen a bug. Here is an example of the bug.
              17
             +5
             ──
              13
              Now you can give me problems to determine the bug.
   Team:   18     43
            +6   +79
            ──   ───
BUGGY:       15       23
 Team: Got the bug! Student adds the number to be carried in the last column worked in.
  BUGGY: Here are some problems to test your theory about the bug. What is:
               21
             + 39
             ────
   Team:   51
 BUGGY: That's not the bug I have. Using my bug:
               21
             + 39
             ────
               15
             Try giving me more examples ...
   Team:     51    99   68
          + 1707  + 99  + 9
          ──────  ────  ───
  BUGGY:   21   36   23
  Team: Got the bug! The student is adding all the digits together.
 BUGGY: Very good. My description of the bug is: The student always sums up all
            the digits, with no regard to columns.
```

Figure A.16: An interaction protocol with BUGGY [Burton, 1982]

The BUGGY model was extended to a sophisticated off-line diagnostic system for dealing with multiple bugs, DEBUGGY which analyses tests taken by students and an on-line version IDEBUGGY which diagnoses the student's procedure incrementally while he is involved in problem-solving.

In their efforts to address the inability of BUGGY to automatically derive primitive bugs from correct skills, Brown and VanLehn [1980] proposed the REPAIR Theory as an information-processing model of the rational genesis of bugs. The theory builds on BUGGY's extensive data to achieve the explanatory power the BUGGY's diagnostic models were lacking. The BUGGY model diagnosed systematic errors that students are observed to make, while the REPAIR Theory provides procedures and constraints that will account for the appearance of the bugs observed. The use of the REPAIR Theory did help BUGGY predict several unobserved bugs.

VanLehn [1983] proposed the STEP Theory to be coupled with the REPAIR Theory in an effort to address the inability of the REPAIR Theory to explain or model the genesis of an incorrect procedure by the student that gave rise to a bug and instead of trying to overwrite this procedure through relevant problem-solving to try and correct the procedure. The STEP Theory through successive lessons transforms functional subsets of the skill into correct and complete versions.

A.5 Knowledge Based Tutoring Systems

There have been many good reasons why existing Knowledge Based Expert Systems seem to offer an ideal basis on which to build Intelligent Tutoring Systems, other than the obvious fact that they embody large amounts of expert knowledge! One advantage of these systems is the usual separation of a knowledge base of (usually) production rules from the procedural interpreter that uses them. This allows access to modular pieces of knowledge, which are expressed declaratively and can often be understood independently. In addition, explanation facilities have been developed to justify the behaviour of some systems. They can trace the chains of inferences, thus offering explanations of both how the reasoning has led to the conclusions the system proposes and why the system needs certain pieces of information when it requests data from the user. A Knowledge Based Expert System with good explanation capabilities, can only justify its actions passively. To be able to present knowledge actively, it is acknowledged that an Intelligent Tutoring System must be endowed with the ability to select instructional material, to be sensitive to the student and to conduct an effective interaction.

The application of Knowledge Based Systems in Intelligent Tutoring System was sparked when Clancey [1982] undertook the task of building an Intelligent Tutoring System on top of MYCIN. At that time, the domain independent infrastructure of MYCIN, its reasoning engine had been extracted and made into the generic system EMYCIN, which had been tested for applicability in various domains. The developers of EMYCIN hoped that a tutor built for MYCIN would be able to handle any EMYCIN domain with a minimum of modifications, and that the principles underlying such a tutor would even be applicable to Knowledge Based Expert Systems in general. In the rest of this section, we examine Intelligent Tutoring Systems which have been developed using the Knowledge Based Systems Engineering paradigm.

PROUST

PROUST [Johnson and Soloway, 1985] is an Intelligent Tutoring System for Pascal Programs Analysis. It came out of the MENO Project which was an attempt to built an

Intelligent Tutoring System for novice Pascal Programmers. The objective with PROUST was to reconstruct a plausible program-design process so as to provide a problem-specific context for the recognition and discussion of bugs rather than explaining the origins of misconceptions in programming knowledge with a generative theory of bugs.

The argument in developing PROUST is that diagnostic methods that look for bugs in computer programs merely inspecting the code cannot cope with a wide variety of programs. Such methods fail to recognise that nonsyntactic bugs are not an intrinsic property of the fault program, but reside in the relation between the programmer's intentions and their realisation in the code. This makes code inspection insufficient and plan-recognition techniques, when used in isolation are easily thrown off by faulty code and by complex interactions between various goals and between different plans that implement them. PROUST intention-based program analysis is a comparison of intended functions and structures to actual ones. PROUST diagnosis approach distinguishes between three levels: problem specifications give rise to an agenda of goals and subgoals, which in turn lead to the selection of plans, which are finally implemented as code. The exact set of intentions underlying a program is usually not available as data, but must be reconstructed on the basis of evidence provided by the problem specifications given to the programmer and by the program proposed as a solution. The rainfall problem in Figure A.17 is an example of the programming assignments that PROUST can deal with.

Included in the figure is the formal description of the problem given to PROUST as input along with the student program to be analysed. PROUST would then search for the most plausible interpretation of the program with respect to these specifications. PROUST needs to infer a plausible design process that reproduces the programmer's intentions in an

Original Problem statement
Noah needs to keep track of rainfall in the New Haven are in order to determine when to launch his ark. Write a Pascal program that will help him to do this. The program should prompt the user to input numbers from the terminal; each input stands for the amount of rainfall in New Haven for one day. Note: since rainfall cannot be negative, the program should reject negative input. Your program should compute the follwoing statistics from this data:

1. the average rainfall per day;
2. the number of rainy days;
3. the number of valid inputs (excluding any invalid data that might have beenread in);
4. the maximum amount of rain that fell on any one day.

The program should read data until the user types 99999; this is a sentinel value signaling the end of input. Do not included the 99999 in the calculations. Assume that if the input value is non-negative and not equal to 99999, then it is valid input data.
Problem statement as input to PROUST (slightly simplified for readability)
Objects: ?DailyRain is of the class "scalar measurement"
```
Goals: Sentinel-controlled input sequence (?DailyRain, 99999)
       Loop input validation (?DailyRain, ?DailyRain < 0)
       Output (Average (?DailyRain))
       Output (Count (?DailyRain))
       Output (Guarded count (?DailyRain))
       Output (Maximum (?DailyRain))
```
Figure A.17: A programming assignment for PROUST [Johnson and Soloway, 1985]

analysis by synthesis theme. The method combines reconstruction of intentions with detection of bugs together, because bugs can lead to misinterpretations of intentions, and intentions are necessary to distinguish bugs from unusual but correct code.

PROUST as a Knowledge Based System relies on a detailed knowledge base that provides information about the types of program expected to encounter. The knowledge base is not an attempt to reproduce the design process that novices follow. It combines expert knowledge about programming with knowledge about programming errors.The components of PROUST's knowledge base are:

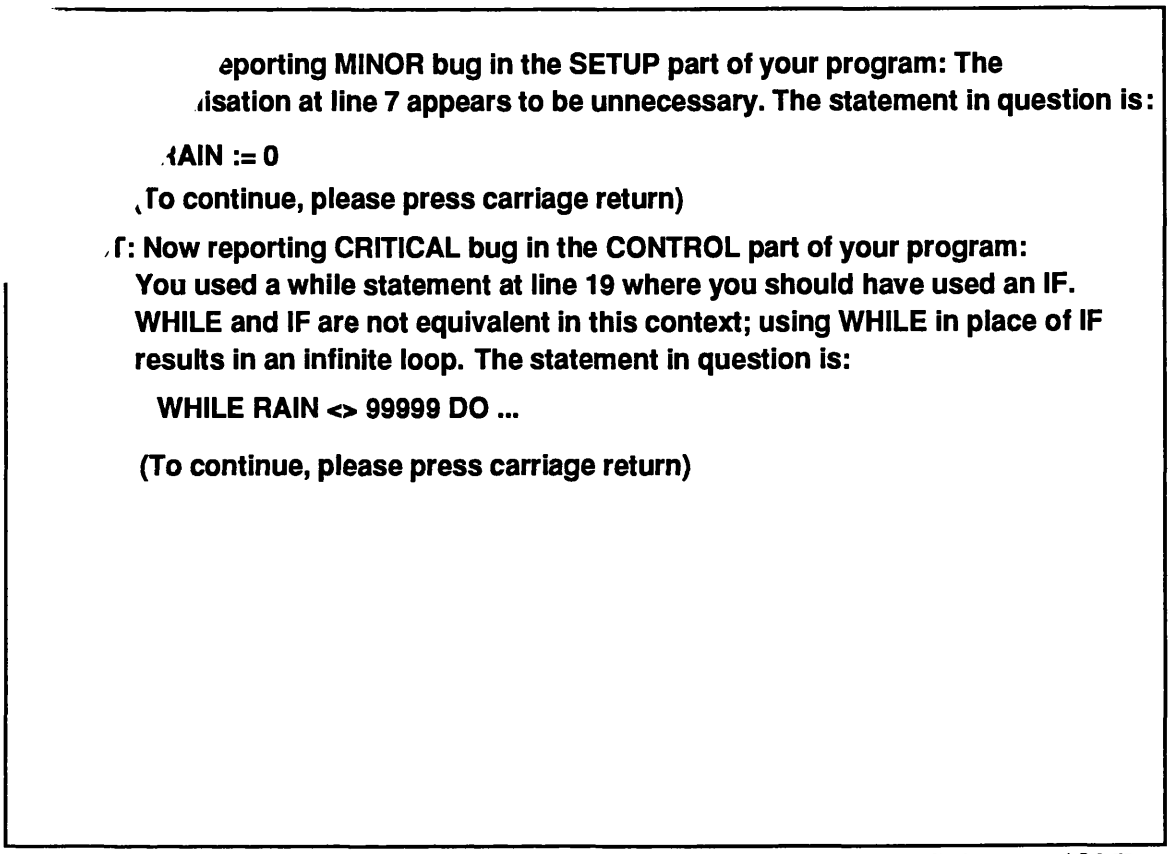
Goals and object classes for problem specifications and the ways in which they can be implemented or reformulated, implicit goals and objects that have to be inferred and can sometimes be omitted in the problem statement along with heuristics rules that can detect goal interactions and generate new goal expectations in connection with certain errors.

Plans list indexed by the goals they achieve including information about incorrect applications of plans along with some buggy plans.

Code consists of two types of rules to deal with plan differences: transformation rules which check for equivalence between two versions of a piece of code and bug rules that explain mismatches by hypothesising a bug of a known type.

With this knowledge, PROUST tries to construct an interpretation for the program to be analysed. Starting with a goal agenda derived from the problem specifications, PROUST selects successive goals for analysis and after performing any applicable reformulation or decomposition in terms of other goals, PROUST searches for corresponding implementations for which there is evidence in the code. Hypothesised plans are then evaluated according to how well they match the code and how well they fit in the context of the overall interpretation. Transformation and bugs rules are then applied on the code. Competing hypotheses are compared to one another to examine how much code they can explain and how bad the students misconceptions are. Figure A.18 shows an example of a program report generated by PROUST.

After PROUST has converged on one interpretation, it evaluates its reliability by measuring how fully it accounts for elements of the code and the specifications by detecting any flaws. It may discard parts of its analysis and thus warn the student about the completeness of its interpretation. The it sorts bugs to be reported, trying to group them so that it can point to common underlying misconceptions. Figure A.19 shows



```
      eporting MINOR bug in the SETUP part of your program: The
     ısation at line 7 appears to be unnecessary. The statement in question is:

    ⁊AIN := 0

   ‚To continue, please press carriage return)

  ‚r: Now reporting CRITICAL bug in the CONTROL part of your program:
   You used a while statement at line 19 where you should have used an IF.
   WHILE and IF are not equivalent in this context; using WHILE in place of IF
   results in an infinite loop. The statement in question is:

     WHILE RAIN <> 99999 DO ...

   (To continue, please press carriage return)
```

Figure A.18: A program report generated by PROUST [Johnson and Soloway, 1985]

another interaction protocol with PROUST.

MENO-TUTOR

MENO-TUTOR [Woolf and MacDonald, 1984] complements the diagnostic abilities of PROUST by addressing the issue of remediation. MENO-TUTOR uses a domain-independent discourse strategy represented as a discourse management network, a kind of augmented transition network, which is to be coupled with a domain-dependent langauge generator that implements strategic decisions from the domain knowledge base, in this case PROUST's knowledge base. The nodes or states in this network correspond to tutorial actions. These states are hierarchically organised into three strategic layers that make the pedagogical decision process transparent. The links indicate hierarchical dependencies whereby actions at one level are possible refinements of the actions at the level above. The arcs of the network define the sequences of states normally traversed by

Here is a programming exercise called the averaging problem: Write a program that reads in a sequence of positive numbers, stopping when 99999 is read. Compute the average of these numbers. Do not include the 99999 in the average. Be sure to reject any input that is not positive.

(a) Student's answer:

```
PROGRAM Average (Input, Output);
 VAR Sum, Count, Val, Avg: REAL;
 BEGIN
  Sum := 0;
  Count := 0;
  Writeln ('Enter Value:');
  Read (Val);
  WHILE Val <> 99999 DO
   BEGIN
    WHILE Val <= 0 DO
     BEGIN
      Writeln ('Invalid entry reenter');
      Read (Val);
     END;
    Sum := Sum + Val;
    Count := Count + 1;
    Writeln ('Enter Value:');
    Read (Val);
   END;
  IF Count > 0 THEN Writeln ('No data entered') ELSE
   BEGIN
    Avg := Sum/Count;
    Writeln ('The average is ',Avg);
   END;
 END.
```

(b) PROUST's output:

You are missing a sentinel test. If a sentinel value is input immediately following a non-positive value, your program will treat it as valid data.

To see this, try the following data in your program:
5 -5 99999

Figure A.19: An interaction protocol with PROUST [Johnson and Soloway, 1985]

the tutor. Figure A.20 shows a dialogue produced by the MENO-TUTOR for the buggy program on the average problem traced by PROUST in Figure A.19.

```
  Tutor: Do you know that DATA, in line 8, is a control variable for the WHILE  loop?
Student: Yes
  Tutor: Does that mean that DATA should be
                 a) Initiated before the loop?
                 b) Added into sum within the loop?
                 c) Modified within the loop?
                 d) Both a) and c)
Student: C
  Tutor: You seem to understand about the control variable of the WHILE loop.
           Now let us talk about some other command statements. What do you think
           READ (DATA) in line 7 does with the value of DATA?
                 a) Reads all the values of DATA at one time.
                 b) Reads in a single value of DATA.
                 c) Assigns several new input values to DATA.
Student: A
  Tutor: Then do you think that DATA := DATA + 1 updates the next of those  values?
Student: Yes
  Tutor: There is your misconception. Adding one is a common way to step through a
           range of values. But it cannot apply here because your READ instruction in line
           7 retrieves only a single value. You should replace line 12 by a second call to
           READ (DATA).
```

Figure A.20: Interaction protocol with the MENO-TUTOR [Woolf and MacDonald, 1984]

The main purpose of the MENO-TUTOR is to serve as a generic tool for exploring tutoring strategies. The hierarchical network provides a set of tutoring primitives with default sequences, so that a variety of pedagogical approaches can be generated.

GUIDON

GUIDON [Clancey, 1987] is a tutor build on top of MYCIN, with the following goals: assessing the pedagogical usefulness of MYCIN's knowledge base, uncovering the additional knowledge a tutoring system would require and attempting to express tutorial strategies in domain-independent terms. For the first version of GUIDON, Clancey did not modify MYCIN. The rule base and the consulting facilities serve as a core, which is extended with new capabilities and other augmentations to produce an active tutor. The central theme of GUIDON was the complete separation of domain knowledge from pedagogical knowledge. The separation allowed for various experiments with different strategies. This also makes the tutor easily adaptable to other domains. The pedagogical goal of GUIDON is to impart MYCIN's rules to a student and thus its medical expertise.

The pedagogical strategy adopted for GUIDON's presentation is the case method, a mixed-initiative dialogue which concentrates on successive specific cases so as to convey MYCIN's knowledge to students in a realistic problem-solving context. This mixed-initiative dialogue resembles partly the Socratic method used in WHY and partly the issues and examples used in WEST. GUIDON's purpose was to guide the student's reasoning as a Socratic Tutor does using MYCIN's rules as issues for discussion in the context of examples. Once a case has been selected and described by GUIDON, the student acting as a diagnostician, asks questions to gather important data and proposes hypotheses. By applying differential modelling, the student's behaviour is compared to the expert behaviour (i.e. MYCIN's). The tutorial program intervenes when the student asks for help or when his actions are suboptimal. Figure A.21 shows an interaction protocol with GUIDON.

For tutoring, GUIDON needed the final conclusions from MYCIN and also information about its entire reasoning process. MYCIN serves as the domain model for GUIDON. This was communicated to GUIDON in the form a tree which also contained the goals accomplished by each rule. This formed the basis for the tutorial dialogue (as seen in Figure A.21). The tutor follows its tutoring strategies to select issues it wants to bring to the attention of the student for discussion. Furthermore, to support explanations given by GUIDON, pieces of canned text justifying inferences or references were attached to individual rules in MYCIN. Also metarules, which are rules about how to use rules, in the

Student: I think that the organisms present are Diplococcus-pneumoniae, Pseudomonas-aeruginosa, and Neisseria-meningitidis.

GUIDON: Good. The age of the patient gives evidence for concluding that Diplococcus-pneumoniae might be present (Rule 507). The fact that the patient has been seriously burned is good reason to supsect Pseudomonas-aeruginosa (Rule 578). Both conclusions apply because the infection which requires therapy is meningitis no organisms were seen on a gram stain of the culture, and the type of infection is bacterial.
The age of the patient also leads one to suspect Neisseria-meningitidis (Rule 507). Do the fact that the infection was acquired while the patient was hospitalised and information about diseases to which the patient has been exposed support this conclusion?

Student: No.

GUIDON: Right. This would be a good reason to rule out Neisseria-meningitidis (Rule 561). What facts about this case tell you that the organisms which might be present are E. coli and Klebsiella-pneumoniae?

Student: Help

GUIDON: Choose from:

1. The patient does not have a TB risk factor and the patient's x-ray is not abnormal
2. Ottidis-media and Epigolottitis are not among the diagnoses consistent with the patient's clinical history
3. The white count (WBC) from the patient's peripheral CBC is less than 2.5 thousand
4. The infection was acquired while the patient was hospitalised
5. The patient was seriously burned
6. None of these

Student: 1 3 4

GUIDON: You're right about the WBC (rule 557) and the evidence gained from the fact that the infection was acquired while the patient was hospitalised (Rule 545). However, the lack of a TB risk factor and a normal x-ray is weak evidence that Mycobacterium-TB is not one of the organisms which might be present (Rule 160).

Figure A.21: An interaction protocol with GUIDON [Clancey, 1987]

form of metaknowledge were also added to MYCIN.

GUIDON's tutoring model is also a rule-based expert system with structure similar to

MYCIN, however, its rules never mention any fact specific to MYCIN's domain. Its domain expertise is the management of a tutorial interaction following the case method. To adapt the dialogue to the specific needs of a given teaching session, the tutor production system maintains and refers to a separate database of facts relevant to the interaction. This database which Clancey calls the communication model encompasses three different parts: the student model, the case syllabus and the focus record.

The student model is a simple overlay model. GUIDON does not model misconceptions, thus it does not use bug or bug part libraries and thus no buggy rules to interpret the student's actions or model his behaviour. GUIDON's overlay uses three values per rule to indicate, the certainty of the rule, its belief that the student could apply the rule in a given circumstance and its belief that he did apply it to produce his current statements. The case syllabus contains information about the relative importance of topics and thus serves to determine the future topics to be covered. The focus record keeps track of the global context of the dialogue in a set of registers so as the ensure of a certain continuity. GUIDON's tutorial rules embody knowledge about discourse procedures and dialogue patterns, and about updating processes for the communication model. Tutorial rules address the basic issues of finding opportunities to intervene, selecting relevant information, and presenting it. Also, they respond to the student's hypotheses and guide him towards understanding how they fit with known information. GUIDON updates the student model with sophisticated (at that time) diagnostic reasoning.

GUIDON's limitations are its first its inability to manage the dialogue when a student follows a diagnostic strategy different from MYCIN's. This would lead to MYCIN rejecting reasonable hypotheses by the student. Second, the complexity of MYCIN's rules especially the strategies and knowledge organisation which they embody, made MYCIN's rules hard to understand and remember especially for experimental work. Clancey [1987] addressed these problems by reconfiguring MYCIN's compiled rules in order to obtain an explicit model of diagnostic thinking. This gave birth to NEOMYCIN.

With NEOMYCIN strategic knowledge is now separated for from domain facts and rules. MYCIN's black-box backward chainer is replaced by an explicit control structure which is domain-independent set of metarules that explicitly represent a hierarchically organised reasoning strategy for medical diagnosis which Clancey calls metastrategy. This metastrategy is another rule-based expert system in the strategic domain used to control an expert system in the object domain. With NEOMYCIN, all domain inferences and data requests take place under the control of the metastrategy, which fires domain rules itself using forward chaining. In addition to the MYCIN interpreter being expressed in terms of an explicit reasoning strategy, the knowledge base containing the rules is also reconfigured so that its structure provides the type of information required for the metastrategy. With NEOMYCIN, strategic information embedded in the form of If-Then clauses in MYCIN's production rules become explicit reasoning strategies. Domain knowledge is organised into coherent epistemic categories like principles, facts, causal relations, heuristic rules, etc.

The GUIDON project is regarded by the Intelligent Tutoring Systems community as one of the most sophisticated Tutoring System ever built. GUIDON came very close to being described as a full-scale knowledge communication system because its representation of knowledge and processes reflects the human approach to the domain. It is worth mentioning that the GUIDON project provided the basis for Wenger model of Knowledge

Communication Systems. In his effort, to develop a generic tutoring paradigm, Clancey launched the GUIDON2 project which aims to provide domain-independent tutor modules for the whole class of problem-solvers typified by NEOMYCIN. Out of NEOMYCIN, came HERACLES, a generic system that captures the domain independent mechanisms of NEOMYCIN. This includes a reasoning strategy and a language of relations between objects which organise domain knowledge so that it can be reasoned about. Two Intelligent Tutoring Systems developed with HERACLES are IMAGE and ODYSSEUS.

Clancey's aim with HERACLES was to create a family of complementary instructional modules. Out came in 1985 GUIDON-WATCH, a graphic animation interface with multiple windows running on workstations in order to make the reasoning strategy of HERACLES inspectable by the student. In 1986, out came GUIDON-MANAGE, a problem-solving environment where the student manipulates a set of operators whose detailed problem-solving effects are implemented by the system. These operators are expressed in terms of a patient-specific model, a causal graph linking conclusions to findings. Finally, in 1987 out came GUIDON-DEBUG, a module that allows the student to modify the knowledge base by criticising problem-solving sessions through the ODYSSEUS interface. The student is able to appreciate how domain knowledge is organised to support instantiations of the reasoning strategy and how it affects the course of the diagnostic process. By introducing bugs into the domain knowledge base which the system invites the student to correct, learning of specific pieces of coded knowledge takes place.

The LISP and GEOMETRY Tutors

John Anderson's Adaptive Control of Thought (ACT^*) Theory of Cognition that levies a great deal of emphasis on skill acquisition provided a theoretical ground for the validity of their proposed Advanced Computer Tutoring Principles (ACTP). The application of these principles in the domain of Lisp Programming and Geometry saw the development of the Intelligent Tutoring Systems classics, The LISP and GEOMETRY Tutors [Anderson, Boyle and Yost 1985] [Anderson and Reiser, 1985].

The ACT* Theory's first assumption is that cognitive functions can be represented as sets of production rules. To this end, Anderson and his colleagues at Carnegie-Mellon University developed GRAPES [Anderson and Reiser, 1985], the Goal-Restricted Architecture for Production Expert Systems. GRAPES productions are strictly interpreted within a hierarchical goal structure. This means that the use of a production rule is determined both by the state of the system and by the goals.

The second assumption concerns the mechanisms of the learning model. According to the theory, knowledge is acquired declaratively through instruction and it has to be converted and reorganised into procedures through experience. The learning mechanism is called knowledge compilation, which comes in two forms: proceduralisation, in which a general piece of knowledge is converted into a specific production to apply to a special class of cases, and rule composition in which a few rules used in sequence to achieve a goal are collapsed into a single rule that combines their effects.

The third assumption for teaching concerns the size of memories. Since the individual rules do not disappear after they participate in composition, there is, therefore, no limit on the size of long-term memory.

Both the LISP and GEOMETRY Tutors function as individualised problem-solving guides. The tutors communicate with the student in terms of the various tasks required for, for instance, constructing a proof in the GEOMETRY tutor or designing a computer program in the LISP tutor. These tasks are viewed as different problem-spaces in which production rules must be selected for following and guiding the student. The LISP Tutor can function in four distinct problem spaces to cover issues of design and of coding: means-end analysis for sequences of operations, problem decomposition, case analysis, and Lisp coding. The first three combined are what the developers of the LISP Tutor call Planning.

The production rules of the GRAPES model of Lisp programming are the units of knowledge the tutor is trying to communicate. With GRAPES the production rules are a modular representation language that encodes cognitive processes. In addition to the ideal model represented by correct rules, the tutor's knowledge base also contains a buggy model whose mal-rules are buggy variants of the ideal model's rules. This enables a simulation of a wide variety of correct and incorrect behaviours for the domain. With both tutors, each lesson makes use of a different rule set, especially tailored to the needs of its specific level. Therefore, each set of rules is limited to the expertise of the ideal student at the corresponding level. The explicit goal structure of GRAPES production rules allows the tutor, at the local level, to relate its explanations to the current situation and to present the rules in a context where their relevance to problem-solving goals is clear and it also allows the tutor, at the global level, to decompose the problem into a hierarchy of explicit goals and subgoals thus enabling the student to remember it along with the actual form of the resulting Lisp function.

The mechanism of knowledge compilation is triggered by the successful application of productions in the achievement of a goal, it does not support useful learning during explorations of erroneous paths. This lead the authors of the LISP and GEOMETRY Tutors to incorporate in the GRAPES model the ability to provide immediate feedback on errors. To this end, the tutors monitor every keystroke by the student and intervene as soon as they perceive a meaningful error. Nevertheless, both tutors will leave the student explore correct but fruitless paths of inference before any tutorial intervention. Tutoring rules associated with ideal and buggy rules provide the student with various levels of explanation. Both tutors use the expertise of their problem solver to predict the steps the student will take. While a student is working on a problem, the problem solver generates all the possible next steps, correct and incorrect according to its rules. These are compared to the student's step and the rule that matches is selected as an interpretation of the student's action. If the tutors cannot find such a rule then their ability to continue tutoring deteriorates dramatically. They both resort to usual prompt of "I do not understand the last input" and after a few trials, it suggests the best next step according to its ideal model. This is the model tracing technique which makes the student follow the system's model quite closely. Model tracing makes sure that the student never strays far from a correct solution path. The problem is that the student model is always limited to one rule local rule or mal-rule whose output match the student step. The resulting student model with both tutors is at its best an incremental overlay on the ideal and buggy models of the student's knowledge.

In trying to minimise the load on human working memory and thus allow the student to concentrate all his efforts on problem-solving, both tutors maintain plenty of contextual information on the screen, inspect his student model, knowledge to which he has been exposed to, previous problem solving episodes, etc. For an interaction protocol with the LISP Tutor see Figures A.22a and A.22b.

In the new versions of the LISP Tutor, the problem-solving models are based on a new architecture, the PUPS (PenUltimate Production System) production system whose control mechanisms are more flexible than those of the GRAPES model. The system does not impose a programming approach upon the student but allows the student to move around the program to expand any unfinished portion. The PUPS problem-solver is presented with a programming exercise and uses its correct and buggy rules to generate a solution trace that the model can possibly recognise for the given exercise.

Define rightp. It takes 3 arguments, side1, side2, side3, which represent the lengths of the three sides of a given triangle, where side3 is the longest side. rightp tests whether the three sides form a right triangle. In a right triangle, the sum of the squares of side1 and side2 is equal to the square of side3. However, your function should allow for a measurement error of 2 percent. In other words, your function should return t if the sum of the squares of the first two sides is within 2 percent of the square of the third side.

For example (rightp 3 4 5.01) = t, because 3 squared + 4 squared = 25, which is within 2 percent of 5.01 squared.

```
1. (defun rightp (side1 side2 side3) (cond))
```

Yes, you could use cond here, but there is a much easier way to test if the sum of the squares of the first two sides is within 2 percent of the square of the third. You can use a predicate, since we only have one test to do.

```
2. (defun rightp (side1 side2 side3)
     (iessp (difference) <NUM2>))
```

You want to code the amount of difference between the sum of the squares of the first two sides and the square of the third side, and you don't care which of those two numbers is larger. So we want to take the absolute value of the their difference.

```
3. (defun rightp (side1 side2 side3)
     (lessp (abs (difference <NUM3> <NUM4>)) <NUM2>))
```

Which do you want to do first?
PRESS: IF YOU WANT TO:
1 Write code to get the sum of squares of the first two sides
2 Write code to get the square of the third side.
Menu choice: 1

```
4. (defun rightp (side1 side2 side3)
     (lessp (abs (difference (plus) <NUM4>)) <NUM2>))
```

No. You should create a helping function for this code, rather that coding it directly. Our function, rightp, is now 4 levels of embedding deep, and the relation we are about to code -the sum of squares of two numbers -would be a useful helping function. Thus, let's code the sum of the squares of the first two sides using a helping function.

```
5. (defun rightp (side1 side2 side3)
     (lessp (abs (difference (sumsq <ARGUMENTS>) <NUM4>)) <NUM2>))
```

What do you want the helping function sumsq to do?
PRESS: IF YOU WANT TO:
1 Take 2 arguments, the square of the first side and the square of the second side. Compute their sum.
2 Take 2 arguments, the first side and the second side. Compute the sum of the squares of these numbers.
3 Have the tutor choose.
Menu choice: 1

Figure A.22a: An interaction protocol with the LISP Tutor [Anderson and Reiser, 1985]

No. We wouldn't need a helping function just to compute the sum of two numbers. You should create your helping function so that it does more than just a sum. It would be better to define sumsq so that it takes two arguments, in this case. [1] the first side, and [2] the second side, and computes the sum of the squares of these numbers.

Menu choice: 2

```
6. (defun rightp (side1 side2 side3)
     (lessp (abs (difference (sumsq side1 side2) (times))) <NUM2>))
```

You should use a helping function that you have already coded, square. Since this function computes the square of a number, you can use it to code the square of the third side.

```
7. (defun rightp (side1 side2 side3)
     (iessp (abs (difference (sumsq side1 side2) (square side3)))
      (plus)))
```

I don't understand that, Rob. How are you going to code 2 percent of the square of the third side?

[EXPLAIN]

OK. I'll explain what I wanted. You could either code this directly or create a helping function here to code 2 percent of the square of the third side. Let's create a helping function just to make things easier. We'll call this helping function percentage. We will define percentage when we are finished coding rightp.

What do you want the helping function percentage to do?
PRESS: IF YOU WANT TO:
1 Take 2 arguments, one hundreth of 2 and the square of the third side. Compute their product.
2 Take 2 arguments, 2 and the third side. Compute the given percentage of the square of its second argument.
3 Take 2 arguments, 2 and the square of the third side. Compute the given percentage of its second argument.
4 Take 1 argument, the square of the third side. Compute 2 percent of its argument.
5 Have the tutor choose.
Menu choice: 3.

Very good.

```
8. (defun rightp (side1 side2 side3)
     (lessp (abs (difference (sumsq side1 side2) (square 3)))
      (percentage 2 (square side3))))
```

OK. Now we have finished coding the function rightp.

Figure A.22b: An interaction protocol with the LISP Tutor [Anderson and Reiser, 1985]

APPENDIX B: QUESTIONNAIRE

PART A

Subject knowledge

Q1. Can the systems answer arbitrary questions from the user about the subject?

Q2. Can the systems give an explanation of a problem solution (including one of a problem posed by the user)?

Q3. Can the systems give alternative explanations, perhaps using analogy?

Q4. Can the systems answer hypothetical questions, that is, questions not about the present situation but about some imagined situation relating to it?

Student knowledge

Q5. Could the systems give a report on the student's level of understanding?

Q6. Are the systems' explanations tailored to the user?

Q7. Do the systems provide informative feedback?

Q8. Are the problems presented by the system adapted to the users' needs?

Student control

Q9. Do the systems teach prerequisite skills?

Q10. Do the systems maintain control over the whole tutoring endeavour or does it leave part or all of the initiative to the student?

Q11. Do the systems "actively" engage the user?

Q12. Do the systems enable the student to communicate his plans (i.e. intentions) prior to executing them?

Q13. Do the systems motivate and support a flexible style of tutoring?

Q14. Can the user initiate some new area of investigation?

Q15. Do the systems support the various idiosyncratic ways which the student might choose to solve a problem?

Q16. Do the systems monitor the student step by step?

Q17. Do the systems monitor such proposed changes, and comment upon them if they seem to be unwise?

Q18. Do the systems intervene if the user appears to be having difficulty?

Q19. Do the systems remedy in a problem-solving context?

Q20. Do the systems provide hints, pieces of advice, corrections, remedial demonstrations, traces of reasoning, interpretations, explanations, simulations, motivation?

Mode of communication

Q21. Do the systems provide an environment in which the interaction between it and the student is close to reality?

Q22. Can the user express his inputs to the system in whatever way is most natural?

Q23. Do the systems help if the users' input is not understandable by the systems?

Q24. Are the systems' outputs natural?

PART B

General

Q25. Are the systems robust, especially with respect to user input?

Q26. Are the systems helpful, especially when the user gets stack?

Q27. Are the systems simple to use?

Q28. Are the systems perspicuous or do they provide the user with mystifying choices?

Q29. Are the systems "powerful" enough in terms of graphics facilities, explanations, etc.?

Q30. Are the systems navigable or can the users easily get lost?

Q31. Are the systems consistent or do they behave differently in different situations?

Q32. Are the systems transparent, especially with respect to the effect of students' actions?

Q33. Are the systems flexible enough to accommodate tutoring for different classes of users?

Q34. Do the systems enable redundancy, that is different views of the subject material?

Q35. Are the systems sensitive to the individual student needs with respect to tutoring?

Q36. Are the systems omniscient enough to lead the users sometimes "by the hand"?

Q37. Are the system docile or are the users sometimes in command?

PLEASE FEEL FREE TO MAKE ANY ADDITIONAL COMMENTS

Thank you for answering this questionnaire. Now, please, come and have a chat with me about the two systems.

Marios Angelides

APPENDIX C: A NOVICE'S GUIDE TO INTELLIGENT TUTORING SYSTEMS

C.1 Leading experts' definitions

Given below are leading experts' definitions on what an Intelligent Tutoring System should be and what it should do.

Intelligent Tutoring Systems are educational devices which by incorporating Artificial Intelligence understand what, whom and how they are teaching and can therefore tailor content and method to the needs of an individual learner without being limited to a repertoire of prespecified responses as happens with conventional computer assisted instruction systems. [Dede, 1986].

Intelligent Tutoring Systems are instructional programs that use Artificial Intelligence techniques to incorporate well-prepared course material in lessons optimised for each individual student. [Clancey, 1987].

Intelligent Tutoring Systems are computer programs that use Artificial Intelligence techniques to help a person learn a topic. [Kearsley, 1987].

An Intelligent Tutoring System has a well-articulated curriculum that embodies units of domain expertise and an explicit theory of instruction represented by its tutoring strategies. This completeness permits an Intelligent Tutoring System to package existing expertise and focus on the novelty, with the use of its mechanically embodied sets of rules as a tool for instruction. [Lawler and Yazdani, 1987], [Yazdani and Lawler 1991].

Intelligent Tutoring Systems are concerned with the provision of situated help via models which support local or "real-time" assessment of the actions of the computer user. The primary objective of such systems is to infer the user's knowledge and misconceptions about the system by observing his actions, rather than relying on either error conditions or explicit requests for help. [Suchman, 1987].

Intelligent Tutoring Systems are Artificial Intelligence based Knowledge Communication Systems which possess the ability to cause and/or support the acquisition of one's knowledge by someone else via a restricted set of communication operations. [Wenger, 1987]

Intelligent Tutoring Systems are computer systems developed to provide the student with the same instructional advantage that a sophisticated human tutor can provide. A good human tutor understands the student and responds to the student's special needs. [Anderson, 1988].

An Intelligent Tutoring System is able to analyze a wide range of student responses, model the student's current knowledge state (including misconceptions), teach in a variety of ways, diagnose and/or determine what and when to teach, and is able to engage in appropriate interactive conversations. [Brecht and Jones, 1988].

Intelligent Tutoring Systems are a form of computer-based learning which incorporates Artificial Intelligence Techniques such as knowledge representation and natural language processing in order to adapt better the computer instruction to the needs and interests of the students. [Duchastel and Imbeau, 1988].

Intelligent Tutoring Systems are systems that can make inferences about student knowledge and can interact intelligently with students based upon individual representations of what those students know. [Mandle and Lesgold, 1988]

An Intelligent Tutoring System is a computer-based system intended to provide effective, appropriate, and flexible instruction through the application of Artificial Intelligence techniques and Knowledge Representations. An Intelligent Tutoring System is distinguished from a traditional computer-based training system by its use of Artificial Intelligence Techniques and Knowledge Representations. [Murray, 1988].

Intelligent Tutoring Systems are computer-based learning programs in which Artificial Intelligence Techniques have been used to represent or reason about topic matter, students or teaching strategies. [Sleeman and Ward, 1988]

Intelligent Tutoring Systems are systems which can adapt their instruction based on a student model derived from an analysis of the standard errors that students make. By running the student model on an actual student's response it becomes possible to predict aspects of the student's current state of understanding and hence to offer appropriate problems, remediation or exposition. [O'Shea, 1989].

Successful Intelligent Tutoring Systems are successful not because they enable a learner to ingest performed knowledge in some optimal way, but rather, because they provide initially undetermined, threadbare concepts to which, through conversation, negotiation, and authentic activity, a learner adds texture. Learning is much more an evolutionary, sense-making, experimental process of development than of simple acquisition. One must, therefore, attempt to use the intelligence in the learning environments to reflect and support the learner's or user's active creation or co-production, *in situ*, of idiosyncratic, highly textured models and concepts, whose texture is developed between the learner/user and the activity in which the technology is embedded. [Brown, 1990].

C.2 A simple guide to the functionality and components of Intelligent Tutoring Systems

Intelligent Tutoring Systems are instructional software systems endowed with the capabilities of a human teacher working on a one-to-one basis with the student, carefully diagnosing what the student knows, how the student reasons, and what kinds of deficiencies exist in his ability to apply his knowledge. The system then uses this inferred knowledge of the student to determine how best to teach a subject to the student [Brown and Burton, 1978]. Given below are some desirable properties of a human tutor that the Intelligent Tutoring System should also possess [Gable and Page, 1980]:

[1]. The tutor causes the problem solving heuristics of the student to converge to those of the tutor.

[2]. The tutor will learn and adopt student solution methods if they are superior.

[3]. The tutor chooses appropriate examples and problems for the student.

[4]. When the student needs help, the tutor can recommend solution scheme choices and demonstrate how to apply techniques.

[5]. The tutor can work arbitrary examples chosen by the student.

[6]. The tutor is able to adjust to different student backgrounds.

[7]. The tutor is able to measure the student's progress.

[8]. The tutor can review previously learned material with the student as the need

arises.

[9]. The tutor will give immediate feedback on errors while allowing the student a free hand in deciding how to solve a problem.

[10]. After the student solves a problem, the tutor may point out more direct solutions or ones that use recently learned theorems or techniques.

Intelligent Tutoring Systems embody knowledge of what is being taught, who is to be taught, and how is to be taught [Nwana and Coxhead, 1988]. Intelligent Tutoring Systems have an explicit representation of the domain-specific knowledge and the problem solving knowledge of the topic, which they try to teach to the user. This enables for a comparison to be made between the behaviour of the user against that of the 'expert'. They are also equipped with teaching expertise [Yazdani, 1988]. They also facilitate diagnosis of the user's performance, competence, and learning preferences. They are capable to explore and understand the user strengths and weaknesses and respond to these accordingly. This provides for a high level of individualization. They are also equipped with the knowledge and ability to help the students clear away any misconceptions. All these three sources of knowledge will be used by the system to build a representation of the user in an effort to individualize teaching or training.

The Knowledge Based Systems approach is by far the most popular approach in developing Intelligent Tutoring Systems. Being a Knowledge Based System, one expects to find some characteristics which are common to all Knowledge Based Systems [Duchastel and Imbeau, 1988]; a knowledge base which contains the knowledge about the domain being learned (i.e. the domain model), some form of natural language processing ability in the form of an user interface which opens up the human-computer interaction beyond short-answer or menu-driven interactions and finally, some kind of inference mechanisms to drive the reasoning process with the domain model.

In addition, an Intelligent Tutoring System builds a working model both of the student (i.e. the student model) in respect to the domain being learned and of the teaching process itself (i.e. the tutor model). The model of the student must include his knowledge about the domain, errors made by the student during the interaction and the student's misconceptions about the domain knowledge. This helps the tutor model to adjust its tutoring to the level of the student. The tutor model by using a set of tutorial rules provides instruction to the student. Finally, the last component of an Intelligent Tutoring System is a bugs library which contains a list of possible, expected errors and misconceptions about the domain being learned. Given below is what each component of the Intelligent Tutoring System is expected to do in greater detail.

The *Domain Model* holds the domain-specific knowledge that the system will try and impart to the student either by direct exposition or through problem-solving. When this knowledge is combined with inference mechanisms, it enables the system to employ it in problem solving situations. The domain model is the source for material for problems that the system will prompt the student to go through and solve. It is also the source of examples, associated explanations and remedial material should the user is diagnosed to suffer from some kind of misconception. The domain model must be able to solve all the problems that has generated for the student, in several ways. The correctness of the student's solutions can be evaluated by reference to the domain model's own solutions. Ideally, the domain model possesses the ability to adopt its solution to students' solution methods and learn from them should they be superior to its own methods of solving a problem.

The *Student Model* represents the student's understanding of the material to be taught. The model must be able to represent knowledge, concepts and skills which the student has acquired, as well as those which the student has been exposed to and for which the student has shown some understanding. The model must be able to represent misconceptions, bugs or erroneous information which the student has been seen or suspected to have acquired. The model must be able to represent the most suitable teaching strategy for the student. All this information is inferred by inference mechanisms from the student's answers and the student's problem solving behaviour with reference to the domain model and the bugs library. Therefore, the model represents a history of the student's responses and problem solving behaviour.

The *Tutor Model* is responsible for providing instruction to the student. It must be able to vary the teaching method for different students and topics. Therefore, the tutor model must have access to knowledge of how to teach, knowledge of what is being taught and knowledge of who is being taught. The domain model provides the knowledge of what is being taught and the student model provides the knowledge of who is being taught and how is to be taught. The most commonly employed teaching strategies in a tutoring system are [Brecht and Jones, 1988]:

[1]. Coaching the student within a particular activity. The tutor manipulates the environment and the coaching so that the student acquires the correct and right set of skills and problem solving ability.

[2]. Questioning the student in order to encourage reasoning about current knowledge

and in order to modify or formulate concepts. The student is offered more flexibility and initiative to manipulate the environment.

[3]. Providing tasks for the student and evaluating the student responses in order to detect the student's misconceptions.

The tutor model must be able to intervene and provide the student with help and explanations when this is asked or when there is a call for them as a result of an error or detection of misconceptions. This is the model, as mentioned by Bumbaca [1988], that communicates with the student through the user interface, selects problems for the student to solve, monitors and criticizes the student performance, provides assistance upon request, selects remedial action, simply knows how to teach, knows when it is appropriate to offer the student a hint, how far the student should be allowed to go down the wrong path. The tutor has specific goals and teaching tactics and follows certain plans to meet the goals. The tutor may be given a flexible character profile which is adjusted depending on the character of the student as represented in the student model.

The Bugs Library is a library of common misconceptions and errors in the domain. These are the possible deviations a student can make from the domain knowledge. The student's answers and the student's problem solving behaviour are checked for correctness against this library.

The User Interface is responsible for the interaction between the student and the system, preferably in the student's own language. The user interface is the front-end to the system and stands between the system and the student.

The interaction between the system and the teacher is usually necessary at least at the following three levels [Carbonell, 1970]:

[1]. Preparation of the domain knowledge base or database.

[2]. Setting the conditions for student/computer interaction, that is define the system parameters necessary to stimulate the conditions of the interaction.

[3]. Collection of results, in the form of scores, statistics, and general history of the student/computer interaction after it has taken place by examining the student models.

There is a fourth role for the human teacher: that of a supervisor in real time of the actual operation of a system. Instead of the typical "Sorry, I do not know", the system can ask the supervisor for an answer. It may be necessary in this case for the system to trace back and record how such a case arises and provide the supervisor with the information.

GLOSSARY OF INTELLIGENT TUTORING SYSTEMS TERMS

Advancement. The use of an algorithm to determine whether to advance the student to the next curriculum topic.

Apprenticeship Teaching System. A situated learning environment in which a novice is given the opportunity of learning by doing with an expert providing feedback and motivation.

Artificial Intelligence. The study of techniques and principles for applying computers to issues normally reserved for human intelligence. Artificial Intelligence systems typically exhibit some characteristics of human intelligence (including silly errors) when learning, reasoning, simulating, or understanding natural language.

Authoring system. A domain-independent component of an Intelligent Tutoring System that allows the developer to enter specific domain knowledge into the tutor's knowledge base.

Backward Chaining. A pattern matching technique that tries to prove the condition part of rules whose actions match the conditions of proven rules. (See *Forward Chaining*).

Bandwidth. The amount of the student's activity available to the diagnostic model. The three categories of bandwidth in Intelligent Tutoring Systems, from low to high are: final states, intermediate states, and mental states.

Black box expert system. A procedure that generates correct behaviour over a range of tasks in the domain, but whose mechanism is inaccessible to the Intelligent Tutoring System. (See *glass box expert system*).

Bug catalogue. A set of well-analyzed and carefully collected patterns of typical errors. (See also *bug library*).

Bug library technique. A student-expert difference model that represents misconceptions. It augments an expert model with a list of bugs.

Bug part library. A student-expert difference model that generates bugs from fragments of valid rules.

Bugs. Student misconceptions in declarative or procedural knowledge.

Case-Based Reasoning. Problem solving based on a collection of individual experiences rather than general rules.

Causal Stories. Causal stories, in troubleshooting context, are elaborate knowledge structures (and narratives drawn from those structures) that relate observable evidence and symptoms to causes of faults through various models and knowledge about the device in question.

Coach. A form of student modelling in which the Intelligent Tutoring System intervenes only when it is fairly sure the student is doing something wrong. The intervention is with graduated hints and examples.

Coarse-grained student model. A student model that does not describe cognitive processes at a detailed level.

Cognitive fidelity. The measure of correlation between the cognitive model and actual human problem solving strategy.

Cognitive model. A representation of human cognitive processes in a particular domain.

Computer Based Instruction. The use of computers in instruction and training. Generally this refers to instruction in which no expert system or production rules are used to order the sequence of information presented. It often results in linear sequences, or chains, of presented material. (However, see *Microworlds*).

Concept Hierarchy. A graph of more and less general topics or ideas.

Condition induction. A diagnostic technique used in the student model for constructing buggy rules for bug part libraries, a student-expert difference model. (See *bug part library*).

Constructivism. A pedagogical philosophy that views learning as *constructing* knowledge, rather than *absorbing* it.

Curriculum module. The component of an Intelligent Tutoring System that selects and orders the material to be presented to the student.

Curriculum selection techniques. Techniques that deal with selecting problems to exercise those areas in the curriculum where the student is weak.

Decision tree technique. A diagnostic technique used in the student model that creates a tree of paths. Each diagnosis corresponds to a path from the root to some leaf.

Declarative knowledge. Knowledge represented as basic principles and facts of a domain. It is usually portrayed as static and structural, for instance, as frames, production rules or semantic networks. Declarative knowledge is usually contrasted with knowing how to use facts, that is, *procedural knowledge* (although the distinction is not always useful).

Deep-level tutoring. Tutoring that can provide explanation of the internal reasoning of its expert module.

Demons. Rules that actively wait for their conditions to become true and fire in dynamic systems.

Diagnostic module. The component (a process) of an Intelligent Tutoring System that infers and manipulates the student model. The selection of a diagnostic algorithm is dependent on the bandwidth of the system.

Didactic Operation. A unit of decision in the tutoring process. It is more general than a didactic intervention, in that it does not necessarily correspond to actions visible to the student. A didactic operation has four characteristic aspects: the plan of action that enacts a didactic operation, the strategic context in which the operation is triggered, the decision base, that provides constraints and resources for the construction of the operation, and the target level of the student model at which the operation is aimed.

Direct manipulation interface. See *first person interface.*

Direct Manipulation. An interface approach that provides simulations (usually visual) that can be altered (visually) to produce corresponding changes in the underlying symbolic representation.

Divergence principle. A curriculum principle that states that there should be a broad representative sampling of exercises and examples in curricula for procedural tutors.

Dynamic Systems. Complex mechanisms that require swift and effective interactions, so that instruction and tutoring must be terse and to the point, and more lengthy instruction delayed to a later debriefing.

Enabling objectives. An instructional objective's immediate prerequisite.

Environment. The component of an Intelligent Tutoring System that specifies or supports the activities that the student does and the methods available to accomplish those activities.

Explanation Based Simulations. Simulations or models whose design are predominantly driven by the need to provide explanations to students about device functions. Veridicality is subordinated to simplicity of explanations.

Expert module. The module of an Intelligent Tutoring System that provides the domain expertise, in other words, the knowledge that the system is trying to teach.

Expert system. A computer program that uses a knowledge base and inference procedures to act as an expert in a specific domain. It is able to reach conclusions very similar to those reached by a human expert.

Expository tutor. A tutor that is concerned with declarative knowledge. Usually interactive dialogue is the instructional tool used in this type of tutor.

External evaluation. Evaluation of an Intelligent Tutoring System that focuses on the impact of the Intelligent Tutoring System on students' knowledge and problem solving.

External-international task mapping problem. A problem in the human-computer interaction component of an Intelligent Tutoring System. It is a gap between what the user wants, the goal of the interaction, and the actions the user must make to achieve the goal.

Fault Diagnosis. A problem-solving technique used to uncover the source of system malfunction.

Felicity conditions. Principles of instruction that facilitate ease of learning, such as presenting only one new step in a procedure per lesson.

Fidelity. A measure of how closely the simulated environment in an Intelligent Tutoring System matches the real world. There are four kinds of fidelity: physical, display, mechanistic, and conceptual.

Fine-grained student model. A student model that describes cognitive processes at a high level detail.

First-person interface. A type of user interface where the actions and objects relevant to the task and domain map directly to actions and objects in the interface. With this interface the user has a feeling of working directly with the domain. An example of this type of interface is the icon.

Flat procedural knowledge. Procedural knowledge that is not organized by subgoals, in other words, an undifferentiated set of production rules.

Forward Chaining. A pattern matching technique that tries to prove the condition part of rules whose actions are then used to prove other rules. (See *Backward Chaining*).

Generate and test. A diagnostic technique used in the student model that generates bug combinations (sets of bugs) dynamically and tests these for validity against student performance.

Glass-box expert systems. An expert system that contains human-like representation of knowledge. This type of expert system is more amenable to tutoring than a black box expert system because it can explain its reasoning.

Goal-factored production system. A rule-based system that makes explicit references to goals in the conditional part of its rules.

Graduated Models. Qualitative models whose power and extension grow in some sort of correspondence with the capabilities of students using them.

Grain-size of diagnosis. The level of detail used by the diagnostic technique for processing student models. Closely related to *bandwidth.*

Heuristics. Rules of thumb that are practical and often work, but are not based on a principled, theoretical understanding and therefore are not guaranteed to work.

Hierarchical procedural knowledge. Procedural knowledge with subgoals.

Hypertext. A text-based system that goes beyond text to include graphics, video, and sound (hypermedia) as well as links, crossreferences, and hierarchical structures. It is interactive so that one word can be expanded on command into other media (hypermedia). The term was coined by Ted Nelson.

Increasingly Complex Microworld framework. A pedagogical technique of exposing the student to a sequence of increasingly complex microworlds that provide intermediate experiences such that within each microworld the student can see a challenging but attainable goal.

Individualization. A curriculum principle that states that exercises and examples should be chosen to fit the pattern of skills and weaknesses that characterize the student at the time of exercise or example is chosen.

Instruction. Actual presentation of curriculum material to the student.

Instructional Amplifier. A computer used to enlarge the scope and powers to teachers for instruction, that lets teachers personalize instruction more than they now can.

Instructional Design. A process of organizing knowledge and selecting frameworks for effective instruction.

Instructional Environment. See *environment.*

Instructional Strategy. A general approach toward teaching or training, including objectives, plans and teaching style.

Instructional Systems Design. A systems engineering approach to the analysis, design, development, delivery, and evaluation of instruction.

Intelligent Computer Assisted Instruction. Synonym for Intelligent Tutoring System.

Intelligent Tutoring System. A computer program that (a) is capable of competent solving in a domain, (b) can infer a learner's approximation of competence, and (c) is able to reduce the difference between its competence and the student's through application of various tutoring strategies. It tries to individualize instruction by creating a computer-based learning environment that acts as a good teacher, correcting mistakes, offering advice, suggesting new topics, and sharing curriculum control. It should have the ability to analyze student responses, develop a history of the learner's preferences and skills, and tailor the material to suit the trainee. Some important subtopics for Intelligent Tutoring System are knowledge representation, simulation, natural language, expert systems, and induction.

Intelligent Tutoring System Architectures. A systematic approach to structuring the many components that comprise an effective, working Intelligent Tutoring System. Usually these consist of a student model, an organized domain of knowledge, instructional principles, and a tutorial interface.

Interactive diagnosis. A diagnostic technique used in the student model that does not use a fixed list of text items.

Internal evaluation. Evaluation of an Intelligent Tutoring System that focuses on the relationship between the architecture of the system and its actual behaviour.

Issue-oriented methodology. A methodology for building an Intelligent Tutoring System that relies on access to intermediate states of cognitive processing. These intermediate states are used to identify instructionally useful issues characteristic of differences between expert and student performance.

Issue-oriented recognizes. Methods that note in student behaviour the presence or absence of issues or characteristic traits of expert performance.

Issue-oriented tutoring. A type of tutoring that bases instruction on patterns of differences in the intermediate cognitive processes underlying student and expert behaviour.

Issue tracing. A diagnostic technique used to construct a student model. A variant of model tracing that relies on access to intermediate states of student performance rather than on access to a highly detailed cognitive process model.

Knowledge Base. Codified knowledge (usually represented on a computer) of a domain or subject matter.

Knowledge Acquisition. The fundamental bottleneck in instructional design for informal systems: How does one acquire and organize the subject matter or knowledge base?

Knowledge level analysis. An internal evaluation method; it attempts to characterize the knowledge in the Intelligent Tutoring System and thus answers the question: *What does the Intelligent Tutoring System know?*

Knowledge Representation. Computer-based techniques for storing and retrieving knowledge organized according to specific principles. Prominent techniques include frames, semantic networks, and object oriented techniques.

Link. An arc that joins nodes in a graph.

Manageability. A curriculum principle that states that every exercise should be workable and every example should be comprehensible to students who have completed previous parts of the curriculum. Manageability applies to procedural tutors.

Matching principle. A curriculum principle that states that both positive and negative instances of concepts, procedures, or principles should be presented.

Mental Model. A popular theoretical construct for a knowledge representation form that supposes that people simulate their environments with models of the world that they are able to run in their minds. These executable mental models can be used to predict the outcomes of thought experiments using novel conditions. Mental models can also be used to trace the causal connections of events and devices in the world.

Microworlds. Computer-based learning environments in which trainees are free to explore and discover the limits of their own understanding. The computer provides little direction or guidance, but it does narrow and constrain the topics for search to those that are valid within the current world. The environments can also raise sharply focused contrasts between alternative hypotheses about the world to facilitate insight and discovery.

Misconception. An item of knowledge that the student has and the expert does not have. A type of student-expert difference. *A bug.*

Missing conception. An item of knowledge that the expert has and the student does not have. A type of student-expert difference. See *overlay model.*

Mixed Initiative Dialogue. An Intelligent Tutoring System environment that accepts and responds in natural language to both solicited and unsolicited natural language input from the student.

Model-tracing. A diagnostic technique used to build a student model. it uses the student's surface behaviour to infer the sequences of rules fired in a rule-based model of performance; that is, the student's actions traced a path through the rule base.

Node. An entry in a graph that is usually labelled and boxed. Often it is a concept or a relation of some sort.

Novices. Students or trainees learning a knowledge domain.

Overlay model. A student-expert difference model that represents missing conceptions, usually implemented as either an expert model annotated for those items that are missing or an expert model with weights assigned to each element in the expert knowledge base.

Path finding. A diagnostic technique used to find a path from one state to the next, which is a chain of rule applications. This is a way of representing the student's mental state sequence. The path is given to the model tracer.

Plan recognition. A diagnostic technique used in student models that represent hierarchical procedural knowledge. It is similar to path finding in that it is a front end to model tracing.

Predicate. A relation defined for a set of concepts. For instance, for "If an apple is red", an appropriate predicate that links "apple" and "red" could be called "colour".

Procedural Knowledge. A form of knowledge representation distinct from Declarative knowledge (although the distinction is not always useful) in which the knowledge is portrayed as active and functional, for instance, functions, objects, demons, and algorithms. Sometimes production systems are viewed as a procedural form of knowledge to distinguish them from the organized declarative structures of semantic networks. Procedural knowledge is usually domain-dependent knowledge about how to perform a specific task.

Procedural tutor. A type of tutor that teaches procedural knowledge, like skills and procedures. Usually exercises and examples are used by procedure tutors.

Process model. A model that reveals the *mechanism* behind behaviour.

Production rule. A rule of the form *condition(s) imply action(s),* used in model cognitive behaviour. A set of production rules and an interpreter for processing them is termed a *production system.*

Program process analysis. An internal evaluation method; it attempts to answer the question: *How does the Intelligent Tutoring System do what it does?*

Propaedeutics. Knowledge that is needed for learning but not for proficient performance.

Qualitative Approximation. Qualitative approximation is a term designated by T.

Govindaraj to refer to the use of difference equation modelling techniques and other good engineering practices to create efficient working models of devices as simulation components of Intelligent Training Systems.

Qualitative Models. A computer-based simulation composed of ordinal or even nominal metrics, such as "good" and "better", rather than higher-order mathematical models.

Qualitative process model. A type of cognitive model, concerned with reasoning about the causal structure of the world; the simulation of dynamic processes in the mind. It is an important facet of troubleshooting behaviour.

Reify. To make concrete and experiential. Within the context of Intelligent Tutoring System, to make something inspectable and interactive.

Repair theory. A generative theory of bugs; a method of deriving bug libraries directly from correct procedures, reducing the need to collect bugs through empirical observation.

Rule-based model. An expert module of an Intelligent Tutoring System that is implemented with a rule-based (production) systems. (Also called a "production model.")

Second-person interface. A type of user interface where the user gives commands to a second party. Examples of this type of interface are command languages, menus, and (limited) natural language interfaces.

Semantic Networks. A graph structure that links concepts with conventional links such as "part-of", "isa", "instance", "super", "class", etc. Often seen as a Declarative form of knowledge. (See *Concept Hierarchy*).

Situated Learning. The context or situation of much expert activity directly supports (learning) the skills the expert has. These skills are otherwise rarely invoked. The result is that learning by doing is cued and accelerated by the environment.

Step theory. A theory that states that curriculum should be divided into discrete lessons, each of which adds a single decision point or step in the procedure to be learned. (See *felicity conditions*).

Structural transparency. A curriculum principle that states that the sequence of exercise and examples should reflect the structure of the procedure being taught and should thereby help the student induce the target procedure.

Student-expert difference. The difference between the expert's knowledge and the student's knowledge. There are two basic types of student-expert difference; missing conceptions and misconceptions. The three models used to represent student-expert differences are: overlay model, bug library technique, and library of bug parts.

Student model. The component (a data structure) of an Intelligent Tutoring System that represents the student's current state of knowledge (mastery) of the domain. This is a detailed model of student cognition. Various student modelling systems have been proposed: bug catalogs, overlay models, issue oriented models, coaching systems, and psychometric systems.

Subject Matter Experts. Subject Matter Experts are knowledgeable in a domain and possess a fragmented, self-imposed organization of things that has considerable pragmatic value in dealing with everyday problems.

Surface-level tutoring. Tutoring that can be implemented with issue-oriented recognisers. Access to the internal reasoning of the expert module is not available.

Target knowledge type. The type of knowledge that is represented in the expert and student model modules. Knowledge representation can be categorized into three types: procedural (both flat and hierarchical), declarative, and qualitative process model.

Technical Manuals. Written descriptions of complex systems, outlining system operation and troubleshooting.

Temporal Fidelity. The degree of veridicality with which the propagation of effects of a change (including failures) in a simulation over time approximates the temporal sequence of changes in the real system.

Tutorial domain analysis. An internal evaluation method for iteratively adding and subtracting requirements of the Intelligent Tutoring System design.

User Interface Management System. A strategy that attempts to separate the interface component of an application program from the computational part.

Web teaching. A curriculum approach where selection of materials is guided by two principles: relatedness (priority is given to concepts that are closely related to existing knowledge), and generality (discuss generalities before specifics). Web teaching applies to expository tutors.

Wizard-of-Oz system. Semiautomated tutors where a human tutor replaces some or all of the instructional functions of an automated tutor. Used in research and development of Intelligent Tutoring Systems.