The Knowledge Engineering Review, Vol 6:2, 1991, 59–95

Towards knowledge-based tutors: a survey and appraisal of Intelligent Tutoring Systems

TOMAS SOKOLNICKI

Application Systems Laboratory, Department of Computer and Information Science, University of Linköping, S-581 83 Linköping, Sweden

Abstract

Intelligent tutoring systems can be seen as a next step for computer-based training systems, but also as an important by-product of knowledge-based expert systems. This paper surveys the development and progress in the area, with a special emphasis on the potential for an emerging engineering discipline as opposed to a mere crafting of systems. Major components in intelligent tutoring systems as realized so far are discussed, and key issues for successful future development identified. Knowledge representation, student modelling, planning, natural language issues, explanations and learning are discussed in more depth as being the cornerstones of both tutoring systems and artificial intelligence. Examples from specific implementations are used to illustrate key points.

In the concluding discussion we present our attempt at dealing with some of the problems facing the area. In the project Knowledge-Linker, we aim at extending the functionality of a knowledge-based system with tutoring capabilities, and suggest one way of explicitly dealing with teaching strategies.

1 Introduction

Throughout the history of computers much interest has been devoted to the idea of using the machine as an intelligent companion, a private teacher with the ability to expose to us the most intricate areas of knowledge. The first attempts were modest—essentially, books were "decomposed" into portions suitable for step-by-step digestion, and the computer was the vehicle by which the knowledge was delivered. Later, when it was recognized that teaching (especially with computers) was a much more complex task than expected, a whole area of research grew up around these questions.

The complexity of the task became apparent when "intelligent tutoring" was introduced as the ultimate aim of the area.[1] Work on trying to endow the computer with intelligent behaviour has been named Artificial Intelligence (AI) as a contrast to natural intelligence reserved for people.[2] AI as a research field deals basically with three areas: representation of knowledge, natural language understanding and problem solving, all of which are equally important when developing the concept of intelligent tutoring. Many names have been used to describe the area which deals with the computer as an education tool; Computer-Assisted Instruction (CAI) turned into Intelligent Computer-Assisted Instruction (ICAI); the terms Computer-Based Instruction (CBI) or Computer Based Education (CBE) have been used in certain circles;[3] finally the notion of an Intelligent Tutoring System (ITS) implies using the computer in different roles—as a teacher, tutor, observer, etc. According to *Webster's New World Dictionary*, what distinguishes tutoring from teaching, instructing or educating is the notion of a one-to-one relation, i.e., a tutor is a

[1] There is, of course, no specific date when such a goal was introduced; nor is it the ultimate goal for computer scientists in general.

[2] The term 'Artificial Intelligence' was coined at a conference in Dartmouth in 1957.

[3] See, for example, Montague *et al.*, 1984, Freedman, 1984 and Rosenking, 1986.

private teacher. Thus the aim of intelligent tutoring must be to provide individualized, dedicated teaching.

Already, from the definitions in Webster's dictionary, we get indications as to necessary components of an ICAI tool: there has to be *knowledge* to be taught; teaching implies a *communication part*; if knowledge is to be provided, then a *student model* is needed to gauge student difficulties and progress; giving lessons means that a *teaching strategy* component has to decide the contents and sequencing of knowledge "pieces". Still, it is difficult to synthesize terms—what is meant by intelligent teaching? To quote Hartley (1973):

> "Elementary teaching intelligence . . . should adapt to individual differences by providing tasks and feedback which are performance sensitive, and they should be able to alter the teaching rules which are being applied if expectations of improvement in performances are not being met."
>
> "In the normal classroom setting, to say a teacher is intelligent is to imply that he uses effective methods for accomplishing his objectives."

One could claim that the key notion here is flexibility. The system needs to adapt the material and the presentation method to the individual student, and if necessary change its current strategy hopefully to a better one. Kearsley and Seidl (1985) define automation as a replacement of human functions by machine functions. If this is what we want to do with education, then the goal might not be attainable—a good teacher is so much more than a presentation device for knowledge bits. There is also a social issue, namely the claim that computerized education is antihumanistic.[4]

Let us instead settle for an intermediate solution—the computer as a powerful complement to existing educational materials: books, slides, films, tests, etc. As such, it could contribute very much. The aim would then be to individualize teaching, or provide personalized attention and interaction with the aid of the computer (Kearsley and Seidl, 1985). Thus the objective could be expressed as:

> ". . . automating instruction . . . (is) the ability to provide interactive, individualized learning experiences via machines." (Kearsley, 1985).

On the other hand, one can pose the question: are there reasons for having CAI at all? Two important factors are *necessity* and *quality of instruction*. There exists indeed a need for computer-based training: in the classroom setting it allows for greater flexibility for the teacher, letting him concentrate on certain students, complementing passive material such as books, and permitting active training in areas with practical applications. Teachers are a scarce resource, therefore computerized support would make better use of their talents (and maybe even extend them further!). In industry, banking, etc., there is a continuous need for education, and for keeping skills and knowledge up-to-date. Quality of instruction follows from the above-mentioned. Not only are there studies showing a significant reduction of learning times (Kearsley and Seidl, 1985) but there is also widespread agreement that learning improves with the involvement of more senses—doing is better than only listening or seeing.

1.1 System types

Assume that we accept the term Intelligent Computer-Assisted Instruction as the one characterizing all kinds of use of computers for teaching and training. There are then several dimensions along which systems could differ. To mention just a few examples: a program can be run on a dedicated machine, thus having access to all its power, or it may be an integrated subcomponent of a larger system; depending on the context, the program needs to take an active part in the instruction process, or to be a passive onlooker, interfering only in critical or incorrect situations; different requirements are set in a gaming environment where the student is actively pursuing a task compared to a domain with little possibility for practical training.

[4] I leave this discussion out—I presume it would be similar to the one carried out when the first books were printed.

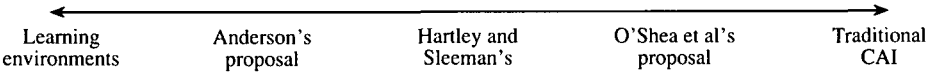

Figure 1 Yazdani's spectrum.

Yazdani (1986*b*) proposes a scale based on the key research efforts made by Anderson (1982, 1987), Hartley/Sleeman (1973) and O'Shea *et al.*, (1984)—this is convenient for the following discussion. The spectrum he offers has the appearance shown in Figure 1.

The chief dimension in this suggestion is initiative or active teaching ability—in learning environments, it is the student who has the initiative and performs actions which are then evaluated (by the student himself and the tutor); in traditional CAI, the tutor guides the learning process following some pedagogical goals.

Six basic categories of tutoring modes have been described by Bonar *et al.* (1986). These modes define the different levels of tutoring in terms of initiative, a student's level of knowledge and the balance between problem-solving and demonstration. One end of the spectrum is *exploration*, where the student looks for information in a microworld to form or confirm some hypothesis. *Experimentation* implies that the student tries to confirm or differentiate hypotheses which are recognized by the tutor. When such a hypothesis has been confirmed, the student can still *elaborate*, i.e., test it further on other cases. A *didactic* dialogue can be carried out, which means that the tutor presents problems to the student and drives the interaction. When the student is at a loss, the tutor can *demonstrate* concepts explicitly. Finally, during *coaching* the tutor goes one step further by providing hints as to how to solve a problem or to understand a concept.

Computer coaching has been discussed in detail by Burton and Brown (1979). The main task of the coach is to transform unguided, uninspiring game-playing into a constructive environment where the student's knowledge and/or skills are expanded and enhanced. The way to do this is by detecting errors in the student's reasoning, and pointing out correct ways of doing the actions. When observing actions which could be improved, the coach could indicate how this can be achieved. It can easily be seen that these tasks offer at least two problems to be solved. One is *timing*, i.e., when do we interrupt the student? This is not a trivial problem, as it is closely related to the topic of the preceding paragraph and the question of maintaining interest. The second problem is *content*, which is a question relevant to all ICAI tools. In order to put together a reasonable amount of knowledge, with the important distinction that it has to be relevant for a certain student in a specific situation, the coach needs to have some idea of what the student knows. This implies a diagnostic component, which in this context has one major restriction. As the student in involved in a (presumably) difficult activity, it would be unwise to interrupt the playing or problem-solving process at unnatural points of time. Thus the diagnostic modelling has to be implicit, based on the actions the student performs rather than on explicit dialogue with the coach. The technique used is called *differential modelling*, as it compares what the student is doing with what the expert module would do in the same situation, by evaluating the quality of the student's actions and determining what distinguishes the student's underlying skills from the expert's.

The question of cost is not unimportant—according to Kearsley and Seidl (1985), the cost and time of development increases drastically as we go from computer-based tests and drills, mainly using multiple-choice questions, keyword matching or screen position, to embedded training, tutorials with more complex answer analysis and feedback, to the most resource intensive type being simulations and games.

Between 5–500 hours of development time is necessary per hour of instruction for the above-mentioned categories. The amount depends among other things on whether the reasons for making mistakes are known, and whether the instruction already exists in some form. Another obvious factor that influences development time as well as the quality of instruction, is the designer's experience. Experience is many-faceted, but in this context we are talking about familiarity with the techniques, the student population, the system and the authoring language, if one is used.

In his survey of recent research in ICAI, Dede (1986) draws the distinction between an intelligent tutor and an intelligent coach. He states that

> ". . . the coach is a device designed for developing process skills in learning-by-doing situations, while the tutor provides a more instructor-centred approach geared to building a foundation of descriptive knowledge."

Based on this, we could already at this stage intuitively claim that a tutor must possess a higher degree of 'intelligence'. In fact, I propose the following scale where the requirement on intelligent behaviour increases: *warn – inform – guide – coach/train – teach/instruct – tutor*. The main reason being that the more to the right we move, the more do we imply a higher degree of initiative, flexibility and individualized attention.

1.2 Background for this study

In the vein of probing deeper into some of the above-mentioned issues, we are constructing a series of knowledge-based systems in the domain of biochemistry. The project focusses mainly on three goals:

- building tools to automate the acquisition of knowledge from experts;
- providing support for building generic knowledge systems, covering whole classes of applications or problem types;
- reusing knowledge bases for tutoring.

The tutor is being realized as a coaching system integrated with the planner (Ericsson and Sandahl, 1989), which based on a description of a biological sample suggests a series of steps to isolate a desired target protein. The solution depends on matching partial plans to pre-conditions which change dynamically as the steps are carried out. These partial plans are the core knowledge for the tutor as well as for the planner; they embody known solutions to a number of well-specified problems, but can also be in-house preferred methods, or even descriptions of failed experiments. Such knowledge is crucial to any organization; as is the ability to select among disparate pieces of knowledge to find those with bearing on the current problem. This kind of knowledge is what we want our tutor to teach, which should be facilitated by the fact that it matches closely the way the planner works.

Apart from wanting to validate the model we have chosen and to gain experience, we also investigate the following:

- how to make different instructional strategies explicit, and thus accommodate for various pedagogical styles;
- how well our knowledge representation supports the enhancement of the explanation process as partial plans allow for a significant amount of hindsight and prediction;
- how well our model is suited to the expert critiquing approach (Hagglund and Rankin, 1988).

Today, ITSs are created by craftsmen, using specialized tools towards the aim of producing unique systems. Yet, with the increasing co-operation between researchers from various disciplines, the possibility increases that sometime in the future we will have techniques accepted as being efficient enough to complement human teachers. This paper then, is written with the intention to map out and point to critical problems, on the road towards integrating intelligent tutors in knowledge-based systems.

1.3 Outline

In section 2, we describe the history of ITS; a little about early attempts at using the computer in education and the development which led up to the forming of the area Intelligent Tutoring Systems. Finally, we give a summary of the state-of-the-art by discussing some recent systems.

Whenever we review a system of importance a little more extensively, we do this under easily found section titles of the form "The *System* Experience".

Section 3 discusses several suggestions for ITSs design. We look into the modularization of ITS into components (representation of knowledge, student modelling, teaching strategies and interfaces), and scrutinize different attempts both at implementation and design levels.

The emphasis of the paper lies in Section 4, where we go into detail in areas which constitute major barriers, those which form important building blocks of ITS, or those which in the future could be found to be of use. Again, when systems through their design or implementation are good examples of a certain problem of idiosyncrasy, we describe them and thus put the issue(s) in context.

A summary of the paper together with conclusions is given in Section 5.

2 History

The use of computers for education is not a novel idea, although viewed globally its history is not very long (just like the history of computers themselves, of course). Those who summarize the development of the area such as Yazdani (1986*a*,*b*) usually divide the history into decades. This might be a simplification, but it is nevertheless convenient. We will give a short background of the techniques used before intelligent tutoring systems emerged, and go into more depth on those.

Computer assisted instruction (CAI) started in the 1950s with simple *linear programs*. A linear program consists of a series of steps which take the student closer to some desired goal, usually an increased amount of knowledge in some area. This is done by presenting information in large enough, meaningful "chunks", and collecting responses from the user. The pace is thus decided by the student, who on completing the session by answering a quiz or query is told whether (s)he was right.

In the 1960s a small but significant change was introduced. The idea was that if the student got to decide the order in which the material was presented, the learning process would be more effective, and, above all, individualized. The student would not have to work through the material sequentially, but rather select the unknown, useful parts.[5] One application can be found in problem-solving training through written simulations (McGuire *et al.*, 1972). The main features of these *branching programs* were thus that they offered corrective feedback, and adapted the selection of teaching material to the students. The evolution of the branching programs led to the creation of the so called *authoring languages*, which were meant to simplify the development of CAI material. There are basically three kinds of tools one can use for creating instructional programs. The first is to use a general-purpose programming language. This gives most flexibility, but also takes the longest time to develop. Authoring systems (Park *et al.*, 1987) are the other end of the spectrum—they provide a structured framework specialized for producing instructional material, which makes developing easier and quicker, but on the other hand puts constraints on what can be done. Authoring languages are supposed to provide the best of both—tailored to instruction, without the limitations that could intrude upon flexibility and efficiency.

The 1970s saw the birth of a new concept, namely the automatic generation of teaching material. In areas like mathematics, the computer itself could generate problems and their solutions. Given some, often implicit teaching strategy, the *generative program* could then compare the student's response with the correct answer, and offer comments. Unfortunately, these comments were not very insightful—as the programs did not possess any real knowledge of the domain, they could never answer questions like "WHY is this done?" or "HOW is it done?". One could say that the gap between the student's cognitive processes and the internal workings of the program was too wide. CAI never offered a serious 'threat' to human teachers,

[5] The concept of selecting the presentation or work-through order of instructional material was not a novel one. It existed and exists in paper form, and is called 'programmed instruction', where programmed refers to the sequencing order and not to a computer *per se*.

even if it did offer some degree of individualization and feedback. The lack of correspondence between the representation of knowledge in the computer and the representation of the student's mental processes initialized the research for better suited methods for representing knowledge.

Historically, we can distinguish between two uses of computers for education: conceptual learning through general-purpose computers, and decision-making as well as psycho-motor skills in simulators (Kearsley and Seidl, 1985). The trend is now that these two "branches" are converging, mainly in two ways. First, the use of simulation combined with the increased usefulness of personal computers has widened the areas of application. Second, active co-operation between engineers, human factors specialists, educational psychologists and technicians has started to evolve, and should contribute considerably.

2.1 Enter: Intelligent Tutoring Systems

Almost in parallel with the generative systems, the new concept—Intelligent Tutoring Systems[6]—was formed. There were mainly two reasons for introducing this new technique. First, there was the need to improve the teaching capabilities of generative systems. Second, it was a question of timing and new possibilities. The strongly emerging use of expert system technology had 'all of a sudden' provided new ways to represent reasoning chains and to modularize knowledge. At the same time, it was realized that the knowledge used for problem solving could also be re-used for teaching purposes (Clancey, 1982; Nordin, 1985).

Various attempts have been made to design and implement ITSs. It would be too voluminous to describe them all here, but we can at least mention some of them pointing to their main traits.

Research in the field of artificial intelligence had already for some time studied the different ways of representing knowledge in intelligent systems.[7] In SCHOLAR (Carbonell, 1970), which could be said to be the earliest ITS, facts about geography were represented by semantic nets. The knowledge in SOPHIE I (Brown *et al.*, 1982), an electronic circuit troubleshooter, was originally represented procedurally, without the intention of making these processes visible to the student. Later on, in SOPHIE III, a more sophisticated scheme (see Figure 2) was used, incorporating both rule-based representation as well as "local experts" for each device it modelled. The three

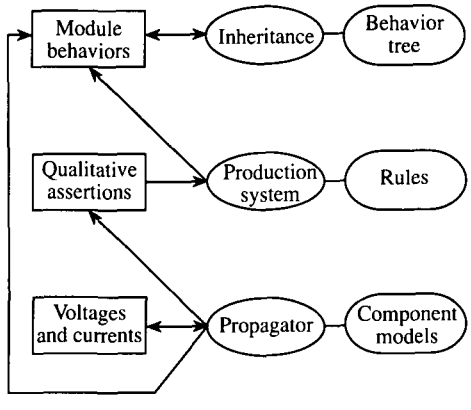

Figure 2 The electronics expert in SOPHIE III.

[6] Different names appear in the literature—A = ICAI (intelligent computer-assisted instruction), AICAI (artificial intelligence CAI) or ICBI (intelligent computer-based instruction). 'Intelligent Tutoring System' will be used as a generic term.

[7] We will continue to use the words 'intelligent' and 'intelligence', even though they are at times misleading or too ambivalent. Intelligent systems here are meant to be systems using AI-techniques, i.e., systems with problem solving or pattern recognition capabilities, natural language interfaces, or systems with explicit (and modularized) representation of knowledge. (A somewhat recursive definition, agreed, but one used by most of the AI community).

knowledge structures involved were a propagation database containing quantitative deductions about voltages and currents, qualitative assertions about the components and a behaviour-tree showing possible behavioural modes of each component. GUIDON (Clancey, 1982, 1983, 1987), a tutor built on top of the MYCIN expert system, has three levels of knowledge: strategic, structural and support. Together, these levels contribute to making explanation of the domain knowledge more coherent as different kinds of knowledge are represented. In the WHY system, script-like structures account for the knowledge, by defining casual and temporal relations between actors. (GUIDON and WHY are described in more detail in section 4.1).

Two products that have been commercially released are the PROUST tutor for debugging Pascal programs (Johnson and Soloway, 1987) and The LISP Tutor[8] (Anderson and Reiser, 1985), for teaching programming in LISP. It is doubtful whether they were better than other instructional programs; however, owing to the high interest in programming in general, and software costs in particular, there was (and is) a market for such products.

Another popular area for trying out teaching strategies and design issues is mathematics. The Geometry Tutor (Anderson *et al.*, 1985) uses the same ideas as the ones behind the LISP Tutor; BUGGY (Brown *et al.*, 1977) was a program for building procedural models of students' skills in arithmetic.

Games and simulations are naturally an interesting area, into which much effort is being invested. One of the first and best known systems was WEST (Burton and Brown, 1979), a coach for game playing. (Coaching was touched upon in section 1.1).

2.2 State of the art of ITS

Implementation-wise, the area of intelligent tutoring seems to have reached a plateau at around 1982 with the completion of SOPHIE. It appears that some kind of threshold has been reached from where it is no longer simple to continue without investing a major effort. Several systems created in the past few years (1982–1987) are described in the literature, but their novelty value is not as apparent as in the early works. What, then, have they contributed, and what are the main obstructions on the road to more effective tutoring systems?

2.2.1 The STEAMER Experience

STEAMER (Holland *et al.*, 1984) is a computer tutor which assists in propulsion engineering instruction. One of the underlying ideas for STEAMER was to create an interactive inspectable simulation, one that conceptually (not physically) resembles the real situation, which would help in understanding how mental models are developed. Propulsion engineering is a feasible area from several aspects, of which the foremost is that the cost of a high-fidelity simulator is around $7 million. The process is complex, and takes a long time to master.

The system's graphical interface is quite elaborate, and allows having many views, i.e., collections of status parameters of a propulsion plant. In real life these are mostly presented as gauges; in STEAMER, the designer has the possibility to combine parameters into graphs or curves, with indicators depicting magnitude and direction of change. The indicators serve as explanations of relationships not easily described in words. The authors call them *continuous explanations*, as they are supposed to more closely model the way experts think. The aims of the knowledge presentation were to allow several perspectives on the status of a plant and to have a flexible model of both the plant and the student.

The graphical interface, being quite complicated, requires a powerful editor which allows the definition of relationships between parts of the plant, and combines those into indicators. This is, however, still done manually—one would, of course, wish for a tutor that depending on the interaction automatically could decide which relationships to present. Almost nothing is said about how the system discusses objects, for example, even though it is claimed that it carries out such

[8] The LISP Tutor is described in more detail in 4.4.

discussions. The STEAMER effort, in summary, is little else than a programming environment for simulations—there is no teaching or tutoring in the normal sense, at least not yet.

2.2.2 The RBT Experience (Woolf *et al.*, 1988; Woolf and Murray, 1987)
The main emphasis in recent years seems to have gone into industrial applications of ICAL, like the teaching of industrial processes. One effort to build such a system for multiple explanations and tutoring facilities for a boiler process has been done by Woolf *et al.* (1986). Again, the resulting system was an interactive simulation extended with tools to help develop abstract models of complex processes. From the conclusions drawn from this project, the fundamental point which emerges is that a teaching philosophy needs to be clarified early in the development; a philosophy spanning over the range from recognizing actions (relevant as well as irrelevant), individualizing responses and monitoring activities without active dialogue.

The Recovery Boiler Tutor (RBT) is based on four modules: simulation, knowledge base, student model and instructional strategies. These components make it possible to represent knowledge either procedurally or declaratively, or both. For example, a situation may be described both by a mathematical formula and frame-like structure containing preconditions, actions and conditions for solution satisfaction to be used by the tutor. It appears necessary to have a formal representation to model the process as realistically as possible; on the other hand, users prefer being shown parameters in a simpler, but more efficient way. This is confirmed by the work on STEAMER as well.

The aim of the instructional strategy is to let the student explore freely, and intervene only when some goal is not approached. The tutor responds in three different ways: the student may be *redirected*, by pointing to some fact that has been overlooked; data may be *synthesized*, indicating a relationship between parameters; and feedback can be given to *confirm* correct actions. Otherwise, the user learns mostly by experimenting and watching the results of actions through trend analyses, animations and composite meters[9] of the simulation.

Accepting the fact that interactive simulations are quite pleasant and could offer good tutorial environments, it is important to recognize that by simplifying the total output (e.g., composite meters), the learner might receive a construed and artificial view of the process. One can also pose the question whether it is fair to the user to offer him/her a synthetic value, it being a translation from a mathematical formula. Such values could be difficult to create if there are several experts. Dialogue is not produced in natural language—the user input is carried out by menu selection, and responses are given in canned text form.

2.2.3 The MENDEL Experience (Streibel *et al.*, 1987).
MENDEL is an advice-giving computer system designed to be used in genetics education. The main goals of the project are to help students develop an understanding of genetics, as well as scientific research skills such as problem identification, hypothesis generation and testing. MENDEL is divided into a problem generator and an expert tutor.[10] Much work has been devoted to the interface and the interaction building on work by Anderson *et al.* (1987) on cognitive principles for computer tutoring (see section 3.2.2). The student's work with the tutor is claimed to be more realistic than when working with textbook problems; the student may create classes of problems, set genetic parameters and then experiment with the simulated genetic material trying to give support to or discard hypotheses. The tutoring part will be implemented so as to model a human tutor—being able to solve genetics problems, develop a student model, and decide on

[9] Composite meters show the state of the boiler using synthetic measures such as *safety*, *efficiency* and *reliability*, calculated through complex mathematical formulae based on several parameters. Such calculations are rarely (if ever) made by a user.

[10] When this is written, MENDEL is not a fully implemented system. In fact, only a prototype of the problem-solving component and a section in which students may create classes of problems if completed. The most important parts of an ITS such as a student modeller and an advisor (= teaching module) are still at the prototyping stage. For the sake of the discussion we will, however, consider the implications of the design.

whether to intervene and how. In addition, MENDEL is said to provide multiple windows into the reasoning of the expert tutor. This feature is important, as experience shows that different ways of presenting material is crucial to effective teaching (see section 4.1). In an advisory mode, the tutor will be able to offer hints as to what to think of, from general to more specific, or show what the expert would have done as a next step. Explanations may be given on two levels: strategic ones which clarify rules on a more general level; and support explanations which use the domain knowledge and examples to justify rules. A simple review option is described, commenting on whole problem solutions and pointing to undesirable actions. Student modelling is not implemented at all, but it is proposed that it should be able to handle rule-based and model-based reasoning. It is unclear what is meant by the latter, but it points to the conceptual knowledge underlying a rule.

The documentation on MENDEL does not describe any novelties. I am not convinced that the system provides "computer-generated advice", or whether it only is a tool for simulating genetics problems. There is no evidence that the comments generated by MENDEL really are individualized, or whether they are merely canned texts popping up at convenient moments. One of the more important parts of tutoring, deciding on when to intervene and how, is mentioned but left unexplored.

2.2.4 The OBIE-1:KNOBE Experience (Freedman, 1984; Freedman and Rosenking, 1986)
OBIE is an authoring system which enables non-programmers to develop interactive procedural simulations. It allows making connection between graphical objects and their abstract descriptions of actions and attributes. Editing produces representations similar to statements such as: "*If GND is up and TEST is down, then REM is off and NAV is on and BATT is off.*" The aim of the project was to make a more general system, one that could be used in several domains. The domains have in common that they have some kind of panel which is to be used by an operator.

In short, we can say that the OBIE-1:KNOBIE effort produced an authoring system, not a tutoring system.

2.2.5 The Bite-Sized Tutor (Bonar *et al.*, 1986; Shute and Bonar, 1986)
One of the objectives of the Bite-Sized Tutor was to modularize knowledge by using an object-oriented approach. Each "item" to be learned is represented by a class in an object hierarchy, which uses standard inheritance mechanisms. Student models and teaching strategies are also classes; this permits having several instances of a teaching strategy, making it possible to define different styles of teaching. Knowledge which is shared needs only to be represented once, thus providing a better-defined system.

A diagnosing component makes sure the student is using each piece of knowledge according to some pre-specified diagnosis-algorithm. Once it is done, a task selector part takes over to decide what to do next. This is determined by looking at the student model, which contains a record of events of the session and the student's performance on each piece of knowledge.

The authors admit that not much of the code is shared between different tutors, which was one of the aims of the project. The object-oriented approach has potential, as the hierarchical structure is a natural way to represent certain domains, and helps as well to modularize the tutor itself. An interesting idea to pursue is using an object-oriented approach for teaching strategies, not only for representing knowledge.

2.3 Summary of recent efforts

What conclusions can we then draw from the above-mentioned projects? Have they contributed anything substantial to the area of Intelligent Tutoring Systems, or have they only been reconfirmations of theories developed before the 1980s? I am bound to agree with the second view, with one important exception, namely in interactive simulations and the use of graphical techniques. In themselves, these two fields do not have much to do with intelligence; however, they

have opened up new ways to view and develop training facilities. As such, they promise great potential, and will be able to contribute more and more as hardware and software become more powerful and costs decrease. These projects I think confirm the hypothesis that major, collaborative efforts are needed to cover the scope of intelligent instruction and produce the desired systems.

3 Components of an ITS

In this section, we review some different approaches to describing the architecture of an Intelligent Tutoring System. It is noticeable that every author or designer has his own specific notion and terminology—there is no agreement, largely due to the (so far) experimental nature of the area. It is worthwhile to have a look at the different philosophies to see which parts are just renamed and which are novel, useful divisions or extensions.

We have divided the section into three parts: in the first, we present the various approaches to modularization of the ITS, i.e., the decomposition of the ITS into subcomponents. In the second part, we discuss some design principles, based on cognitive aspects of tutoring. We conclude with a discussion and comparison.

3.1 Different approaches to modularization

3.1.1 Hartley and Sleeman

In one of the earliest papers on intelligent, computer-based teaching, Hartley and Sleeman (1973) set up a goal which to this day remains as a major aim of research. They stated that:

> ". . . an interesting problem for computer-assisted instruction (CAI) and one which will affect its future potential is to devise teaching programs in which the [teaching] strategy itself is written as a set of (general) rules which the computer program has to apply in ways which lead to efficient teaching."

Two things are noteworthy in this statement. First, the explicitly stated need to represent teaching strategies which can be interpreted by a computer system. This implies that there is a need for research in educational theory as well as in machine-representable models of teaching. An implicit need then arises to devise systems within the framework in which it is possible to experiment with different strategies. Second, Hartley and Sleeman describe the goal as "*efficient teaching*", which is further described as the ability to adapt to individual differences by providing tasks and feedback which are performance sensitive, and the ability to change the teaching rules if the desired teaching objectives are not met.

Based on the idea of what teaching is all about, four educational components in the form of distinct knowledge bases are necessary: a *representation of the teaching task*; a *representation of the student*; a *set of teaching operations*; and *means-ends guidance rules*, which relate teaching decisions to conditions in the student model. The teaching task will ideally be represented so that teaching material and solutions can be generated automatically, allowing the teaching operations control such things as teaching style, type of feedback or curriculum management. A student history is saved, and all actions are evaluated by some measure of achievement.

Teaching intelligence, in some sense, is equated with adaptability, i.e., the capability of improving and enhancing the material, error detection, as well as changing strategy with respect to the competence of the student.

3.1.2 O'Shea et al. (1984)

As the proposal of O'Shea *et al* (1984) aimed at producing a design for an authoring system for designers without any programming skills, it is somewhat more extensive. In consists of five components with the following function (see also Figure 3):

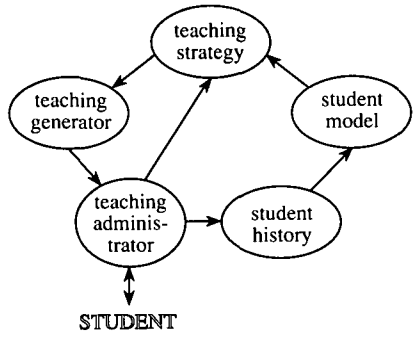

Figure 3 The O'Shea *et al.* (1984) five ring model.

1 A *student history*, which is a record of the material presented to the student and the student's responses.
2 The *student model* contains the current state of the student's knowledge and ability, and makes predictions about the future performance.
3 The *teaching strategy*, which relates the system's view of the student to the possible teaching operations, and decides on the one to choose.
4 A *teaching administrator*, which presents material to the student and processes the responses.
5 A *teaching generator* yielding a specific teaching action to be used by the teaching administrator.

The authors state that the model can be constrained to provide *all* existing computer tutor and CAI environments from games to problem-solving monitor.[11] As can be seen in the model, the student may request a particular teaching action through the administrator. This is, however, not always possible—it depends on what type of system is being used. Three types of systems are distinguished: those that can only present material, those that are also able to tutor the student, and finally those that also demonstrate mastery of the subject being taught. *Presenting material* is constrained so that the student can never affect the course of action, only indicate whether (s)he is done looking at some information or not. *Tutoring* is extended with interaction, i.e., the student may ask questions of the system, and the questions asked by the system depend on teaching goals given by the strategy. *Demonstrating mastery* means that the system must be able to use the knowledge it teaches, and to show the student how to use it as well. From the student's point of view, systems either evaluate their performance or not.

On this model, a set of Courseware Design Templates are created, to be used by instructors not proficient in programming to produce computer-based training material. The templates provide default values which facilitate the creation of teaching material.

3.1.3. Anderson

Anderson *et al.*, on the other hand, when building the Geometry Tutor (Anderson *et al.*, 1985) and the LISP Tutor (Anderson and Reiser, 1985*b*; Reiser *et al.*, 1985) (for a description, see also section 4.4) based their design on slightly different components. The underlying ACT-theory[12] is based on the consideration of the psychological processes that are used when acquiring cognitive

[11] The proposed scale is actually a hierarchy, with the template for *mixed initiative* on the top, having as subtypes *problem-solving monitor*, *quiz network* and *exposition*. Problem-solving monitors are divided into *game* and *modelling*. quizzes into *enquiry* and *drill-and-practice* and expositions into *slide show* and *exam*. It is not relevant here to make clear the differences—to do this, see O'Shea *et al*, 1984.

[12] ACT stands for Adaptive Control of Thought. A thorough overview of this theory is given in Anderson (1982).

skills. It is composed of two major stages: a *declarative stage*, under which facts about the domain are interpreted; and a *procedural stage* performing the skill. The theory is mainly concerned with how symbolic or cognitive skills are acquired, and how they are later compiled from the declarative to the procedural stage. Within this framework, Anderson proposes four components:

1 A *domain expert*, which is capable of solving the problem, Sometimes called the "ideal student" model.
2 A *bug catalogue*, which is a library of common misconceptions and errors in specific domain.
3 *Tutoring knowledge* in the form of strategies.
4 A *user interface*, which handles the interaction between the student and the tutor.

On a general level, the functions of these components are fairly straightforward. The domain expert should itself be able to solve the problem presented to the student. Knowledge about the domain is represented as production rules;[13] the same formalism is used for the known misconceptions.[14] The interface and the interaction in the LISP tutor are quite simple—when the student makes a mistake, the editor "wakes up", diagnoses the situation and provides a hint if the situation is one of the buggy ones (see point 2 above).

The ACT-theory evolved, and was presented in 1983 as the ACT* cognitive model of performance and learning (Anderson, 1983; Anderson *et al.*, 1987). With the model as foundation, Lewis *et al* (1987) presented the TEACHER'S APPRENTICE, a project based on the principles of the theory (see section 3.2.2), to design an ITS. The components are the same as the original ones, with two exceptions. The ideal model of the student now contains the common misconceptions (buggy production rules) as well as the correct ones. Two points are emphasized concerning the generation of rule sets for the student model:[15] first, the grain size of the rules is critical when stimulating students, and second, both strategic and axiomatic knowledge must be stimulated, preferably separately. An example of strategic knowledge could be to say that a mathematical expression should be distributed; axiomatic knowledge would be that if we want to perform addition, then we should use the operator '+'.

The second exception to the model is the addition of a rule induction component. This is supposed to relieve the instructor/implementor of entering all the rules into the system. Instead, the instructor/implementor simulates the parts of a teacher and an ideal student, and the system translates the surface behaviour of the interaction into a set of general tutoring rules within the context of a particular problem. Anderson claims that this reduces the amount of work required to create an intelligent tutor from months to days.

3.1.4 Ford

Assuming instruction to be a goal-directed process, Ford (1987) suggests a six-component model for an ITS. Goal-direction implies that a student at any point of time is in one of three types of states: initial, intermediate or goal. The possible states are states of knowledge which the student has attained in the teaching domain. The "state view" of the instruction process gives rise to several important questions. How do we characterize the initial, intermediate and goal states? Also, the current state has to be identified, and a next state has to be selected. One of the most critical issues is the question of how to get the student from one state to another.

These questions have to be resolved in order to achieve effective teaching. In solution to this, the following model is suggested (see Figure 4): a *strategy* suggests new states that the student should achieve based in a *learning theory*, which explains what (cognitively) takes place between successive observations of behaviour. The *instructional theory* activates a cognitive process which is

[13] If the concept of production rules or rule-based systems is unknown, a good introduction is given in Chapter 13 in Walters and Nielson (1988).

[14] The ACT theory does not claim that the productions exist in the student's mind, but rather that the behaviour of the student can be modelled with these rules.

[15] Remember that the term 'student model' is used by Anderson in a different way from the usual one. Here it is equivalent to a 'domain expert', or an *ideal* student model.

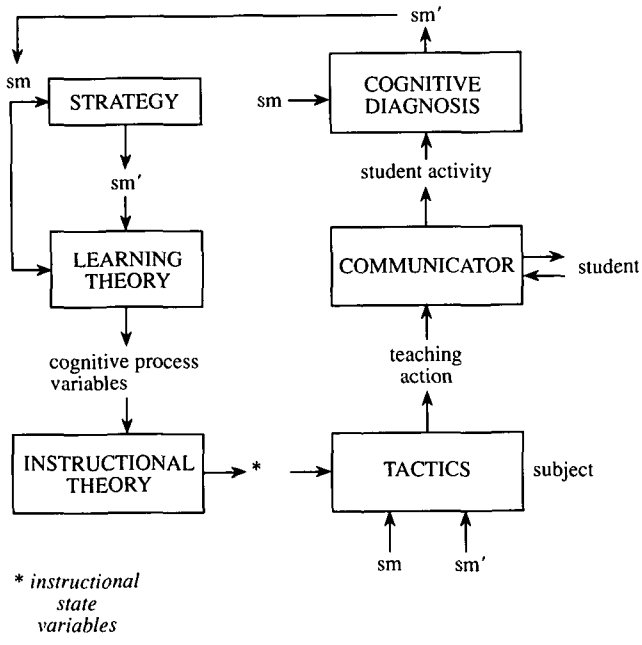

Figure 4 Ford's (1987) suggestion for an ICAI architecture (sm and sm' are different states of knowledge).

to carry out cognitive learning possibilities given as output from the learning theory. This is a domain-independent theory, covering such situation as constrained by the strategy and the learning model. Modelling of the student is performed by a *cognitive diagnosis*, which records all actions taken. The *tactics* component selects teaching actions based on its knowledge of the subject, and forwards them to a *communicator*, which decides on the appropriate presentation form.

The inputs to the learning theory are expressed in subject-dependent ways, which indicates that the theory should be a learning model for the particular subject.

3.2 Approaches to design

In many cases, work on computer-assisted instruction lacks a serious attempt to establish or understand the underlying notions of learning and teaching. Learning represents the cognitive processes which help us to assimilate new facts, build on previous knowledge, relate new areas to old ones or solve problems never before encountered. Aspects of teaching that need to be understood are how to choose what to say to a certain student in a specific situation. Some guidelines can be extracted from observing and analysing teaching or training situations in their right context; other principles have to be based on our knowledge of learning, what is best for the student.

Most system designs are created on a very weak theoretical foundation, usually a few *ad hoc* ideas. However, some attempts have been made to formulate coherent design strategies. We here review a few interesting suggestions, by listing the principles and discussing them.

3.2.1 O'Shea's design philosophy

O'Shea's design philosophy is described in general terms defining the functionality of the system. *Robustness*, *helpfulness*, *simplicity*, *power*, *flexibility*, *consistency*, and *transparency* are self-explicatory for any computer-based system. Apart from these, the following criteria are mentioned: all choices should be easily understandable and surveyable (*perspicuity*); the user should at all times know where (s)he is in the system (*navigability*); instructors with different perspectives should be able to create the same material, even though using different pathways to get there (*redundancy*); the system's response should be tailored to the needs of the specific user (*sensitivity*);

the system should take control when it knows sufficient about what the user needs (*omniscience*); and finally, the system must always be seen to be under the user's control (*docility*).

Most of these guidelines are general, commonsense ideas about how computer systems should behave and the user's role while using them. They are not specific to ITSs, but could be said to apply to any computer system. The ones that apply more specifically to CAI are *redundancy*, *sensitivity*, *omniscience* and *docility*. What they more specifically describe is the student modelling issue—all actions that the system performs should be 'need triggered', i.e., individualized and adapted to the student; and the 'mixed-initiative' aspect—both the user and the system should be able to take the initiative in a dialogue. Redundancy takes the student modelling one step further, but in the administrative direction—the possibility for instructors to work in the way they know best.

This is an example of a 'theory' which I believe is too general, because the above-mentioned concepts do not explicitly take into consideration cognitive aspects of learning. The form-based technique which helps instructors create diverse kinds of teaching situations also restricts them to exactly those situations (see footnote 11 in section 3.1.2) and their characteristics.

3.2.2 Anderson's design principles

Based in the ACT* model of cognition (see also section 3.1.3) Anderson puts forth the following design principles (Anderson *et al.*, 1987; Lewis *et al.*, 1987) (they underlie the design of the TEACHER'S APPRENTICE, a system built to explore the domain-independent parts of tutoring):

1 Represent the student as a set of productions.
2 Communicate the goal structure underlying problem solving.
3 Provide instruction in the problem-solving context.
4 Promote an abstract understanding of the problem-solving knowledge.
5 Minimize working memory load.
6 Provide immediate feedback on errors.
7 Adjust the grain size of instruction with learning.
8 Facilitate successive approximations to the target skill.

Anderson's model is somewhat specialized towards problem-solving, and adopts a very authoritative view of instruction. Instruction is seen on a rather low level, not taking into consideration the student's overall knowledge of the problem, but just the current step. As soon as an incorrect action is taken, it is pointed out and corrected. It is proposed that instruction is performed by tutoring rules (see Figure 5).

The rules seem to be very specific, and do not contain any real knowledge about the situation or the student. They contain information about the context to be used in, but only through direct pointers to domain rules which cause their invocation (*Production*). The tutoring rules themselves form an interaction sequence with options (*Active Set Transition*), guided by the activation of productions fired by the ideal student model.

Ohlsson's principles of intelligent tutoring

In the following, I will try to summarize the main points in Ohlsson's principles for intelligent

Name: T5
Production: P2
Action Type: Output
Action Body: Print "What is the result of distributing" num "over"term1 + term2 "?"
Active Set Transition: (T6)

Name: T6
Production: P4
Action Type: Input
Action Body: Make the student enter an algebraic expression.
Active Set Transition: if the student input is term1 + term2 ⇒ (T7), anything else ⇒ (T8)

Figure 5 Examples of tutoring rules from the TEACHER'S APPRENTICE project.

tutoring (Ohlsson, 1986). The summary will by necessity be brief compared to the excellent paper which present viewpoints from education research, psychology and computer science.

1 *The principle of pragmatic diagnosis*. The purpose of the diagnostic component of an intelligent tutoring system is to support the execution of its instructional plan.
2 *The principle of expectation testing*. The function of the diagnostic component of an intelligent tutoring system is to test whether the expectations presupposed by the tutor's current plan are consistent with its current model of the student.
3 *The principle of generative interfaces*. In order to provide adaptive instruction, a tutor must distinguish between the subject matter and the formats in which it can be presented, and be able to generate different presentations of each subject matter unit as needed, at each moment in time choosing the form which is most beneficial for the learner at that moment.
4 *The principle of non-equifinality of learning*. The state of knowing the subject matter does not correspond to a single, well-defined cognitive state. The target knowledge can always be represented in different ways, from different perspectives; hence, the process of acquiring the subject matter have many different, equally valid, end states.
5 *The principle of versatile output*. In order to provide adaptive instruction, a tutor must have a wide range of instructional actions to choose from.
6 *The principle of strategic repertoires*. The range of teaching tactics in a tutoring system is ultimately limited but the conditionality of the teaching strategy of the system; unless the strategy can identify circumstances under which a particular tactic is to be evoked, the tactic will not increase the power of the system.
7 *The principle of teaching plans*. A tutor needs to be able to generate a teaching plan on the basis of its representation of the student, its knowledge of the subject matter, and its current tutorial goal; furthermore, it should be able to revise its plan if it discovers that the plan does not suit the student.

Pragmatic diagnosis:[16] using the term 'instruction plan' implies that tutoring is a structured activity for which it is possible to develop and execute plans. This means we need to adopt a meta-view of tutoring, i.e., not only look at what can be taught in a particular situation, but whether it accords with the overall strategy. The instructional plans are linked to the student model through (implicit) expectations of what the student will do in response to the tutor's actions. If, for example, the student has to understand a principle to be able to apply it in another situation and does not, the whole plan has to be revised.

Expectation testing: to test expectations, we need to be able to run the student model and predict the outcome. If the prediction and the student's actual behaviour do not agree, both the student model and the tutor's plan have to be changed. Instead, if the teaching strategies contain implicit expectations about the student's behaviour, and these expectations diverge from the predictions of the model, then the plan needs to be revised. A new point of view, corresponding to changing the main question of modelling from "What goes on in the head of the student?" to "What does the tutor need to know in order to teach?"

This distinction seems to lead away from a cognitive approach to a more educational one. In a way, this argument can be compared to the 'glass' *versus* 'black box' dilemma (du Boulay *et al.*, 1981)—not all processes have to be accounted for, nor are they always interesting.

Generative interfaces: this may be the hardest requirement, and the most general one in the area of knowledge,[17] and also intuitively the most useful for teaching—as different people sometimes need different explanations of the same problem, then the natural thing is to have the ability to generate *different presentations of the same, underlying representation*. Separation of content from form is the main advantage of computer-based systems compared to other teaching materials.

[16] I will refer to the principles by the later part of their names.
[17] This is a term adopted by computer scientists—other names could be: *subject matter analysis* in the educational world; *task analysis* among psychologists; and *didactic analysis* within mathematics.

Non-equifinality of learning: this is basically a critique of the overlay models, or rather another warning. If there is only one representation of the knowledge, and the student model is of a subset of it, then we are bound to be restrictive on what constitutes an 'end state' of learning. One solution is to have multiple representations of the same domain knowledge; in rule-based systems, each representation could consist of one set of rules.

Teaching plans: this principle is actually a hypothesis about how an ITS should be designed. Ohlsson states seven requirements: Assuming teaching to be goal driven, a tutor must be able to *decompose goals into subgoals*. It must have *a description of the student* and of the *subject matter*, and *generate a plan for how to satisfy the tutoring goal*. Plans need to be *executed*, and if the tutor *detects a mismatch between the student and the plan*, then this *plan needs to be revised*. Plans thus need to be generated, executed and revised, and tutoring is a cycle going through these steps (more on planning, see section 4.3).

3.3 Design issues—a discussion

Above we have discussed several approaches to describing the design of an ITS through its components as well as the ideas underlying their design. At first glance, all modularizations seem relevant and instrumental. Let us look at them and see if there are any incompatibilities.

"What's in an interface?"

Obviously, no training or instruction can take place without an interface. The substance to be taught has to be communicated—as we saw in the Introduction, the terms 'teach', 'train' and 'instruct' are defined using phrases like 'to show how', 'to give to', 'to provide with', etc. Not only does the ITS need the capability to communicate, but also to be able to communicate the same matter in different ways, depending on whom it addresses, and in what situation. Anderson's model does address the issue of interface; it is, however, very simple, and reacts only when a mistake has been detected and is to be corrected. Ford recognizes the need for a 'communicator', but does not provide its functionality. All that can be seen is that the communicator takes a teaching action, forwards some kind of surface manifestation to the student and relays the responding student action to the cognitive diagnosis module. It is unclear whether the communicator has any qualification tasks or not. O'Shea *et al.* (1984) use the notion 'teaching administrator' in a similar way.

"What's in a teaching strategy?"

All models agree that there is a need for a separate module handling teaching strategies (tutoring knowledge, teaching operations). What then characterizes a strategy? Well, being the goal directed proponent, Ford stated a teaching strategy should point to a new state the student has to achieve. Unfortunately, he does not say what constitutes a state, other than that it is not a complete model of the student, but a 'slice' of knowledge. On the other end of the spectrum, Anderson appears to claim that teaching is not a goal-directed process; instead he proposes an induction component to extract tutoring rules from the surface behaviour of either a teacher or ideal student. It may be that this is an administrative improvement, but its qualitative results would have to be inferior. Teaching requires deep knowledge about the student, the subject matter and the situation at hand—it is hard to believe an inductive component could grasp that.

Generally, it seems that the role of the teaching strategies is to select appropriate presentation and testing methods, and send them off to the interface (communicator, administrator) where they become concrete operations. One of the simpler strategies is to compare what the student knows with what there is to know, and decided on where to continue from there by selecting a missing part. Appropriate methods imply heavy dependence on and close co-operation with the student model (cognitive diagnosis). Before presenting information, a relevant part of the subject matter has to be selected. This task is more loosely attached to the student model, in that some priority list may be preconceived. Independently of this, of course, the goal must be to fill in incomplete

knowledge. Testing can be done by introducing faults in a simulation mode, with the restriction that it has to be within the knowledge that the student has already acquired.

The strategy is also to account for the type of interaction. Here, the work of O'Shea *et al.* (1984) is at least an incipient structuring of the area. Just as Carbonell (1970), they claim the most general type of interaction to be the one providing mixed-initiative teaching. It is therefore naturally the most demanding. A hierarchy (see section 3.1.2) is not necessarily the only way to structure the area, but it is at least a beginning to recognize differences and requirements. The latter could be a need for a game-playing program, a modelling package; different dimensions could be control (student or tutor) or interaction (O'Shea *et al.*, 1984).

3.3.3 "What's in a student?"

It is commonly agreed that without knowledge about the student, teaching can never be efficient. The student is normally represented through a history of interactions performed, and some kind of model of assessed knowledge. This can be a synthesized measure (Hartley/Sleeman) or a state, being a collection of the facts (pieces of knowledge) known to the student (O'Shea). O'Shea also claims that a good student model should be able to make predictions about future actions; this function perhaps belongs in the teaching strategy component.

Ford carries the concept one step further. He also calls his model cognitive diagnosis, as it tries to explain what cognitively happens between observations of student behaviour. Thus a learning theory is needed, which together with the cognitive diagnosis forms input to the instructional theory. This theory returns a cognitive process variable which should try to achieve some learning possibility. Unfortunately, not much is said about the contents of these theories, nor their implementation. The latter could presumably be done as a production rule system. It is, however, crucial to have a coherent theory relating (a) cognitive states to (b) learning possibilities to (c) cognitive process variables. Without this causal chain, rules cannot be formulated.

Anderson's proposal is the only one which deviates somewhat from the "norm". From the start, what accounted for the student model was a catalogue of common misconceptions in a specific domain. Later, it was combined with an ideal student solution; a problem-solving situation was always compared to the way an advanced student would solve it. Hopefully, a student's intentions are thus traced, allowing direct correction when a buggy rule matches the current situation. When there is no buggy rule, the system can only help by showing the correct solution; it cannot give an explanation based on the student's previously acquired knowledge, lack of knowledge etc. This means that there is no adaptation to the current cognitive state of the student— all "teaching" is done on a one-rule level. This seems to be a serious drawback in most domains.

3.3.4 "Wherein lies the intelligence?"

After looking at these suggestions for design, the above question seems to be justified. First of all, none of the components can be said to have intelligent qualities. None of the components has the ability to respond successfully to a new situation or to learn from experience, which could be one way to define intelligence—nor do they aspire to. The second question would be if the components together exhibit intelligent behaviour. Again, if the above-mentioned definition is accepted, the answer is no, even though ongoing research on machine learning (see section 4.6) strives towards that goal. However, *efficient* teaching could be defined as a successful, step-by-step procedure by which the student is made to approach a certain educational goal. A lot of work has to be put into synthesizing educational and psychological knowledge into what constitutes a definition of educational goals and subgoals, teaching tactics and their application in recognizable situations, and cognitive states related to surface observation of behaviour. At that stage, the composition of an operational theory would be feasible.

3.4 Closing remarks

This whole section is centred around the components that make up an ITS. Remember however

that this is actually only one way of looking at things. Another way, suggested by Bonar *et al.* (1986) is to build object-oriented systems around the pieces of knowledge[18] that make up the domain to be taught. This approach (according to Bonar *et al.*) would avoid some of the general problems appearing in all implementations such as repetition of information, spreading out information that should be kept together and the supposed diffusion of code which should be modularized. In an object-oriented system, information can be shared by using inheritance mechanisms. Different styles of teaching can be incorporated through instances of a tutoring component, being a generic class in the system. As these ideas are still not completely implemented it is hard to determine their validity; it is impossible to anticipate how well we can meet the requirements set forth by the general problems.

4 Key issues in tutoring systems

4.1 Knowledge presentation

The representation of knowledge in a system is crucial to its performance. Stevens *et al.* (1982) argue that the representation of domain-specific knowledge plays a central role in knowledge-based systems. In the context of intelligent tutoring systems, not only does it determine the content of the tutorial interaction, but it also forms the goal structure that governs the selection of examples, questions and statements. In addition, it influences the types of misconceptions (or bugs, as they are also called). A common problem with knowledge presentation is that it is too shallow, i.e., it can be used to solve problems, but not to explain or justify the solution,

We divide the discussion of knowledge representation into two parts, both viewing the problem from the standpoint of a system. The first one is the WHY system (by Stevens *et al.*, 1982) showing on the importance of several ways of viewing the same knowledge; the second is GUIDON (by Clancy, 1982, 1987), mainly emphasizing the need to put knowledge in its right context.

4.1.1. The WHY Experience

The WHY system (Stevens, *et al.*, 1982) was part of the result of trying to build an intelligent computer-aided instructional system to teach students about physical processes. The basic assumption was that "real", human dialogue in a relevant area was the goal; such dialogues were studied, and modelling of them was attempted. The domain chosen was 'the causes of rainfall'.

The model which was developed had a script-like form (see example in Figure 6), with the following characteristics.

- The basis is formed by generic knowledge structures representing information about classes of phenomena.

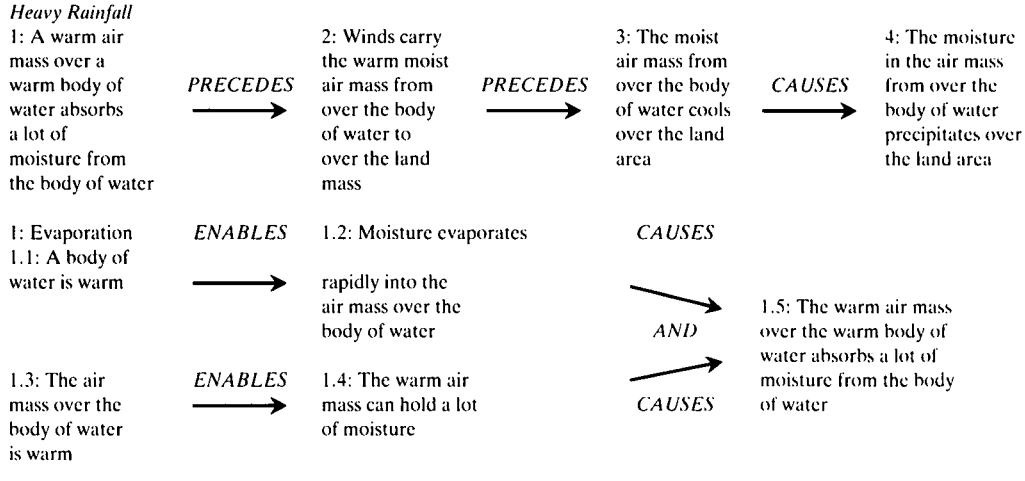

Figure 6 A script from the WHY system.

[18] These pieces of knowledge are called 'bites'.

- Sequences of events are partially ordered, linked by casual or temporal connections.
- The structure is hierarchically embedded, i.e., some parts can be expanded by subparts.
- A set of roles can be instantiated to particular entries when applied to a specific case.

The utility of this representation was seen immediately—the resulting system can ask questions about the domain, and even inform the student about incorrect steps. A main drawback, however, was that it totally missed the underlying misconception, the causes of errors on a deeper, semantic level. It was also noticed that many times it was impossible to describe a certain aspect of the domain; WHY is not general enough to cover all aspects and ways of describing physical processes. This was analysed, and considered to stem from the lack of multiple representational viewpoints, i.e., different ways to view the same piece of knowledge. The system handles *temporal orderings* and *casual relations*, but not, for example, *functional relationships*. In order to remedy this, Stevens *et al.*, (1982) attempted an extension of the model.

The functional relationships consist of four parts. *Actors* have roles in the physical process and are in some way related to each other. *Factors* are the attributes of actors (like temperature or humidity). The *result* is a change in the value of some factor. Finally, there is a *relationship* which holds between the factors and the result. These structures are script-like, but there are differences: casual relations are implicit rather than explicit, and they are non-linear and interactive rather than ordered and sequential. As was noted before, the difference is not qualitative, but makes possible another view of the knowledge. The approaches emphasize different viewpoints, and are thus useful for tasks of different types.

In their results Stevens *et al.* (1982) indicate that there is a need to map surface errors onto a deep-level representation, and to do this by representing common bugs explicitly. That is, of course, one way—the question is whether it is possible, at least in some situations, to find means of doing this procedurally.

Patterns often show up in errors. Stevens *et al.* found that in several cases they are due to the use of an incorrect metaphor (instead of viewing air mass as a 'container', students might view it as a 'sponge'), which has to be discovered. Another common misconception is that the student does not realize the relation between processes (as, for example, 'evaporation' being the inverse of 'condensation'). The authors do not propose a solution to these problems, but point to them as shortcomings of the current model.

Multiple viewpoints provide several ways of regarding the same knowledge. One problem that arises is obviously how to decide which viewpoint to choose. Another problem is how to detect interacting misconceptions (several bugs causing one surface error). No answers are evident to these problems. Apart from the viewpoints mentioned above (temporal ordering, causal and functional relationships) there are many others: energy, change-of-state, spatial viewpoints. It is important that possible viewpoints are investigated; misconceptions on the part of the student could also suggest a lack of insight.

4.1.2 The GUIDON Experience

GUIDON is a program for teaching diagnostic problem-solving in the area of medicine, developed by Clancey (1982, 1987). It is based on the expert system MYCIN, which is a consultation program handling infectious diseases. MYCIN was extended with approximately 200 tutorial rules used for guiding the dialogue, presenting and discussing rules, constructing a student model and responding to student actions. As the tutorial rules are modularized, the domain knowledge in MYCIN can be replaced by that of other problem areas.

Evaluation of MYCIN had shown that its competence was at the same level as the teachers' at the infectious-disease faculty at the Stanford University School of Medicine (where it was developed) in selecting antimicrobial therapy for meningitis and for bacteremia. The question that initiated the GUIDON project was whether the representation of knowledge, already proven to be adequate for problem-solving, would be applicable to teaching as well. A second issue was to consider what kind of knowledge could be added to MYCIN in order to make it an effective tutor.

Clancey and his colleagues soon discovered that the rules in MYCIN, although adequate for problem-solving, were not transparent enough to be used for teaching. A large proportion of knowledge was implicit in the rules, thus making it impossible for the program to provide acceptable explanations of the reasoning. Therefore, the concept of what constitutes a rule had to be revisited. The "traditional" performance knowledge of a rule was embedded in two other levels: *meta-level abstraction* and *support* (see Figure 7).

Patterns in the domain knowledge are represented at the abstraction level, through schemata and rule models. Facts are grouped together to form a common base upon which several rules are built. The support level contains a canned-text justification for the rule, as well as references to literature and the author of the rule. This extension of the knowledge representation allows several uses for the rules:

- for determining the subgoals needed before the rule can be applied;
- for finding (during a tutorial interaction) what else needs to be done before a rule can be applied, and select an appropriate method for presenting it;
- for generating questions;
- for understanding a student's hypothesis;
- for summarizing arguments by extracting parts of rules.

As one of the major goals of GUIDON was to experiment with teaching strategies, the ability to use domain knowledge in multiple ways is considered essential. Teaching strategies are implemented as tutoring rules, which can select and present appropriate knowledge flexibly. To summarize, Clancey (1982, 1987) has shown that production rules, if being the only source of knowledge, are not sufficient to be used in a teaching situation. They lack or do not make explicit the justifications for decisions, and often omit steps which would be of importance to students.

4.2 Student modelling

A student model is, maybe, the most important component of a system which is to exhibit behaviour often referred to as "individualized", "intelligent", "tailored to the user", etc. What

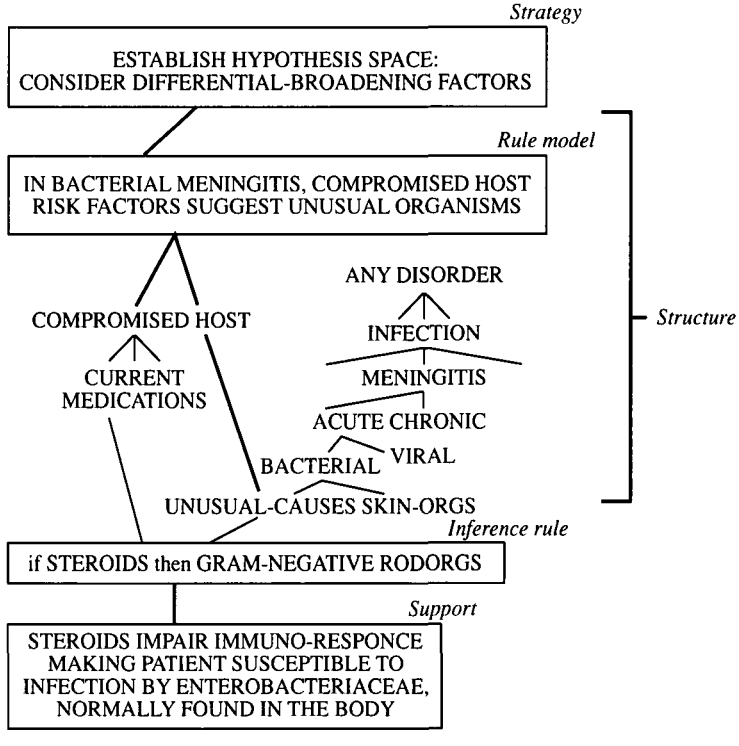

Figure 7 Extended knowledge representation in GUIDON.

these descriptions imply is actually not very strange—as a matter of fact, it is something we all do when we are asked to explain or describe a situation, fact or event. We automatically take into consideration who we are talking to—the person's background and his/her competence—while at the same time trying to figure out how we best can formulate our response.

The most simple view of a student model is to look at it in terms of a "best solution", or an expert model. If we consider the expert model as the goal (in some sense) for the student to achieve, then the student's knowledge at any point in time can be expressed as a subset of the expert model. This *overlay model*[19] shows which pieces of knowledge have been acquired and verified by the student. Here appears one major problem of student modelling, and that is the question of whether the student really knows a skill after only having demonstrated it in an isolated context. The opposite question is also justified: is it always true that some piece of knowledge is absent just because it is not used in a certain situation? Just as Burton and Brown (1979) note: the set of moves not taken indicates that *potentially* the student lacks some knowledge about each move. They formulate two basic problems:

- If there are several way to derive a move, we cannot know which one the student used.
- Which if the underlying issues necessary for a move is "responsible" for the student not using it?

Apart from overlay models, which specify which part of the subject matter the student has mastered, there are also *performance measures*, which indicate the proportion of the subject matter; *error descriptions* are the distorted or misconceived 'knowledge units'; and *simulations*, which are executable and allow the making of predictions about student performance (Ohlsson, 1986). Ohlsson calls student modelling 'cognitive diagnosis' owing to its more general nature, and states furthermore that:

> ". . . any commercially available education software which, as a rule, does not have any diagnostic capabilities, will not, in fact, produce more learning that traditional teaching materials."

4.2.1 Dimensions of student models

The knowledge about a student which forms a student model can be discussed along several dimensions. Rich (1983*a*) distinguishes three.[20]

- models on one single, *canonical* user *versus* a collection of models of *individual* users;
- *explicit* models given by the system designers or students themselves *versus implicit* models inferred by the system;
- models exhibiting *long-term* characteristics such as areas of interest or expertise *versus short-term* characteristics such as solving the problem at hand,

A typical example of a model built around a canonical student could be when the student is asked to rate himself as being either a novice, proficient user or expert in a certain domain. The system then adapts its functionality according to these stereotypes, along some preconceived guidelines. Individual models have the advantage that they can provide an interface more relevant to the student's needs than a canonical model. However, individualization implies that a model has to be constructed in more detail, and it has to be determined how various factors influence the performance of the system. This leads on to the next dimension. When explicitly asking the student questions the probability of the model being correct increases, while we at the same time want to avoid unnecessary and excessive interaction. Implicit techniques allow monitoring student behaviour and building a model based on those observations. Naturally, some information will contradict other information, and the system designer should be aware of that. Short-term models have to be very good at detecting changes in the situation, while long-term characteristics form some kind of

[19] The term 'overlay model' is due to Goldstein and Carr (1977). It not only states that the student's knowledge is a subset of the expert program's, but also emphasizes that it derived its structure from the structure of the underlying expert system.

[20] Rich uses the term 'user' in her paper, but I will here consider it as being equivalent to 'student'.

base of permanent information about the student. An example of an application where short-term models are important is in natural language understanding,

In Rich's opinion (1983*a*), the model to be preferred is one that is individualized and implicit, giving a more user-friendly system. As to the third dimension, a system needs the whole range from short-term to long-term information to be as flexible as possible. Rich suggests however that the methods for inferring the two types may be different, thus justifying the differentiation.

4.2.2 Basis of student models

Four different sources of evidence are said to support the student model (Barr and Feigenbuam, 1982:

1 *Implicit*, by monitoring the student's problem-solving behaviour;
2 *Explicit*, extracted from dialogue with the student;
3 *Historical*, i.e., assumptions based in the student's experience; and
4 *Structural*, reflecting the inherent difficulty of the material

Let us look at GUIDON (Clancey, 1982), which uses special tutoring rules to update the student model as soon as a domain rule has been applied. It tries to determine whether the student has reached the same conclusion as the expert module, and stores that information in the student model. The decision that the program has to make is based on four factors: (a) the inherent complexity of the domain-rule (one that is a simple definition or one that involves iterations); (b) whether the tutor believes that the student knows how to achieve the subgoals of the rule; (c) the student's background; and (d) evidence from previous interactions. Consequently, three properties are associated with each domain rule during a tutorial session. *Use-history* is saved between sessions, and represents the tutor's belief that the student knows the rule. *Sapplied?* is the tutor's belief that the student has applied the rule during the session, i.e., (s)he is likely to use its conclusions in a hypothesis. Finally, *Used?* is the tutor's belief that the student has used the rule to form a hypothesis, so when asked to support it, (s)he would refer to this rule. These factors distinguish between having used a rule earlier in time, being able to use it and actually using it. This is, of course, an important discrimination,

4.2.3 Techniques for building models

How do we actually go about constructing a model? What are the inputs and, more important, what could the inputs be? Which techniques can be used to draw conclusions about the user? These are a few questions we touch upon in this section. Following Rich's example, we divide the techniques into two groups: methods for inferring single facts and methods for inferring clusters of facts.

One of the aims of student modelling could be expressed as not "disturbing" the interaction, but nevertheless, manage to evaluate it and extract information about the student. Such techniques are called *automatic protocol analysis* **or** *silent modelling*. When using a computer system, if a user begins a session by forming a few commands that are rejected, it is likely that the user is a novice who needs some help. A modelling technique would be to associate with each command information about what the use of that command discloses about the user. It seems likely that this method could be extrapolated for use in student modelling, by substituting command for 'concept'. Associated with each concept there would be information about for example if the student has used it, which other facts build upon it, necessary knowledge for using it, etc.

Another method, or rather heuristic, is to look at patterns in the user's actions. If, for example, the user reformulates a request, it means that (s)he did not get the sought-for information the first time.

Inferring clusters of facts presupposes that by recognizing one or more traits we can map them onto a stereotype which is characterized by either all or some of these traits. A stereotype in this sense is just a set of traits that (often) occur together. Using stereotypes, if successful, provides the student model with more information about the student at one time. Some of the traits might be more easily observable that others. Rich calls them *triggers*, because when they are observed they

activate the whole stereotype. The triggers can be associated with ratings assessing how high the probability is that the stereotype in question should be activated. These are naturally only *ad hoc* methods for deciding these ratings.

Stereotypes can also be used in a hierarchical sense, i.e., some being more specific than others. At the top we would have a canonical user, under it more specific types. Thus we could extract a stereotype at the most detailed level, as well as the most probable one.

There are several problems with most student modelling systems. One is the difficulty of detecting and resolving conflicts between inferences, another is choosing between several stereotypes which are triggered simultaneously. Yet another problem is justification. When several stereotypes predict the same value, the level of confidence is higher than if only one does. Similarly, if one stereotype predicts one value, but the student model already contains another, then it is important to know where that value came from.

4.3 Planning

Planning is undoubtedly one of the central issues of intelligent tutoring. As Ohlsson (1986) pointed out, the whole tutoring process can be seen as teaching plan generation, execution and revision (see also section 3.2.3). We look at plans from the tutoring point of view, and discuss planning techniques, plan recognition and plan matching.

4.3.1 Problem formulation and plan representation

Planning techniques have a great potential for implementing teaching strategies. This presumes there is a global plan, built around an individual student, which selects topics during a teaching session. A requirement on the plan structure, stated by Peachey and McCalla (1986), is that it should be able to represent unordered sequences of steps and alternative strategies. When a plan fails, this allows for revising the part of the plan not yet executed.

Having a teaching plan presupposes we have some means of representing the problem in terms of what is to be learnt, what concepts there are, and what the possible misconceptions could be. Once we have the subject matter, we need teaching operators which actually try to teach a concept to the student. Peachey and McCalla suggest a STRIPS-like (Fikes and Nilsson, 1971) formalism, where the operator has the following parts: a name, the conditions (in terms of concepts or misconceptions) under which the operator can be used, the effects of the operator (add and delete lists applied on the student model) , and an action which tries to teach the student a concept.

One problem here is that the effects of an operator on a particular student are not predictable. It is therefore necessary for the planner to be able to use alternative paths and to revise a failed plan. A plan's goals are to be represented by expressions consisting of the logical connectives AND (&) OR (|) and NOT (~), and the predicate SK (which stands for 'student knows'). For example, the goal *SK(cpt1) & SK(cpt2)* states that the student knows concepts *cpt1* and *cpt2*. Misconceptions are represented in the same way. The plans themselves are represented as directed acyclic graphs (DAGs), where the nodes are either instances of operators or goals (see Figure 8).

The main problem with planning in tutoring systems is that they deal with something not easily observable, namely the state of the student's knowledge in a certain domain. Instead, that state has to be guessed at, from studying the behaviour of the student, interacting with him/her, taking into consideration the inherent complexity of the material, etc., (more on student modelling, see section 4.2). According to Peachey and McCalla (1986), planning in an ITS consists of two functions: *generating* a plan adapted to a particular student, and *executing* the plan, taking into consideration that situations may arise that could cause the plan to be revised.

4.3.2 Plan execution

A brief description of a plan executor would be that it uses a generated plan to invoke teaching operators, and if the plan fails, the executor should be able to pass it back to the planning component for a revision. The main features of Peachey and McCalla's algorithm can be viewed

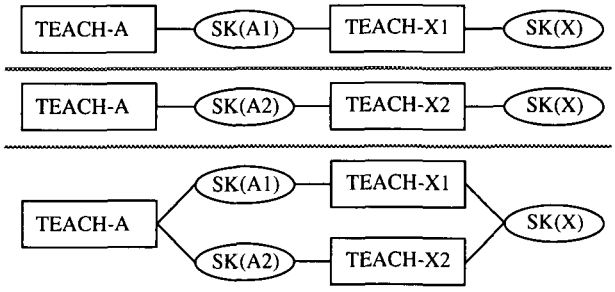

Figure 8 An example of a plan (Peachey and McCalla, 1986).

through the two data structures they use. One is the list of ELIGIBLE plan steps, i.e., those whose preconditions have been met, and they can thus be carried out. The second is the PRE list, which contains the precondition goal nodes for each plan step. When a precondition is met, it is taken out of the PRE list, so when the list is empty for a certain step, it may be executed. As planning is performed in a dynamic environment and some conditions could change at the last moment, a last check of the preconditions is made just as the teaching operator is invoked. Revision of the unexecuted part of the plan takes place each time a step has been carried out (for more details, see Peachey *et al.*, 1986, pp 85–86).

No AI methods are used to guide the selection of subsequent steps to be executed. Any eligible step is chosen any time, as they are each known to strive for the successful completion of the plan. Several techniques are possible, such as depth-first or breadth-first search,[21] and their choice determines the style of interaction. Here, I believe, there is room for improvement of the model of Peachey and McCalla (1986) by introducing some means of representing teaching strategies which would set the parameters for the dialogue, thus further constraining the planning process.

Generally, a plan is created from scratch bearing upon some ulterior goal. In the planner we discuss above, planning is always based on a partial plan, passed to it by the plan execution component. Plan formulation consists of two stages; a naïve stage, under which a tentative plan is formed, and an editing stage, when the naïve plan is restricted. The tentative plan does not take into consideration that goal may interact, thus having to be executed in parallel. The plan is developed from the goal backwards (right to left), but is of course executed from left to right. The editors task is then to scrutinize the plan, looking for situation where goals do interact, and introducing further constraints to handle the ordering of goals.[22] It is stated that the editor is rarely used in the system, due to the fact that learning is a cumulative process, thus inhibiting interaction between subgoals.

Peachey and McCalla (1986) recognize five different types of behaviour of the execution component which could be useful to keep in mind during design. They are:

1 *Normal execution*, where the plan proceeds as expected.
2 Use of an *alternative branch* due to failure of an operator.
3 *Bypassing steps* in the initial plan, which are deemed unnecessary because the student learns spontaneously.
4 Plan failure and replanning, *reapplying operators* already used.
5 Plan failure and replanning, *using remedial operators* to correct misconceptions.

4.3.3 Plan recognition

Another way to view plans in a tutoring system is as a means to understand how the student goes about trying to solve the problem at hand. More specifically, plans could be used to explain how a

[21] An introductory and very pedagogic text about problem-solving methods and search strategies is given in Rich (1983*b*, Chapters 2 and 3).

[22] The authors compare this to the NOAH system (Sacerdoti, 1977; Rich, 1983*b*), where *plan critics* examine the plan, each making some specific observation about it and, if necessary, proposing some modification.

set of actions achieves a certain goal in terms of the student's knowledge about the subject. Genesareth (1982) has designed and implemented *plan recognition routines* that reconstruct such plans directly from sequences of actions. Below we will give an account of this work, and discuss its usefulness.

To provide effective instruction, a system needs to know about the student's knowledge of the subject matter and the misconceptions which obscure this knowledge. Tests can be used to deduce misconceptions from student answers, but it would be beneficial of there was some way to find out what beliefs led the student to that final answer. The objective leads to consider ways of automatically analyzing students' steps to find and interpret the underlying reasons for choosing those steps. In simple mathematics, there is usually a fixed solution sequence which the student only executes, and deviations can be found by comparing the student's steps with those of the correct algorithm. More complex areas require a sequence of steps which could be combined in several ways, making it difficult to foresee and especially explain why a certain combination was chosen. The need arises for some way of understanding what initiates the choice of a step, and what governs its composition.

4.3.4 Research issues in planning

Planning in Intelligent Tutoring Systems is more complex than traditional planning, as is, for example, robotics or micro-world problems, because the evidence of success and failure are more difficult to recognize. In addition, to find such evidence usually requires specific actions on the part of the planner. The area being complex does not, however, exclude the possibility for improvement.

One area of research with Peachey and McCalla (1986) point to is retaining successful plans and reusing them in situations similar to the one they were created in. In this case we need some matching capabilities to compare situations with respect to particular characteristics. The saved plans could also be used for similar students, requiring some mechanism for comparing students as well.

Another problem is how to recognize at what stage in the execution of a plan it is most efficient to abandon and replan, for example in situations where a dialogue is interrupted. The technique used by Peachey and McCalla (above) continues until circumstances make it impossible to go on. Variations are also possible where a revision is initiated as soon as a misconception is detected, or when all the effects of some action are not obviously reached. Replanning can, of course, be avoided to some extent, if it is possible to anticipate the effects of alternative paths. If, on the other hand, replanning has to be done, it can be greatly facilitated by having well thought-out communication and feedback between planning and execution.

4.4 Dialogue and interaction style

If one is to talk about intelligent machines, then communication with the machine must be conducted along intelligent lines. This involves a strategy for discourse and the ability of the system not only to be able to understand communication in natural language, but also to be able to respond appropriately. Clancey (1982) finds three types of knowledge especially desirable for a tutor. First, it needs to know about dialogue patterns, i.e., types of utterances recurring independently of speaker. These can be used to either interpret or to generate speech.[23] Second, it has to know the subject to be discussed. Finally, it would have to be able to put the communication in context, to generate the correct information for the student at the right moment. In GUIDON, Clancey implemented this as a *communication model*, incorporating a student model, a lesson plan of topics to be discussed, and a focus record, remembering the factors that have been discussed. This touches upon an interesting research question: how is the focus determined for discussions of long and complex problems?

[23] Speech, naturally, is here limited to a man-machine dialogue.

4.4.1 The SOPHIE Experience: Natural Language Engineering

The implementors behind SOPHIE (Brown *et al.*, 1982) had as the goal for the natural language interface to successfully understand any sentence users type into the system. One limitation is, however, that users are supposed to be above all interested in troubleshooting the electric circuit at hand, thus ensuring a dialogue centred around problem solving (and not philosophical discussions, to take a counter-example). The interface builds on two techniques. A *semantic grammar* is used to incorporate the domain semantics into the parsing process, which above all makes this process more efficient. The second technique is a *dialogue mechanism* which is able to handle frequently occurring constructs in natural language such as ellipsis, pronominal reference and anaphora. SOPHIE handled more than 90% of the queries correctly.

The semantic grammar was first realized in the form of LISP functions, which were hand-coded and thus made use of the powerful programming environment. No special formalism was necessary. Almost to be anticipated, this caused more problems in the end than it solved, mainly because the grammar became so difficult to understand. The representation was changed to an augmented transition network (ATN), which is more concise. The early advantage LISP had over ATN speedwise was overcome as an ATN compiler was built.

4.4.2. The LISP Tutor experience

The LISP Tutor (Anderson and Reiser, 1985; Anderson and Skwarecki, 1986) is one of the few computer-based tutoring systems which has passed the prototype stage and is now available commercially. It owes this partial success to the amount of work invested in the program—three person-years, or approximately 6000 hours. The program offers between 30 and 40 hours of instruction, which means that each hour of instruction "cost" around 200 hours work. The second, important factor is that the tutoring domain is restricted to LISP programming, which is a domain where the knowledge (at least at a beginner's level) is quite easily formulated.

The objective of the program is to teach students to program in LISP. This is achieved by posing a problem, which the student then tries to solve with the aid of the tutor. The methodology used is top-down, left-to-right programming; the tutor provides templates which the student fills in with code. As soon as mistakes are detected, the tutor provides ready-made comments as to how to correct the fault at hand. Misconception are modelled as 'buggy' production rules, so error correction is based on having the "right" misconception in the bug catalogue. The LISP Tutor contains 325 correct rules, and 475 faulty ones, in a sense the most common misconceptions. It seems the tutor's effectiveness is solely dependent on the amount and quality of the 'buggy' production; it must be difficult to attain an exhaustive encoding (my remark).

After studying transcripts of parts of the dialogue during a session, certain questions surface. Firstly, there does not seem to exist any meta-level teaching strategy. When an error is detected (at any level), it is corrected at once, disrupting the flow of the dialogue and "hiding" the focus. For example, consider the problem of writing a function to calculate factorials, where the main feature to be taught must be the use of recursion and the stopping criterion. In this case, it does not seem appropriate to go into detail about whether one system function is better than another.

Secondly, the interaction style is quite restrictive. Student initiative is limited to entering code and to experimenting with code. There is no way the student can build programs bottom-up, or mixing strategies so as to receive comments on a higher level. Anderson and Skwarecki (1986) state that immediate feedback is usually best, and that on average, top-down, left-to-right programming is the correct style. It can be argued whether this is a correct view, or only an idealized one. At the same time we have to remember that the LISP Tutor was to be used for teaching programming at a university; thus one of the main objectives *could* be to teach a certain style of programming.

4.5 The role of explanation facilities

What makes expert systems so interesting from a teaching point of view if the explanation facilities feature. It was originally thought that incorporating explanations in an expert system would not

only help to debug and test an expert system, assure a sophisticated user of the 'credibility' of the system, but also endow it with teaching capabilities (Hasling *et al.*, 1983). This was not the case—explanations were too rigid as a means of transferring knowledge about specific situations. The main drawback was that explanations did not take into consideration all the knowledge that went into a knowledge item (e.g., a rule), such as justification or causal relations to other items, and even left solution steps out of the reasoning chain. We will have a look at some different attempts to tackle these problems, i.e., work done with the intention of reinforcing the explanation power of expert systems.

4.5.1. The XPLAIN Experience

Traditionally, when creating expert systems knowledge is encoded, but its justifications are either incorporated implicitly or left out. To remedy this problem, Swartout (1983) proposes a system which, by using an automatic programmer, refines abstract goals into a consulting program. While doing this, the programmer leaves behind a trace of its reasoning (a refinement structure), allowing a consistent way of handling explanations by generating them automatically. For this effect, two bodies of knowledge are used: a *domain model*, containing the descriptive facts of the area; and the *domain principles*, prescriptive knowledge consisting of methods and heuristics, i.e., the knowledge of how to solve problems. The separation helps in producing higher level explanations, and allows independent modification of the knowledge.

Once the refinement structure has been created the explanations are generated. A phrase generator constructs English phrases directly from the knowledge base, and an answer generator determines what is to be said by choosing the appropriate parts of the refinement structure. The selection is guided by three factors, namely the state of the execution, knowledge of what has already been discussed, and the likelihood of the user being interested in a certain piece of knowledge. With this in mind, XPLAIN uses so called *viewpoints* separating out steps which are only interesting for a certain kind of user, e.g., a physician as opposed to a programmer. This can be done both on the level of individual steps as well as higher levels such as prototypes. The resulting functionality when answering WHY-questions is:

1 The appropriate level is chosen by examining the control stack of the interpreter, and selecting the first procedure which is to be included according to viewpoints.
2 The user is given a general overview by stating the aim of the next higher procedure.
3 If a domain rationale[24] was attached to the domain principle in question, its whole pattern is instantiated and converted to English.
4 The explanation is completed by describing the current step of the procedure. If another, nested WHY-question is posed, the system moves the current description level up one step.

The domain rationales improve the quality of explanations by explicitly stating facts or reasons which are otherwise implicit, thus impossible to get at.

4.5.2 The NEOMYCIN Experience

NEOMYCIN (Hasling *et al.*, 1983), as the name implies, was created from the experiences drawn from MYCIN (Shortliffe, 1976). Its approach is characterized in that it tries to ". . . model human reasoning, representing control knowledge (the diagnostic procedure) explicitly." This requires separating control knowledge from domain knowledge and from code which is application-specific. NEOMYCIN's explanations are called strategic because they aim at making clear plans and methods in reaching a goal, not just showing the goal itself, Differing from other approaches, the strategic knowledge is thus completely separate from domain knowledge, and is in addition instantiated dynamically as the consultation proceeds, allowing the program to discuss both a

[24] A domain rationale provides additional information necessary to achieve a goal, apart from the method itself. For example, a domain rationale may define the characteristics required of a finding in order to consider it for a certain goal.

general as well as a specific problem solution. Explicit representation handles cases where, for example, control knowledge is embedded in rule clause ordering.

Several criteria guided the design of NEOMYCIN:

- Explanations should be informative, i.e., delivering a rule number is not sufficient.
- They should not presuppose any particular user population.
- Depending on the situation, an explanation should be either concrete or abstract.
- The system should be flexible enough to change terminology according to user type.
- Different levels of explanations should be possible; details should be included in certain situation, omitted in others.

These guidelines are implemented around a *task structure*, which are the goals and subgoals of the problem, and *metarules*, which are the methods to achieve these goals. Instead of just presenting the reasoning trace of a deduced fact like in MYCIN, NEOMYCIN tries to put the explanation into a context (Figure 9).

Examples of some of the generic tasks[25] in the system are to identify a problem, process data or generate questions, review a differential diagnosis, collect information by establishing a hypothesis space, etc. Each metarule builds on some task, and conversely, each task is achieved by a collection of metarules. A metarule has a premise part which determines when it is applicable, and an action part, which could be to find out about a domain goal, apply a rule or invoke other tasks. Essentially, WHY questions correspond to the premise part of a metarule, and HOW questions to the action part.

NEOMYCIN explanations are more on the level of 'teaching' than MYCIN rules were; nevertheless, some similarities are apparent. Where MYCIN talks about a goal, NEOMYCIN discusses a task; a rule corresponds to a metarule. The improvement lies in that the tasks and metarules are domain-independent, significantly improving reusability. Among feature development issues for NEOMYCIN, the authors mention four major ones; developing discourse rules which determine levels of explanation for different users or user types; a means to extract main points from a whole session into a summary; higher level reasoning about metarules such as why a rule failed, why rules are ordered in a certain fashion and how rules are justified; and finally, by recognizing that the same procedure is applied in different situations, the possibility of using analogies.

Problems normally encountered when trying to produce explanations using the canned text approach include the following (Swartout, 1983):

- As the code and the text that explains it can be changed independently, problems of inconsistency arise. Cases could appear where the program does something totally different from what it *claims* it is doing.
- The obvious problem of having to find all the cases for which explanations are desired.

17) Has Mary been hospitalized recently?
*** WHY*
[i.e. WHY is it important to determine whether Mary has been hospitalized recently?]

[21.0] We are trying to round out the diagnostic information by looking generally into past medical history and by reviewing systems.
There are unasked general questions that can help us with the diagnosis.

Figure 9 Explanation taken from NEOMYCIN (Hasling *et al.*, 1983).

[25] Generic in this context mean that tasks do not contain application-dependent information.

- Advanced explanations require that the system has a conceptual model of what is being done, so that reasons and justifications can be given at the proper level.

Using a direct transformation from program code (or trace) to explanations instead produces other defects or limits. One is that the quality of the explanations depends on the state of the code. What constitutes the program also shows up in the explanations, although good code could form bad explanations, or *vice versa*. Another possible disadvantage is that it might be difficult to structure the program so that it is easily understandable by the user. As programming by necessity has to be very detailed, some explanations could be too low level. The EXPLAIN system handles this through its viewpoints, which allow distinguishing between the interests of different user categories.

4.6 Machine learning and self-improving systems

An obvious extension, or rather an ultimate goal of any theory claiming to be intelligent, is to have systems which improve their performance over time, After trying out a certain way of teaching (say) and noticing that it is not as effective as other methods for a significant number of cases, the system should be able to re-evaluate its preference (if such exist) for the strategy and use a better one.

It is important at this point to note the difference between an *adaptive* and a *self-improving* system. As defined by Hartley and Sleeman (1973), and adaptive system is one which is able to generate teaching material and vary its difficulty depending on the student's competence. Feedback is chosen according to individual preference or effectiveness, and remedial exercises are generated and monitored for each error. However, the rules controlling the teaching operations and the representation of tasks and users require experimentation, and what is more important, they are specific to the domain, the type of task and the user population. Therefore, an adaptive system may become more accurate in using decision rules, but will, nevertheless, use these rules even when they are not satisfactory. A self-improving system, will on the other hand, change strategy altogether if the need arises. In effect, what a self-improving system has to do is to be able to form hypotheses. Experiments, of course, have to be subtle, so as not to impinge on the teaching process.

An attempt at using machine learning for student modelling has been made by Self (1986). He concludes that traditional machine learning is unsatisfactory for computer tutoring as it has concentrated on formal concepts with precise definitions; tutoring, he claims, is most of the time not formally defined and "... those (concepts) that are, are learned through the progressive refinement of informal definitions." In Self's view, a computer should be able to deal with informal concepts, as these are the first the users try to learn.

5 Summary and concluding discussion

We have above described the history and development of Intelligent Tutoring Systems. It is notable that the area seems to have reached some kind of barrier at the beginning of the 1980s. New systems are still being created, using more and more advanced hardware and software. However, the results as to what concerns novelty are not obvious; efficiency and usefulness have not improved much. Recent systems excel in areas like flexibility of interface design, user-friendly presentation methods using (amongst others) qualitative techniques and graphical possibilities, and diminishing time for producing prototypes through improved editing facilities. According to Yazdani (1986*b*), ITSs need a counterpart for authoring languages. This is one area which *de facto* has advanced during the last years. Many of the prototypes produced and evaluated up until now have almost emphasized the interface and editing tools. The editors could be compared to authoring systems—they simplify the process of creating the knowledge base, and broaden the bandwidth of communication—text to pictures, diagrams, graphs and sound. What needs to be

done, is to try to make these tools more general, so that they are able to handle classes of applications instead of only one instance.

Nevertheless, there are areas in which advances are not as far-reaching, and this mostly involves teaching itself. There remains much to be done in the area of explanation, which is very central. Explaining by presenting traces of computer-performed reasoning is one thing—being able to find the core of the student's misconception and remedying it is another. Student modelling has reached the stage when techniques are beginning to appear, but it is still far from producing a cognitive diagnosis module which can be "plugged" into an existing system. Knowledge representation still remains a core issue, where the major goals are to be able to represent various types of knowledge such as declarative or procedural, to make knowledge reusable by different presentation techniques for different purposes, to make it transparent, so that explanations can be generated directly from it, yet still keeping it efficient so that "reasoning" remains a quick process.

An area in which there is much still to be done includes teaching strategies. First, there is the need to develop various strategies, each one being suitable for one type of student in one type of teaching. There is no one strategy which is "best", even though in some situations a strategy may cover several students or a type of teaching. Second is the question of representation. Teaching strategies, just like other types of knowledge, need to be represented in a clear, transparent way. Here much remains to be done—up until now, strategies have either been represented as production rules, or have been implicit in the interpreter or the knowledge structures.

5.1. The KNOWLEDGE-LINKER project

In the above, we have tried to identify areas of promise for generic applications, techniques and methods which can be reapplied for classes of problems as they are in other engineering disciplines. The result is a design for a coaching system enhancing the functionality of a knowledge-based system, the core issue being the reuse of problem-solving knowledge (Sokolnicki, 1990). Below, we relate this design to other approaches, thus giving an account of an approach we deem feasible, and a logical consequence of the problems mapped out above.

The context in which this expert coach was designed is the KNOWLEDGE-LINKER project, which deals mainly with four problems: tools for domain-oriented knowledge acquisition allowing the expert to actively participating in knowledge base development, automatic generation of specialized instances of knowledge acquisition tools, knowledge representation and reasoning strategy resembling that of the expert and transfer of knowledge through tutoring. The reference domain, biochemistry, or more specifically purification of proteins, was selected for several reasons. It not only offers a realistic and complex domain, but also exhibits the characteristics for a knowledge-based system to be of value. The development cost is thus shared by several applications, thus dealing with a common problem of intelligent tutoring systems.

The coaching system centres around four concepts that need to be added to the knowledge system apart from the tutoring interpreter. Teaching strategies are implemented through *instructional prototypes*, or partial plans. Each prototype is a prescription for how to go from one state description (preconditions) to another (effects). The prototypes are composed of *teaching operators*, which are to be implemented as discourse plans based on a logic of belief (Litman and Allen, 1987). The set of instructional prototypes can be used to explicitly define individual teaching styles within the coaching paradigm. To be able to deal efficiently with common situations such as frequently encountered misconceptions, *error descriptions* are provided and can be used as input to prototypes. Finally, the actual interaction is controlled by an *intervention strategy*, which either automatically (based on the user model) or manually (controlled by students or instructors) can be adapted to individual needs or preferences. The parameters that can be controlled are when to interrupt the student, sufficient reasons why (s)he should be interrupted and the actual contents and form of the interruption. The solution chosen to deal with the content of the interruptions is based on five types of hints: attention, level, property, motivation and configuration. Most of these

can be generated automatically from the knowledge base, the rest can be handled by tailoring the knowledge acquisition tool to support necessary input.

5.2 Instructional goals, strategies and tactics

Several researchers have recognized the need to separate out three different levels of goals for a tutor (Breuker *et al.*, 1987; Woolf, 1984): pedagogic,[26] strategic and tactical. They correspond fairly well to the ones in our proposed design—our instructional goals as the pedagogical goals, instructional prototypes at the strategic level and teaching operators as tactics. Woolf's design (1984) centres around discourse procedures (or schemas) which lead the dialogue between states at the above-mentioned levels, while Breuker *et al* (1987) advocate strategic planning of the interaction and indexed retrieval of tactics from a data structure. Breuker's approach to planning the strategic level is similar to ours, by using skeletal strategies in a recursive process of selection and refinement. However, instead of indexing individual tactics, we retrieve whole instructional prototypes. First, in the next step a selection of appropriate teaching operators is performed. Both approaches stress the point that strategies must be planned dynamically so as to adjust to prevailing circumstances and not be fixed in advance.

The number of proposed tactics for both approaches lies around 25–30, which would be interesting to validate empirically— both the amount of basic tactics as well as their generality. As Breuker observantly notes, even though tactics can be described in more abstract terms, there is no theory to justify their adequacy. He also states that strategies with simple branching will not be sufficiently powerful to handle more complex domains. This was also noted by us, to which end we introduced extensions to our instructional prototypes. Apart from the "normal" decision points where instructors may enter self-defined parameters through decision tables, the extension includes iteration, scoping and interleaving. This could be a step in the right direction, especially to handle more systematic attempts to transfer domain knowledge. Our proposed teaching operators form a tentative discourse plan hierarchy which has some similarities with Woolf's (1984) discourse management network.

As to the pedagogical goals, Breuker *et al.*, (1987) suggests that they be generated automatically from a representation of the domain once and for all. He identifies the following relations between concepts: generalisation—specification, inversion, divergence—convergence, analogy and abstraction—concretion. This is similar to Goldstein's (1982) work on genetic graphs, with the exception that the graphs are the actual domain representation, while Breuker advocates generating the relations as an overlay on the domain knowledge.

5.3 Approaches to intervention strategies

Several different approaches to intervention strategies have been tried. The four basic questions to be answered are *when* to interrupt,*why* the interruption is called for, *what* to say and *how* to say it. Few results regarding computer presentation in the context of instructional systems have been empirically validated, with one of the exceptions being the work of Park and Tennyson (1986). They report a comparison between two instructional design strategies, "presentation form of examples" and "sequence of concepts", where the first determines the format of examples while the second adjusts the selection of examples. Their most tangible result is that an example selected for conceptual knowledge formation is more effective when presented in an expository form, while an example selected for procedural knowledge development is more effective when presented in an interrogatory form. More empirical results of this nature are necessary if a firm ground on which to base an instructional theory is to be developed. It is a fact that many educational strategies have been proposed and tested by educational psychologists, but few (if any) are suitable for

[26] Breuker refers to them as didactic goals, but we have chosen the more common term.

implementation. This could be explained by the fact that teachers, students and other people involved in the process are intelligent enough to fill in gaps or details if need arises.

The WEST system handles the content of interruptions by four levels of hints: the first points out an issue on which the student is weak and which is required to make an optimal move; the second shows all possible moves in a situation; the third selects the optimal move and motivates it; and the fourth will describe in detail how to make that move. These hints are similar to our hint types—the one that we have omitted is the second, showing all possible moves. Otherwise the first, third and fourth correspond to our hint types *level + property*, *motivation* and *configuration*, respectively. It is very interesting to note how similar these hint types are to ours, especially as our reference domain, biochemistry, is so dissimilar to the gaming environment in WEST.

The EUROHELP.PO system (Breuker, 1988) has an interrupt threshold defined by five levels of seriousness of user error. In addition, the style of tutoring is directly influenced by two other parameters, namely the required level of detail and the level of specificity. These are locally controlled by the user, who can request more information or more specific tutorial actions. Users can also decline the offer for advice or coaching—this might sometimes be overlooked, but is an important factor towards user acceptance. Four different tutoring needs are identified, with corresponding tutoring functions to be invoked when a need is encountered: a lack of basic concepts (in the user model) leads to *initial instruction*; a lack of potentially relevant concepts to *expansion* of already known concepts; of the user performs a correct action, *feedback* is given; and finally, when an error or misconception is recognized, *remedial action* is taken. Naturally, the tutor should sometimes merely present information to the user and not carry out a lengthy dialogue. This differentiates a tutor from a help system, where the latter does not recognize long term learning goals.

5.4 Authoring knowledge-based tutors

Among the obstacles on the road to accessible and relevant ICAI systems is the time it takes to develop them. As Breuker (1988) notes, even though there is an overproduction of shells for expert systems, there are still none for tutoring systems. He supplies two reasons for this: ITSs in general have a more articulate architecture, and employ heterogeneous formalisms for representing the knowledge required for teaching. In the knowledge acquisition phase, support is needed to gather and maintain domain knowledge, and to flexibly select instructional knowledge to suit individual needs. All this knowledge, especially the instructional, needs to be inspectable.

Begg and Hogg (1987, p.327) point to the four major tasks an intelligent authoring system would have to carry out:

1 *Domain knowledge*: create and maintain a knowledge base.
2 *Instructional knowledge*: specify rules for teaching, error recognition and remediation.
3 *Presentation*: define the format for user input, program output and display information.
4 *Consistency*: help identify inconsistencies and incompleteness in the knowledge bases.

This is a fairly abstract description, but it nevertheless identifies the more important tasks of an authoring system. It is interesting to note what is described is in effect a knowledge acquisition tool extended with support to handle instructional knowledge. Unfortunately, inputting instructional knowledge requires having an instructional interpreter, and this interpreter in turn must be defined in such a way as to allow expressing variable pedagogical strategies.

There are several systems that in one way or another were meant to function as authoring systems or shells, among those BIP (Barr *et al.*, 1976), IDE (Russell *et al.*, 1988) and SMITHTOWN (Shute and Bonar, 1986). In BIP, curriculum scripts and topics control the task selection process; SMITHTOWN centres around pedagogical issues called *bites* instead of the usual diagnostic or expert modules. Each such bite embodies a miniature system with knowledge about its conceptual relations to other bites (*cf.* genetic graphs), relations to remedial actions, a miniature student model, a component able to diagnose the user's use and disuse of the bite and a generator of

problems (cases) and instructional interventions. This approach is organizationally quite attractive, especially its object-oriented qualities.

In order to reduce development time, data and knowledge structures should be provided that can be easily refined and edited for new applications. Our instructional prototypes and teaching operators are such structures, just like the discourse schemas suggested by Woolf and Murray (1987) based on an extended ATN notation. The advantage of these two approaches compared to a pure production system discourse manager like GUIDON's, according to Woolf and Murray, is that they can handle sequencing of situation action chains in a consistent way (Woolf and Murray, 1987, p.191). Other work useful towards this aim is the construction of meta tools, i.e., tools that can be used like the authoring systems to reduce implementation time for knowledge acquisition programs (Musen, 1989).

According to Russell *et al.* (1988), ". . . a gap exists between understanding a set of instructional principles and consistently employing those principles." This is the rationale underlying their work on designing a system to support the development of instructional courses and to help articulate and structure domain knowledge, a system called IDE (Instruction-Design Environment). IDE is aimed at supporting two main difficulties: first, managing the size and complexity of courses; second, adapting the presentation of content according to user characteristics. Facilities are provided for recording the rationale for decisions taken during the development process, by linking decisions with their evidence.

The course is described in terms of literature, external constraints and course objectives, instructional principles, epistemology, cognitive decisions, presentation, knowledge structures, student models, course control and instructional units (IU). The IUs are the final output of the system, and form the interface to the user. Without changing the course content at large, the IUs can be changed to contain different presentational means, so as to define delivery modes such as workbook and lecture, interactive videodisk or computer-based training. Modification is supported by *tracers* and *checkers* which can follow rationale links to map out consequences of a change.

Instructional strategies in IDE are represented as rules creating instructional goals, and constraints restricting the implementation of them. More specifically, this knowledge is divided into three parts: *pedagogy*, *strategy* and *tactics*. The pedagogy rules are course-specific, and control the topic sequencing of the course. IDE's strategies correspond to our instructional prototypes; they are described as encoding instructional methods such as: "*drill and practice*", "*present three concepts, then test for understanding*" or "*monitor a student's solution of a problem, and intervene as soon as the student varies from the solution path*" (the latter seems to be very similar to our overall coaching paradigm). The tactical rules define the lowest level, i.e., the actual display information for a particular type of student.

The initial process of designing a course to include complete specifications of all the above-mentioned concepts is quite laborious, as the authors admit. Once the specifications are there, IDE will certainly support the reuse and modification of course content. It would seem that one of the drawbacks of IDE is the loose definition of its output. Various forms of text-based courses can probably be developed, but the authors have not yet given a complete description of what is needed to support the more complex and dynamic forms of instruction like computer-based training or tutoring. Russell gives an account of the IDE interpreter, and states that to make it operate as a tutoring system the course author needs to create a large amount of supplementary material (Russell, 1988 p.2 29). Even though the work of providing a tutoring interpreter would be substantial, much of the domain knowledge entered through IDE could probably be reused, together with the instructional rationale.

IDE is similar to our approach in that it also tries to reuse knowledge, both domain and instructional. Our design is more narrow, as it is tailored towards a coaching situation in a specific domain; in return it allows more efficient use of domain knowledge and experimentation with various dimensions of coaching. This is naturally a trade-off, and we have paid for it by a loss of generality. The strategic and tactical levels have direct correspondents in our design through the

instructional prototypes and teaching operators respectively. The instructional planning in IDE is carried out by forward-chaining from instructional goals to a plan structure, with backtracking when no IU can be found. The K-linker instructional planner is comparable in that it also "forward-chains" from preconditions, but allows selecting a central prototype according to some criteria and planning backward to the instructional preconditions as well as forward to the desired effects.

5.5 Cognitive considerations

In the longer perspective, the key towards developing better teaching systems lies in understanding the mental models that users incrementally build when carrying out a cognitive process. Waern (1989) identifies three main issues: (a) what the role of a user's mental model in a learning situation is; (b) what model the user should have of the system; and (c) how the mental model should be acquired. Answering these questions would mean formulating a theory about the cognitive processes involved in learning. The resulting models can be of two types: vague and formal. In order to implement the theory, we need formal models which may be used to draw inferences about or to validate the user's cognitive state. At the same time, more informal models are useful for learning purposes, for example to lessen the burden on working memory load.

When modelling users, we need to take into consideration not only a general theory, but also the individual differences between users. These differences may be more or less prone to change through influence, as some are of a more fundamental nature. In the following, four cognitive variables are described in terms of how stable they are, from most stable to most changeable by outside influence (adapted from Waern, 1989, p.113):

Personality factors	Intelligence
	Extroversion/introversion
	Fear of failure
	Creativity
Cognitive style	Context (or field) independence
	Visual/verbal
	Impulsive/reflective
Learning style	Heuristic/systematic
	Operation/comprehension learning
Personal knowledge	Schemata
	Semantic network
	Production rules

It seems therefore that as a consequence a tutoring system should focus primarily on delivering desirable and necessary knowledge, i.e., such knowledge that would help a user understand and solve problems, as it is easier to assimilate; only when this is "guaranteed", the systems should be made so as to allow users to change their teaching style and adjusting for example presentation means.

However, the conclusion could also be that as cognitive and learning style are more difficult to change than that personal knowledge, no matter how relevant the system's output is, it will be difficult to assimilate as long as the teaching paradigm and cognitive preferences are fixed. Lesgold (1988) recognizes this dilemma, and suggests three types of knowledge to be included in an intelligent tutor: (1) a representation of the goals for the knowledge to be taught; (2) knowledge about the curriculum, i.e., "lessons" with prerequisites; and (3) a representation of more enduring characteristics to which instruction should be sensitive. Among the latter we find things like aptitude or metacognitive skills, i.e., learning ability or preferences, and domain-specific capabilities relating both to the result of education and specific learning capabilities.

Acknowledgements

I am greatly indebted to Professor Sture Hagglund for his constant support and many valuable discussions.

References

Anderson, J R, 1982, "Acquisition of cognitive skill" *Psychological Review* **89** (4) 369–406.

Anderson, J R, 1983, *The Architecture of Cognition* Harvard University Press.

Anderson, J R, Boyle, C F and Yost, G, 1985, "The geometry tutor" *Proceedings of IJCAI-85*. Los Altos, Morgan Kaufman.

Anderson, J R and Reiser, B J, 1985, "The LISP Tutor" *Byte* **10** (4).

Anderson, J R and Skwarecki, E 1986, "The automated tutoring of introductory computer programming" *Communications of the ACM* **29** (9) September.

Anderson, J R , Boyle, C F, Farrell, R and Reiser, B J, 1987, "Cognitive principles in the design of computer tutors" In: Morris, P, ed., *Modelling Cognition* Wiley.

Barr, A, Beard, M and Atkinson, R, 1976, "The computer as a tutorial laboratory: the Stanford BIP Project" *International Journal of Man-Machine Studies* **8** 567–596.

Barr, A and Feigenbaum, E, eds., 1982, *The Handbook of Artificial Intelligence, Vol II* William Kaufmann.

Begg, I and Hogg, I, 1987, "Authoring systems for ICAI" In: Kearsley, G. ed., *AI & Instruction: Application and Methods* Addison-Wesley.

Bonar, J, Cunningham, R and Schultz, J, 1986, "An object-oriented architecture for intelligent tutoring systems" *Proceedings of the Conference in Object-Oriented Programming Systems, Languages and Applications*, Portland, Oregon.

du Boulay, N, O'Shea, T and Monk, J, 1981, "The black box inside the glass box: presenting computing concepts to novices" *International Journal of Man-Machine Studies* **14** 237–249.

Breuker, J, Winkels, R and Sandberg, J, 1987, "A shell for intelligent help systems" *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI)* Milan, Italy, August.

Breuker, J, 1988, "Coaching in help systems" In: Self, J, ed., *Artificial Intelligence and Human Learning: Intelligent Computer-aided Instruction* Chapman and Hall.

Brown, J S, Burton, R and Larkin, K, 1977, "Representing and using procedural bugs for educational purposes" *Proceedings of 1977 Annual Conference, Association of Computing Machinery* 246–255, Seattle, WA.

Brown, J S, Burton, R and de Kleer, J, 1982, "pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III" In: Sleeman, D and Brown, J S eds., *Intelligent Tutoring Systems* Academic Press.

Burton, R and Brown J S, 1979, "An investigation of computer coaching for informal learning activities" *International Journal of Man-Machine Studies* **11** 2–24.

Carbonell, J, 1970, "AI in CAI: an artificial intelligence approach to computer assisted instruction" *IEEE Transactions on Man-Machine Systems* **11** 190–202.

Clancey, W J, 1982, "Tutoring rules for guiding a case method dialogue" In: Sleeman, D and Brown, J S eds., *Intelligent Tutoring Systems* Academic Press.

Clancey, W J, 1983, "The Epistemology of a rule-based expert system—a framework for explanation" *Artificial Intelligence* **20** 215–251.

Clancey, W J, 1987, Methodology for building an intelligent tutoring system" In: Kearsley, G, ed., *Artificial Intelligence and Instructions—Applications and Methods* Addison-Wesley.

Dede, C, 1986, "A review and synthesis of recent research in intelligent computer-assisted instruction" *International Journal of Man-Machine Studies* **24** 329–353.

Eriksson, H and Sandahl, K, 1989, "Knowledge-based planning of experiments in a biochemical domain—membrane protein purification" *Proceedings of the Second Scandinavian Conference on Artificial Intelligence* Tempere, Finland.

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

Ford, L, 1987, "Teaching strategies and tactics in intelligent computer aided instruction" *Artificial Intelligence Review* **1** 201–215.

Freedman, R S, 1984, "Knowledge-based courseware authoring" *Training Technology Journal* **1** (4) 4–9.

Freedman, R S and Rosenking, J, 1986, "Designing computer-based training systems: OBIE-1:KNOBE" *IEEE Expert*.

Genesareth, M, 1982, "The role of plans in intelligent teaching systems" In: Sleeman S and Brown, J S, eds., *Intelligent Tutoring Systems* Academic Press.

Goldstein, I and Carr, B, 1977, The computer as coach: an athletic paradigm for intellectual education" *Proceedings of 1977 Annual Conference, Association for Computing Machinery* 227–233, Seattle.

Goldstein, I, 1982, "The genetic graph: a representation for the evolution of procedural knowledge" In: Sleeman, D and Brown, J S, eds., *Intelligent Tutoring Systems* Academic Press.

Hartley, J R and Sleeman, D H, 1973, "Towards more intelligent teaching systems" *International Journal of Man-Machine Studies* **5** 215–236.

Hasling, D, Clancey, W and Rennels, G, 1983, "Strategic explanations for a diagnostic consultation system" *Report No. STAN-CS-83-996* Department of Computer Science, Stanford University.

Hollan, J D, Hutchins, E L and Weitzman, L, 1984, "STEAMER: an interactive inspectable simulation-based training system" *The AI Magazine*.

Hagglund, S and Rankin, I, 1988, "Investigating the usability of expert critiquing in knowledge-based consultation systems" *Proceedings of the European Conference on Artificial Intelligence*, Munich, West Germany.

Johnson, L and Soloway, E, 1987, "PROUST: an automatic debugger for pascal programs" In: Kearsley, G, ed., *Artificial Intelligence and Instruction* Addison-Wesley.

Kearsley, G and Seidel, R J, 1985, "Automation in training and education" *Human Factors* **27**(1) 61–74.

Lesgold, A, 1988, "Toward a theory of curriculum for use in designing intelligent instructional systems" In: Mandl, H and Lesgold, A, eds., *Learning Issues for Intelligent Tutoring Systems* Springer-Verlag.

Lewis, M, Milson, R and Anderson, J, 1987, "The TEACHER'S APPRENTICE: designing an intelligent authoring system for high school mathematics" In: Kearsley, G, ed., *Artificial Intelligence and Instruction—Applications and Methods* Addison-Wesley.

Litman, D and Allen, J F, 1987, "Discourse processing and commonsense plans" In: Cohon, P and Perroult, C, eds., *Formal Theories of Communication* Linguistic Institute, Stanford University.

McGuire, Ch, Solomon, L M and Bashook, P G, 1972, *Handbook of Written Simulations* Center for Educational Development, University of Chicago, Ill,

Montague, W E and Wulfeck, W H II, 1984, "Computer-based instruction: will it improve instructional quality" *Training Technology Journal* **1** (2) 4–19.

Musen, M, 1989, "Conceptual models of interactive knowledge acquisition tools" *Knowledge Acquisition* **1** (1) 73–88.

Nordin, H, 1985, "On the use of typical cases for knowledge-based consultation and teaching" Thesis 48, LiU-Tek-Lic 1985:13, Department of Computer Science, Linköping University.

Ohlsson, S, 1986, "Some principles of intelligent tutoring" *Instructional Science* **14** 293–326.

O'Shea, T, Bornat, R, du Boulay, B, Eisenstadt, M and Page, I, 1984, "Tools for creating intelligent computer tutors" In: Elithorn, A and Banerji, R, eds., *Artificial and Human Intelligence* Elsevier Science Publishers.

Park, O and Tennyson, R, 1986, "Computer-based response-sensitive design strategies for selecting presentation form and sequence of examples in learning of coordinate concepts" *Journal of Educational Psychology* **78** (2) 153–158.

Park, O-C, Perez, R and Seidel, R, 1987, "Intelligent CAI: old wine in new bottles, or a new vintage?" In: Kearsley, G. ed., *Artificial Intelligence and Instruction—Applications and Methods* Addison-Wesley.

Peachey, D R and McCalla, G I, 1986, "Using planning techniques in intelligent tutoring systems" *International Journal for Man-Machine Studies* **24** 77–98.

Reiser, B, Anderson, J and Farrell, R, 1985, "Dynamic student modelling in an intelligent tutor for LISP programming" *Proceedings of the 9th International joint conference on Artificial Intelligence* 8–14, Los Angeles, CA.

Rich, E, 1983*a*, "Users are individuals: individualizing user models" *International Journal of Man-Machine Studies* **18** 199–214.

Rich, E, 1983*b*, *Artificial Intelligence* McGraw-Hill.

Russell, D, Moran, T and Jordan, D, 1988, "The instructional-design environment" In: Psotka, J, Massey, L and Mutter, S, eds., *Intelligent Tutoring Systems: lessons learned* Lawrence Erlbaum Associates.

Russell, D, 1988, "IDE: the interpreter" In: Psotka, J, Massey, L and Mutter, S, eds., *Intelligent Tutoring Systems: lessons learned* Lawrence Erlbaum Associates.

Sacerdoti, E D, 1977, *A Structure for Plans and Behavior* Elsevier.

Self, J, 1986, "The application of machine learning to student modelling" *Instructional Science* **14** 327–338.

Shortliffe, E H, 1976, *Computer-Based Medical Consultations: MYCIN* Elsevier.

Shute, V and Bonar, J, 1986, "Intelligent tutoring systems for scientific inquiry skills" *Proceedings of the 8th Annual Conference of the Cognitive Science Society* Amherst, MA.

Sleeman, D and Brown, J S, 1982, *Intelligent Tutoring Systems* Academic Press.

Sokolnicki, T, 1990, "Coaching partial plans: an approach to knowledge-based tutoring" Licentiate Thesis 237, LiU-Tek-Lic 1990:237, Department of Computer Science, Linkoping University.

Stevens, A, Collins, A and Goldin, S, 1982, "Misconceptions in students understanding" In: Sleeman, D and Brown, J S, eds., *Intelligent Tutoring Systems* Academic Press.

Streibel, M J, Stewart, J, Koedinger, K, Collings, A and Jungck, J R, 1987, "Mendel: an intelligent computer

tutoring system for genetics problem-solving, conjecturing, and understanding" *Machine-Mediated Learning* **2** (1–2) 129–159.

Swartout, W R, 1983, "XPLAIN: a system for creating and explaining expert consulting programs" *Artificial Intelligence* **21** 285–325.

Waern, Y, 1989, *Cognitive Aspects of Computer Supported Tasks* Wiley.

Walters, J and Nielsen, N, 1988, *Crafting Knowledge-based Systems: Expert Systems Made Easy Realistic.* Wiley.

Woolf, B, 1984, "Building a computer tutor: design issues" *IEEE Computer* September.

Woolf, B, Blegen, D, Jansen, J and Verloop, A, 1986, "Teaching a complex industrial process" *Proceedings of the AAAI-86*, Philadelphia.

Woolf, B and Murray, T, 1987, "A framework for representing tutorial discourse" *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI)* Milan, Italy, August.

Woolf, B, 1987, "Theoretical frontiers in building a machine tutor" In: Kearsley, G, ed., *Artificial Intelligence and Instruction—Applications and methods* Addison-Wesley.

Yazdani, M, 1986*a*, "Intelligent tutoring systems survey" *Artificial Intelligence Review* **1** 43–52.

Yazdani, M, 1986*b*, "Intelligent tutoring systems: an overview" *Expert systems* **3** (3) July.