TOWARD A GENERAL ARCHITECTURE FOR INTELLIGENT TUTORING SYSTEMS

Tom Murray

COINS Technical Report 87-89

Abstract

This paper discusses design issues for intelligent tutoring systems, and suggests a general and extendable architecture for program control and knowledge representation. The main focus is on how discourse, teaching, and diagnostic rules, along with the information needed to implement these rules, can be acquired, represented, and coordinated. First, the state of the art in ITS design, important problems, and research goals are discussed. Then, a rule-based, goal-driven architecture is suggested which facilitates an evolving knowledge base, intelligent planning of tutoring actions, dynamicly shifting between different tutoring styles, and the maintenance of student and discourse models. Finally, the principles underlying knowledge acquisition and the identification of design constraints are discussed, and a step-by-step methodology is given. Selected sections of the paper serve well as an introduction to, or review of, the important design issues in ITS.

---

[1]This work was done in conjunction with the Cognitive Studies of Computers in Learning Program, at the UMass School of Education, and with the Intelligent Tutoring Systems Group, in the UMass Department of Computer and Information Science.

[2]This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 ITS systems state of the art

It has been proposed and verified in limited instances that computers have the potential to act as powerful intelligent tutors–to have an expert level understanding of the the knowledge in their domain, an understanding of the pedagogical aspects of that domain, and an ability to tailor instructional actions, on the fly, uniquely for different students and different instructional contexts (Wenger 85, Clancey 86, Sleeman & Brown 85). It is also becoming increasingly clear that computer tutors can provide instructional environments, such as graphic simulations and large structured data bases, that are not obtainable without the use of computers (Burton 86 and diSessa 86). Research in ICAI (Intelligent Computer Aided Instruction) has made significant progress over the last 10 years, but thus far very few ITS's (Intelligent Tutoring Systems, synonymous here with ICAI) have been built which have the scope and the robustness to have been tested extensively in actual academic settings. This is not surprising, considering the complexity of the tutoring problem, which requires state of the art AI methodologies in knowledge representation and acquisition, control, communication, interfaces, and machine learning. Each attempt to use a new ITS system on students uncovers new problems or questions for the pedagogical and cognitive theories in the domain used. All this not withstanding, several sectors of our society, strapped with severe educational and training problems, anxiously await the maturation of the field, and more concrete evidence that it can soon deliver the promise of its touted potential.

1.2 A common ground is needed

The issues facing the builders, designers, and evaluators of ITS systems are many and complex. As the number of ITS systems being proposed and built increases, so does the need for a common ground on which to exchange ideas and results. It is much too early to expect substantial commonality amongst the philosophical and strategic approaches in such theoretically alive areas as pedagogy, knowledge representation, and student modeling, but it is not premature to move toward some common terminological frameworks and guidelines with which to exchange and critique the various theories, tactics, and computer systems in ITS research.

1.3 Purpose of the architecture

This paper has two purposes: first, to present a general control and knowledge representation architecture for ITS systems. And second, to provide methodological guidelines for designing ITSs and analyzing ITS needs in any domain within the framework of the architecture.

The architecture proposed in Chapters 4 through 6 is in part an attempt at generalizing and synthesizing the implicit and explicit components of existing tutoring systems. Some of its features suggest reatively new ways to use existing AI technologies in ITS design. The architecture can be viewed as a "shell" within which to encode the specific information for arbitrary tutoring systems. As such, it is a step toward an authoring system tool for building ITSs.

This is a report on work in progress, and some of the ideas herein are under-constrained. The architecture has not yet been fully implemented, and the details presented in this paper are sure to evolve as we implement it. It is hoped that the issues raised and the solutions proposed will be a useful starting point for further research, even if some of the details fall be the wayside. As an attempt at looking at the "big picture" of ITS design, it focuses on how knowledge is used to control the actions of the system, and on how to modularize the various components within a framework that encourages well structured communication between them. As such, parts of this paper may also serve as an introduction to the important issues in the field, and an overview of the state of the art in ITS system design.

1.4 Purpose for the guidelines

Even if a generalized ITS shell or authoring system (as suggested above) existed, it would be difficult to utilize it effectively. The difficulties of specifying,with computational precision, domain knowledge and tutoring knowledge for such a system are prohibitive for all but the most experienced research teams (given the current state of the art). Two things are needed to ameliorate this problem: an authoring or knowledge acquisition interface which makes it feasible to have pedagogical and domain experts (as opposed to computer scientists) take a larger part in the design of the knowledge base, and methodological guidelines or accepted lore about the analysis of an instructional domain to produce an ITS specification. Chapters 7 and 8 propose such methodological guidelines. (We do not address the issue of knowldge acquisition interfaces directly in this paper.)

1.5 Overview of the paper

The intended audience for this paper are those involved in ITS research. We assume the reader has had some introduction to the field of intelligent tutoring systems, the more popular ITS systems described in the literature (see Sleeman & Brown 82 and Wenger 86), and an acquaintance with basic artificial intelligence concepts such as frames, production rules, inheritance, etc. (see Barr, Cohen, & Feigenbaum 82, Winston 84B). It is not intended to extol the virtues and potentials of Intelligent Tutoring Systems, though they are numerous and exciting. We assume the reader is familiar with the potential and proven capabilities and refer her to (Sleeman & Brown 82, Wenger 86, or Yazdani 86). Following is an overview of the chapters with suggestions for the reader.

Chapter 2

A discussion of the issues and problems in the field which motivate the rest of the paper. If you agree with the tenant that a general architecture is needed, this chapter could be skimmed.

Chapter 3

Gives a top level "systems view" of tutoring; an analysis of the human act of tutoring or teaching in terms of its functional components and the kinds of information exchanged between those components. The control and knowledge representation issues introduced here are

expanded in later chapters. The chapter should be light reading by those familiar with various ITS systems. For those less familiar with the field it may provide a good introducition.

Chapter 4

Knowledge representation issues. A description of the contents of the component data bases, and some design considerations; a bit more technical than the previous chapters. A passing familiarity with AI knowledge representation issues is assumed.

Chapter 5

An analysis of top level control issues; goal specifications of a tutoring session, including such elements as teaching goals, curriculum, and learning environments. No AI knowledge is needed.

Chapter 6

An analysis of the dynamic, low level control issues; components for dynamic decision making aspects of tutoring systems. We proposes a general control architecture for coordinating the types of information discussed in chapters 4 and 5. This is the most technical chapter. A passing acquaintance with AI control issues is helpful, but some of the issues are explained in detail. (The sections on disadvantages of vanilla rule-based control architectures may be skipped by those familiar with these issues.)

Chapter 7

A suggested methodology for tutoring system design. Chapters 4 through 6 were concerned with the knowledge representation and control structures containing the tutoring expertise. This chapter shows how the requirements of the various components of the system constrain the design process. A suggested scheduling of design activities is given. It is not a technical discussion, but assumes familiarity with the components of the architecture introduced in previous chapters.

Chapter 8

A knowledge engineering analysis of the factors, assumptions, and principles behind tutoring system design decisions, giving examples from existing systems. A methodology is suggested for analyzing the requirements of an arbitrary domain to generate overall goals and constraints for an ITS.

Chapter 9

Conclusion. Summary of the major points, and a summary of which aspects of the paper are considered contributions. Finally, some thoughts on the future (of ITSs).

Chapter 2

BACKGROUND

We will not attempt to provide the reader with a summary of ITS research projects and issues (see Sleeman & Brown 82 and Wenger 85). In this chapter we will first list some of the problems facing researchers in the field, which the paper addresses. Next we will list research goals motivated by these problems. Finally we will outline the solutions and suggestions addressed in this paper.

2.1 Problems and issues

2.1.1 ITS systems often have little pedagogical foundation

Many ICAI systems (synonymous with ITS's) have been built which incorporate little understanding of how successful humans tutors teach. Such programs are designed from the intuitions of a small number of AI researchers, often without collaboration with an expert teacher working in the field, and without extended testing of the tutoring strategies or assumptions implicit in the system on real students early in the design phase. Brown et. al. (page 279 of Sleeman & Brown 85): "...the work that went into SOPHIE II and SOPHIE III on explanation put the cart before the horse. We had no adequate theory of what it meant to understand a circuit and hence no well defined "target" model of what we wanted the student to learn." Clancey, in (Clancey 86 page 43) says "The student's knowledge (was) very different from MYCIN." Then he goes on to explain new insights in the organization and use of expert knowledge in the MYCIN-GUIDON- NEOMYCIN-HERACLES series of AI programs. But he says little about the process of learning the expert's knowledge. So far none of the tutors in the GUIDON

series have been shown to work with students, suggesting the same problems articulated by Brown above.

2.1.2 Each new system is built from scratch

Many intelligent tutors are being built and proposed, most of them designed from scratch. It is rare to see the control or knowledge representation schemes from one research group being used by another. General guidelines for ITS system design do exist, such as identifying the major components (student model, domain model, etc.— see Yazdani 86), but there are few specific guidelines on how to implement these components. Systems for organizing tutoring rules have been proposed (Clancey in Sleeman & Brown 85, Woolf 84A, Anderson, Boyle, & Reiser 85), but these systems have not been adopted. This may be due to the fact that they are not presented in the context of a generalizable system, or due to the tendency of AI researchers to "roll their own" rather than incorporate existing methodologies.

2.1.3 Existing ITS systems have limited scope

The majority of ITS systems have been designed to teach in limited domains and within limited pedagogical styles. For example, SOPHIE could only teach using a couple of electrical circuits, the BUGGY (Burton 82) and SIERRA (Van Lehn 83) projects deal exclusively with the subtraction skill. Focus on learning goals at the curricular level is rare (but see Anderson & Reiser 85, and Shute & Bonar 86). In most cases this is necessary because the research efforts have focussed on difficult theoretical issues, such as student modeling or qualitative simulation. But there is need for research on how ITS systems can be used to teach (or help teach) entire curricula using a variety of instructional environments and teaching styles.

2.1.4 ITS's are hard to evaluate

Current ITS systems are hard to evaluate comparatively (i.e. one to another) (see Soloway & Littman 86). Very few have left the lab to be used on large numbers of students where effectiveness can be evaluated statistically. The pedagogical and psychological assumptions behind the rules used in tutoring systems are largely implicit, and differ with each system (see Anderson, Boyle, Farrell, & Reiser 84, Collins 77, and Larkin 83 for examples of rules systems). This makes it hard to compare different systems, and hard to evaluate why a system failed to teach (Was it due to shallowness of the

domain knowledge representation, inadequacy of tutoring rules, or implicit theoretical assumptions?)

2.2 ITS research goals

2.2.1 More pedagogical and psychological foundations

Pedagogical considerations should be the motivating factors for designing ITS systems for specific domains. A theory of learning in the domain, and priorities concerning what types of learning and knowledge are most important for performance in the domain, should proceed the design of the tutor. ITS system designers need to work closely with experts in teaching during the design and evaluation phases of building an ITS. Ideally, the pedagogical assumptions should have been verified to some extent before the effort of building a tutoring system is expended (although it does appear to be very difficult to get conclusive statistical results on any sufficiently specific theory of learning or teaching). Also, assumptions made about learning and teaching in the domain should be made explicit, for the purposes of assigning credit and blame after evaluative experiments, and so that the system can be compared with other ITS systems. Each important design decision and each high level tutoring rule should be annotated with the pedagogical or psychological assumptions it embodies.

2.2.2 Flexible reusable ITS shells

The computer can provide unparalleled learning environments (Papert 80, Burton 86). Instructional strategies found effective in human tutoring can not be assumed to work in computer tutoring (although they must serve as a starting point). It is not possible to fully evaluate the effects of these new environments with off line testing. New environments or tutoring rules will need to be reasearched to gain confidence concerning a system's teaching potential. The goal of designing ITSs using sound pedagoly is hampered by this lack of information about how people learn in these new environments. ITS systems should be powerful enough to embody a variety of tutoring strategies, and allow easy reconfiguration of these strategies in research settings. Therefore, systems should be flexible enough to easily add or modify domain knowledge, or even plug in a pedagogically similar domain. Systems with this kind of generality facilitate research on computer aided learning, ameliorate the problem of starting from scratch every time, and steepen the

research learning curve.

2.2.3 Toward ITS authoring systems

It is not unreasonable, given the current state of the art, to design ITS systems which can be configured, modified, and/or tweaked by domain or pedagogical experts with training in AI concepts, but are essentially non-programmers. We are a far cry from designing anything like an ITS authoring shell, but there is much we can do in the way of making the workings of systems as transparent and modular as possible, in an effort to allow instructional experts to have a larger hand in ITS system design and evaluation. Perhaps we should build systems as if they were to be used or maintained by parties who are initially unfamiliar with the design decisions.

2.2.4 A more precise technical vocabulary

If, as is suggested above, designers of ITS systems routinely tried to consult instructional experts, and performed case studies of learning and teaching situations to verify the underlying assumptions of the tutoring rules used, it would be a difficult and ill-defined task because we do not have a concise vocabulary or ontology with which to discuss the principles involved in designing tutors. Anderson (in Anderson 84) says: "Instruction in general, and computer- based instruction in particular, is pretty much a black art."

If intelligent tutoring systems are to successfully emulate good human tutoring, we need to computationalize the rules of effective instruction which tutors know implicitly. A terminology with the required precision and completeness does not yet exist, but some are striving for one (Clancey 86B). Clancey (in Clancey 85, page 309) stresses the importance of terminology in the development process: "The very process of formalizing terms and relations changes what we know, and itself brings about concept formation." On the same page, he warns of "the difficulty of defining terms," echoing the growing concerns of all AI knowledge representation research.

2.2.5 Tutoring entire topics, breadth and depth

There is a need for more ITS research on systems capable of a more curricular focus, i.e. more focus on how to teach entire topic units or textbook chapters. For example, traditional science courses include lectures, labs, homework, discussion sessions, and exams, all within a (supposedly) well thought out sequence to topics to be covered. Even though there may be much to be

desired in the way science is traditionally taught, it is obvious that a variety of environments and tasks are needed for significant learning in any field.

Students should be presented with several instructional styles and/or learning environments in an attempt to teach concepts from several view points. Most existing systems, since they attempt to teach a very limited subject matter, use a limited number of tutoring rules and environments.

2.3 Solutions and recommendations addressed in this paper

ITS research is not plagued with numerous fundamental problems, as the above summary might suggest. In fact the field is making steady and exciting progress along many dimensions (for example, modeling qualitative reasoning, student plan recognition, and analysis of problem solving skill acquisition). The problems listed above are given only to motivate the subject of this paper: general architectures and guidelines for ITS design. Below we preview the research goal solutions which are addressed in this paper:

2.3.1 A general ITS architecture is proposed

A general architecture is proposed that addresses the needs of managing diverse types of domain-specific teaching knowledge and large sets of general tutoring rules. The system is modular and flexible, and should help with several issues. Modularity and re- usability will lessen the need to design systems from scratch. Clearly distinguished functional components will increase the effectiveness of evaluations and comparisons of systems, and clarify issues in the design phase. Also, flexible rule systems will allow experimentation with different configurations of tutoring rules and knowledge representation. A clear organization of the types of information used to drive the system should also make it easier for pedagogical experts who are not computer scientists to take part in the design of tutors.

2.3.2 An analysis of the ITS design process

Suggestions are given for an organization of design activities. The organization stresses the incorporation of sample tutorial dialogues (Chapter 7) and a domain analysis making explicit the educational, pedagogical, and psychological assumptions and goals of the research team (Chapter 8) which for the basis for global and local design decisions.

2.3.3 Some suggestions for terminological refinement

Interspersed throughout the paper are suggestions for more precise terminology and ontology dealing with knowledge representation, control, and design. Perhaps more important than the specific ontologies proposed, is the idea that one is needed, this paper giving an example case.

Chapter 3

ITS—A SYSTEMS PERSPECTIVE

3.1 The constructivist paradigm for learning

The "learner as container, teacher as filler" paradigm for instruction is giving way to a Piagetian inspired constructivist view (Piaget 72, Lawson 75, Confrey 85, Von Glasersfeld 78). The constructivist view is that one learns by constructing new knowledge from existing knowledge through the self-regulatory processes of assimilation and accommodation. In assimilation experiential information is incorporated into existing "schema" (patterns or groupings of information). Accommodation is the modification of schema (or instantiation of new schema) as a result of experiential information which does not agree with existing schema. Learning is not a passive process of imprinting incoming information in the brain, it is an active process of trying to maintain equilibrium in a changing environment. If we believe, as most educational psychologists and epistemologists do today, that learning is an active process, then teaching can be seen as assisting a student in the learning process. This contrasts with the more didactic belief that knowledge is "conveyed" or "given" to students, who, if they pay attention, passively absorb it. In terms of the constructivist paradigm, teaching consists of: 1. providing a motivating environment in which to learn, and/or 2. specific guidance, helping the student "travel" from one knowledge state to a more desired one. There is a range of strategies which span the spectrum from passive learning environments to actively guiding the student, and we will outline these in Chapter 5.

3.2 Teaching vs. tutoring

For most of this paper we will be using the terms teaching and tutoring interchangeably, but some discussion of these terms is warranted. Teaching, often connoting an academic context, usually implies a structured environment, where the teacher has a predetermined sequence of instructional goals and activities typically for the student to participate in or observe. The activities are for a class of students and there is little opportunity to tailor the activities or the teacher's discourse to individual needs. Tutoring connotes a one-on-one situation with less structure and much more flexibility for attending to individual needs. Often tutoring actions are motivated by what the student says she is having trouble with. Intelligent tutoring systems try to combine both teaching and tutoring. They can use a structured lesson plan, which has high level goals determining an overall sequence of topics to be covered and the activities the student will engage in; they can make fine adjustments or refinements to these high level lesson plans according to individual needs; and they can introduce activities or dialogues on the fly to respond to the requests and instructional needs of individual students. Hence forth in this paper, we will use both "tutoring" and "teaching" to mean a combination of all of the above capabilities in computer or human instruction.

3.3 Rules for tutoring

Wenger (86) equates tutoring with "knowledge communication", and defines it as "the ability to cause and/or support the acquisition of one's knowledge by someone else via the channel of a restricted set of communication operations". ITS systems try to model the knowledge communication ability of successful human tutors. If we are to build ITS systems that emulate human performance, we must assume that human tutors behave according to some implicit rules concerning what actions to take (or not take) in each situation. In figure 1.1 we diagram how information describing the current tutoring situation is used by the tutoring rules and updated by tutoring actions. This schematic is not intended to have deep psychological validity, but to suggest a way of organizing the information flow and control mechanism of a computer tutor so as to account for behavior we observe in human tutoring. The parts shown in this schematic can be found explicitly or implicitly in all existing tutoring systems, and much of the terminology for the components is fairly well accepted in the field.

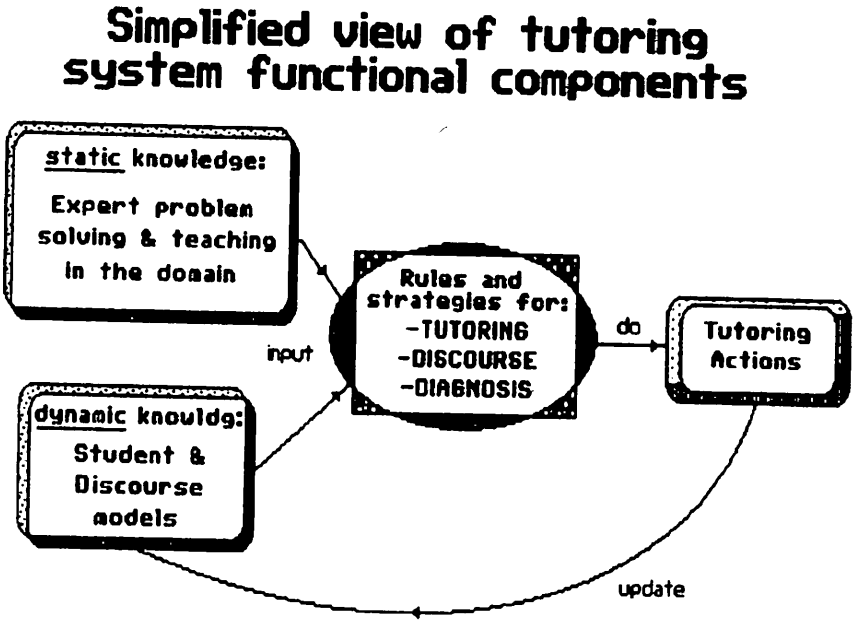


Figure 3.1: Simplified ITS functional components

The tutoring rules are typically of the form "IF <Situation> THEN <Action>", where the Situation specifies properties of the current tutoring context, and the Action specifies the appropriate action to take for that situation. (Such rules are also called production rules or productions.) Conceptually, these tutoring rules can be thought of as a mapping from all possible tutoring situations to all tutoring actions. The static information shown in figure 1.1 represents the tutor's permanent knowledge about problem solving and teaching in the domain being taught. The dynamic information represents constantly updated knowledge about the current situation, such as the tutor's model of what the student knows, the structure of the current discourse, and knowledge of the tutor's current instructional goals. The Situation (also called the condition or antecedent of the rule) is typically a complex combination of predicates, such as "if (the topic is just being introduced and (the student did well on the last topic or the student has 90 percent of the prerequisite concepts for this topic)) then ..." For now we will assume that the Action (also called the consequent of the rule) is the name of a single action (i.e. the name of a procedure—in later chapters we will intruduce more complicatd action patterns). The tutoring rules are scanned to choose one whose Situation part best matches the current state of the world, as represented in the static and and dynamic knowledge bases. (If more than one rule matches, a "conflict resolution strategy" is invoked to select one of them.) Then the Action named in that rule is executed. Some actions may be invisible to the student (we can call these "virtual actions"), such as deciding to change the focus of the discourse (without actually doing it yet), and others will be evident to the student, such as deciding exactly what the new focus will be, and making some utterance toward this new focus. In either case the action will directly or indirectly result in the dynamic information being changed or updated. Then the process is repeated, and typically a new action will be chosen since a different tutoring rule will be chosen to match the information in the updated data base.

3.4 The expert model

It is clear that a tutoring system for some domain, say physics, which can only ask questions about a domain and determine the correctness of the student's answer, is inferior to a system which "knows" physics, i.e. has some representation of physics concepts and problem solving procedures, and can itself solve physics problems. Representing expert knowledge with sufficient

detail and clarity to enable the tutor to be able to solve any problem which it gives to the student (and explain itself as it does so) is quite difficult in most domains. For some domains, the difficulties in building such an expert may outweigh its advantages, and it may be pragmatic to concentrate on the knowledge of how to teach the domain, rather than represent all of the knowledge to be taught. Below are some benefits of including an expert problem solving component:

- The student can ask unexpected questions about the domain, or ask the tutor to solve all or part of a problem.
- Common student errors or misconceptions can be modeled as deviations from the expert knowledge.
- The student's actions can be compared to what the expert would have done. Given a wrong response, the tutor can try to pinpoint which problem solving step or prerequisite knowledge that needs attention.
- An expert system can model expert problem solving behavior for the student to observe and interrogate.

A teacher is more than an expert in her domain (in fact, being an expert may not even be necessary in some cases), she is an expert at communicating the knowledge of the domain. In the next chapter we discuss the importance of including explanatory and pedagogical information in the domain model.

3.5 The student model

If tutoring is an attempt at helping the student "travel" from one knowledge state to another, the tutor must know the student's current "location." Consider an analogy of wanting to transport someone to Fort Worth. If you arrange for them to have a plane ticket from Chicago to Fort Worth, it is of no use if they currently reside in Miami. The analogy to instruction seems obvious, but traditional education (using the "learner as container" model) has often ignored this need to consider the initial knowledge state of students before teaching, and the need to keep abreast of the students' changing beliefs while teaching. Constructing an accurate and/or detailed student model is perhaps the most difficult and important computer tutoring system ability in terms of modeling human tutoring. In the next chapter we will outline the types of knowledge which are needed for a student model.

3.6 Knowledge acquisition

Designers of tutoring systems can have two complementary goals: to design innovative instructional environments/tools which would be impossible without the use of a computer, and to incorporate the knowledge of good human teachers into a system to achieve the level of instructional efficiency possible if the teacher could teach each student individually. Both goals are worthy foci of research. Designing innovative environments is more easily attainable given the state of AI technology. This paper addresses the second goal, modeling human tutoring, which tends to make stronger demands on AI technology. The main theoretical issues are representing knowledge, efficient use of the knowledge in controlling the system's actions, and transferring knowledge from the teacher or expert to a computer knowledge base. This last issue is called knowledge acquisition.

Many different kinds and levels of knowledge are used by a human tutor. The process of codifying this knowledge requires a more precise vocabulary for describing the many facets of tutoring than is now available. Examples of things for which we do not have accepted terminologies for are: types of knowledge, categories of tutoring actions, types of problems (problem solving problems), goals and functions of teaching, and types of examples (examples of concepts, example problem situations, etc.). Terminological precision is necessary because tutors make decisions based on the implicit categories that the properties of the tutoring situation fall into. For example, suppose a tutoring rule says "IF the information being taught is procedural knowledge, THEN ..." There is an implicit assumption that all information to be taught belongs in a "procedural" or a "non-procedural" category; or perhaps the categorization has several groups, such as "procedural", "factual", and "conceptual."

It is preferable from a theoretical perspective for these categorizations to indicate ontological commitments concerning the structure of knowledge and actions themselves. It is acceptable, however, to invent taxonomies that make it easier for the expert to describe his knowledge, and easier for the computer to access its knowledge base. At various points in this paper we will introduce terms for knowledge categories and system functional blocks—ideas for slicing the worlds of knowledge and action with the two goals above in mind. This will make the process of transferring the knowledge from expert teacher to computer knowledge base more efficient, and less ambiguous.

In chapters 4 thru 6 we intruduce a framework for representing declar-

ative and control knowledge which should make knowldge acquisition more tractable. In chapters 7 and 8 we outline a knowledge acquisition methodology for ITS.

Chapter 4

KNOWLEDGE REPRESENTATION

Many diverse types of knowledge are used in tutoring systems. In expert system research much effort is spent on methods and models for representing knowledge. Tutoring systems present even bigger challenges. Each unit of knowledge to be taught (or unit of expert knowledge) must be associated with knowledge of methods for explaining or teaching it, how it relates padagogically to other knowledge, common errors seen when teaching it, and so on. Ultimately, there will be much more of this extra knowledge (which we will call pedagogical knowledge, and could be referred to as meta-knowledge) than expert knowledge in a tutoring system. In addition to all this the tutor needs knowledge about how to tutor and how to diagnose the student.

In this chapter we present a general knowledge representation architecture for tutoring systems. Figure 4.1 illustrates the functional components of the tutor's data base. The static components, on the top row, can be considered permanent knowledge (although this may not technically be the case if such a system is implemented). The bottom row shows the dynamic components, which are updated during the tutoring session. The Decision Mechanism, which matches the tutoring rule conditions against the information in the other data base components (shown as input data), will be discussed in a later chapter. The Action mechanism will also be discussed later.

Below we outline these contents and implementation issues for each of the components of the tutor's data base.

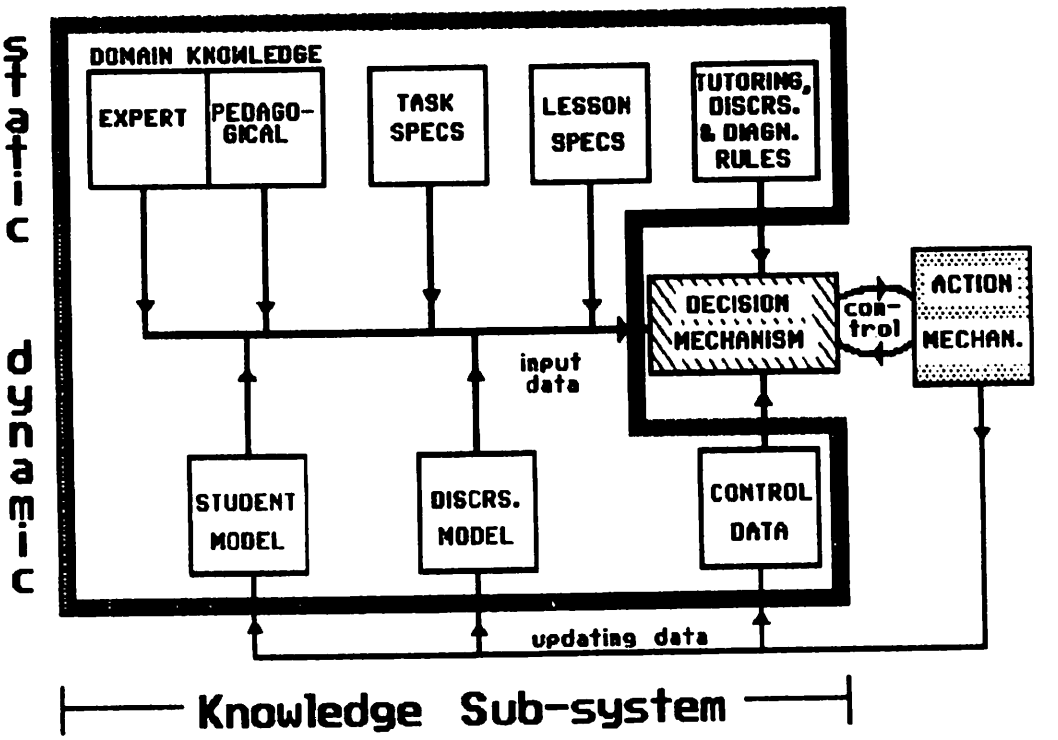


Figure 4.1: The Knowledge Sub-system

4.1 Representing domain knowledge

Some ITS systems have expert systems incorporated into their design. Tutoring can obviously be more effective if the tutor has an "expert's" knowledge of the domain, i.e. an ability to solve problems in the domain. The details of what kinds of domain knowledge needs to be represented, how it is structured, and how it is transferred from a human expert to the computer tutor's knowledge base depend on the peculiarities of the domain, but some general guidelines can be given. Below are some of the important considerations in designing a knowledge representation for a tutoring domain.

4.1.1 Types of knowledge

Different types of knowledge differ in the ways that they are learned and used. For example, memorized facts and problem solving skills differ greatly in how they are learned and used. The efficacy of our tutoring rules will in part depend on the precision and accuracy of our language for categorizing types of knowledge. This assumes that there is some (as yet mostly unknown) useful mapping from types of knowledge to methods for teaching each type (Clancey 86B and Chandrasekaran 85 have some relevant suggestions, but our analysis will be along different lines). A general classification of knowledge into facts, skills, and concepts is given below. Further refinements, motivated by pedagogical factors, will be suggested in a Chapter 8. Appendix B is an example of a more refined taxonomy, showing what such a classification may look like, but which is ad-hoc. (Also, see Murray 86 for more detailed discussions of the categories below in the context of the domain of elementary physics tutoring.)

Facts and skills. The Facts category roughly corresponds to our common conception of discrete factual knowledge. Factual knowledge includes definitions, propositions, properties of things, and relationships between things. Examples of factual knowledge are: the size and color of a ball, George Washington's birthday, the definition of chemical equilibrium, the relative heights of two houses, a mathematical formula, and whether or not two people are neighbors. One can see from this example that it may be useful to subdivide "facts" into sub-categories (but we will not do so).

The Skill category represents our knowledge of how to do things. Skills are procedures, algorithms, guidelines, heuristics, etc., which have a sequencing of physical and/or mental actions associated with them. Examples of

skills are: the long division algorithm, how to construct a geometry proof, and how to buy groceries.

Concepts—Nexus and Deep. Concepts represent groupings of densely interconnected knowledge of diverse types. Concepts can be visualized as nodes in a semantic network of knowledge, or entities representing structural relationships between other pieces of knowledge. The concept of gravity, for example, includes formulas (facts), experimental knowledge about the effect of gravity, skills for measuring the effect of gravity, knowledge of when to apply the associated facts and skills, and intuitive knowledge from experiencing the effects of gravity. For the purpose of representing knowledge in tutoring systems, we will make a distinction between two types of conceptual knowledge: Nexus Concepts, and Deep Concepts. Both Nexus and Deep Concepts are representations in the computer tutor of knowledge which we wish to teach to the student. Nexus concepts are represented completely in the tutor, so that we can say that the computer "understands" the meaning of the concept and can solve problems requiring its use. (By a computer "understanding" the meaning of a concept, I mean nothing more than that it can reason about it.) Deep concepts are those that we expect the student to eventually learn, but which we do not, and/or can not represent with enough computational precision to allow the tutor to solve the problems using the concept. (The term "Deep" here refers to complex, deep understanding in the human, so the computer actually has a "shallow" understanding of Deep Concepts. I'm serching for a better term than "Deep.")

Some of the conceptual knowledge which people use to solve problems is beyond the representational state of the art because of its complexity or lack of definition. Clancey (86, pg. 301) says: "The totality of what people know about a concept usually extends well beyond the schema that are pragmatically encoded in programs for solving limited problems." Even though a Deep concept can not be modeled effectively in an expert system, we may be able to represent sufficient aspects of it to be able to teach it, such as knowledge of how to teach it, examples of uses of the concept, and specifications of behaviors that give evidence that students knows the concept.

Both Nexus and Deep Concepts represent structural relationships between pieces of knowledge. Understanding a Nexus Concept is equivalent to understanding all (or perhaps most) of its components and their relationships which are represented in the computer. Understanding a Deep Concept requires knowing more than the components and relationships explicit in the

computer's rendition of the Deep Concept.

The distinction between these types of knowledge is not made in the literature, probably because there has been little or no intelligent tutoring of Deep concepts (but see Murray, Woolf, et. al. 86 for a recent attempt).

As an extreme example of a Deep concept, consider the concept of friendship. As part of some tutoring session we may want to know whether our student has this concept. We may be able to infer from some testing or behavior that the student has the concept, but we can not represent "friendship" precisely in a computer knowledge base. In contrast, we may be able to specify all aspects of what it means to understand the concept of a FOR loop in PASCAL programming. Such a concept would be a Nexus concept. Whether a concept is of the Nexus or Deep type is not strictly a function of the concept itself, but depends on whether we decide to (or can) capture its meaning with computational precision in our domain knowledge base.

Expert vs. pedagogical knowledge. The domain model is an expert system in the domain with copious information of explanatory or pedagogical nature. Builders of expert system have come to appreciate that there are many circumstances under which it is important that an expert systems not only arrive at an expert's solution, but that it do so in a way which simulates a human expert's problem solving process. It is also now an accepted maxim in expert system design that such systems be able to explain how and why they take the steps they take as they pursue a problem solution. For expert system builders this facilitates knowledge acquisition, debugging, and upgrading of the programs (see Clancey 86A). If computer tutors are to incorporate expert problem solvers, it is even more important that these programs emulate human behavior and can explain their solution steps (Clancey 82, and Brown, Burton, & deKleer 82).

We call "expert knowledge" the knowledge needed to solve a problem in the domain. "Pedagogical knowledge" is additional knowledge needed to explain, justify, or teach the expert knowledge. Knowledge about functionality, purpose, causality, exemplars, prerequisites, etc. are included in pedagogical knowledge, as well as annotations concerning learning difficulty, importance, salience, and suggestions on how best to teach pieces of knowledge. Pedagogical knowledge also includes common erroneous facts, buggy procedures, and misconceptions associated with pieces of knowledge. As mentioned above, a successful tutor is likely to need a high ratio of pedagogical to expert knowledge. In figure 4.1 the expert and pedagogical components are shown as conceptually separate, but both types of informa-

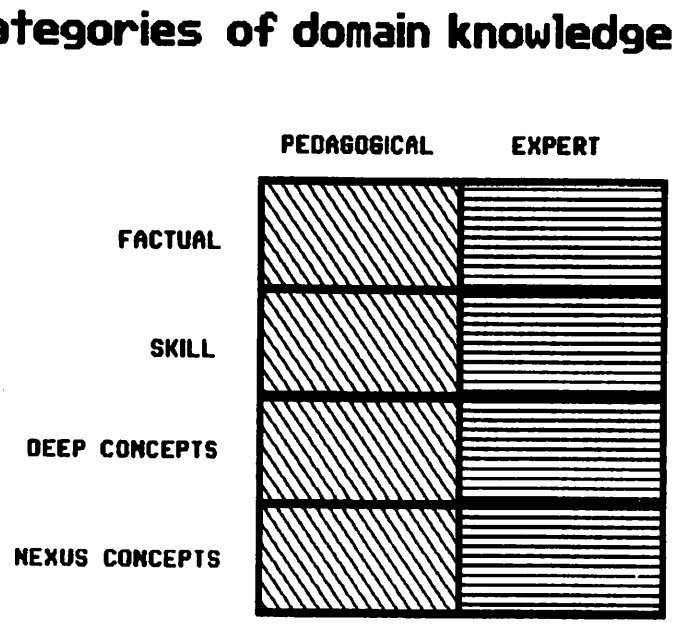


Figure 4.2: Categories of domain knowledge

tion will probably be interleaved in the same representational structure. In fact, it may not always be clear where the boundary between them is, since what constitutes information needed to solve the problem depends on one's definition of an expert problem solution (for example, does it include some explanatory information?). For more reading on "knowledge about knowledge," see Clancey 86B, Rissland 83, diSessa 85, Flavell 80, and Claxton 85.

The distinction between fact, skill, and conceptual knowledge is orthogonal to the distinction between expert and pedagogical knowledge. I.E. there will be some expert and pedagogical component for all knowledge, whether facts, skills, or concepts (see figure 4.2).

4.1.2 The internal structure of domain knowledge bases

When domain knowledge is analyzed for the purpose of teaching, an overall structure for that knowledge is often proposed. The structure is often hierarchical in nature. There are several dimensions over which one can construct such a structure, and there are different reasons for wanting to explicitly outline the structure of domain knowledge. For example, we may at some point in a tutoring session want to specify that "Newton's Third Law" be the next topic, and later another rule may refer to teaching "forces in static equilibrium situations", which is a component of teaching "Newton's Third Law". Here we need representations of different levels of generality. We may also need to represent prerequisite levels, or levels of causality. We have no suggestions here for synthesizing the various ways to structure knowledge and the various uses for these structures, but it seems worthwhile to mention several instances of domain knowledge analysis of structure. We do so below.

Chi et. al. (81) and others have shown that one difference between expert and novice approaches to physics is in how elements of problems are classified. Novices tend to classify according to surface features, and experts have something like a classification hierarchy indexed by non-surface properties. This suggests that both relevant and (to the expert) irrelevant factors must be represented in our expert knowledge base, so that non-optimal student behavior can be recognized and responded to intelligently.

Carbonell & Collins (Carbonell 70), in the SCHOLAR project, found it necessary to distinguish between "super-attribute", "super-part", and "super-concept" links in representing geography knowledge in a semantic network. Super-attributes link objects or concepts with important attributes. Super-concepts show "a kind of" links, such a Peru is a country. An example of a Super-part link is that Peru is a part of South America.

Stevens, Collins & Goldin (82) (the WHY system), and deKleer & Brown (80) have investigated the causal and enabling interconnections between facts and events. WHY's domain, meteorology, contains knowledge about the workings of a physical process. In teaching any domain dealing with explaining the science behind physical mechanisms we may expect that causal interconnections will be important. White & Frederiksen (86), in their electronics tutor, also teach causal relationships between physical mechanisms. To contrast, Clancey's GUIDON (82) shows how information accumulated during diagnostic problem solving can be represented in an evidential network, with each new bit of information pointing back to the piece of infor-

mation which supported its instantiation. Causal analysis of the system, i.e. anatomical processes, is not taken into consideration.

Burton (82), in the BUGGY system, an analysis of subtraction skills, found that the skills could be organized in a "skill lattice": a hierarchical structure representing how subskills use, subsume, or mask other subskills. Barr, Beard & Atkinson, in the BIP tutor for teaching BASIC programming, use a "Curriculum Information Network" to relate tasks to domain skills. The network contained information about which skills were prerequisites of others (dependencies), and contained kind-of and component-of relationships similar to the super-concept and super-part relationships in Scholar.

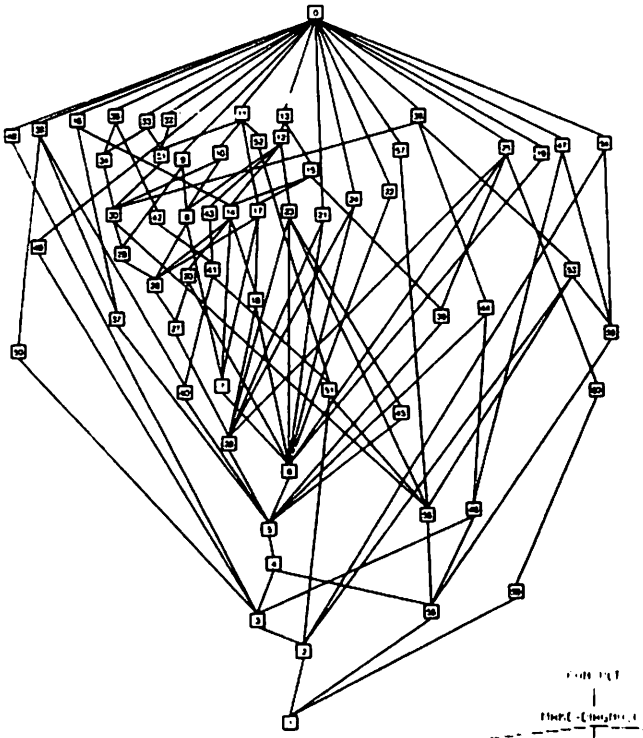
Miller (82), in the Spade-0 system for teaching LOGO programming skills, shows a hierarchical taxonomy for steps in planning or debugging computer programs. Anderson's LISP tutor (Anderson & Reiser 85) and Geneserth's (82) Macsyma Advisor encode problem solving knowledge (or "planning" knowledge) in subgoal structures.

Goldstein's (82) WURSOR tutor represents knowledge as procedural rules which are interrelated by the relationships analogy, refinement, generalization/specialization, and correction/error. It also incorporates phases of successive refinement of domain knowledge, where succeeding phased precludes previous ones, approaching an expert's knowledge.

The above research indicates that the hierarchical or overall classification structure of knowledge is used in various ways. Knowledge of subconcepts, subskills, and prerequisites is used in diagnosing misconceptions and prescribing remedial activities. The same knowledge can be used to direct the presentation of new knowledge in a tutor's default teaching path. Hierarchically structured knowledge allows us to refer to groupings of knowledge in tutoring rules and in tutoring session plans. For example, we can specify to "teach about energy conservation", and the system can infer from the hierarchy what sub-concepts must be taught. It is likely that several of these organizational structures will have to be represented simultaneously (in parallel) in an ITS knowledge base. Utilizing different organizational structures during a tutoring session allows knowledge to be connected in different ways, or approached from different angles.

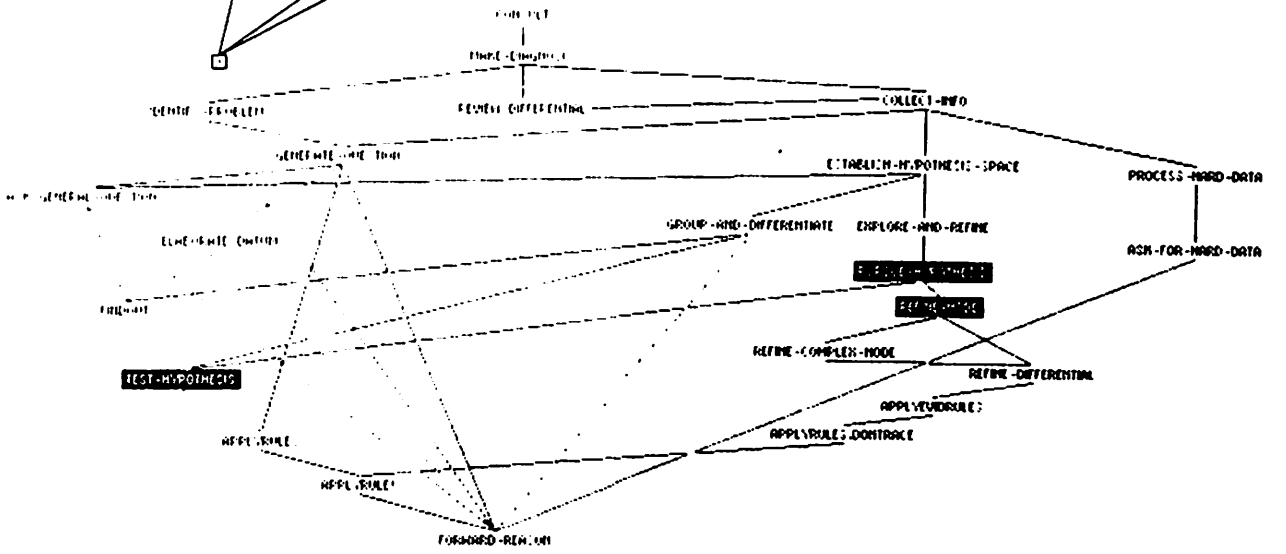
4.1.3 Uses for domain knowledge

We will call each piece of knowledge that we wish to teach a "knowledge unit", or KU (from Servi 86). As indicated in the above discussion on



Skill Lattice for subtraction
(Buggy system, Burton 82)

0 Correct skills
1 No subtraction skills
2 Subtract 0 from a number whose right-most digit is 0
3 Subtract 0 from a number
4 Subtract columns which are n-n or n-0



Lattice for Diagnostic Task Invocation
(Neomycin system, Clancey 86)

Figure 4.3: Example knowledge base structures

knowledge structures, some KU's may subsume lower level KU's. KU's may represent a single fact, a single rule, a single step in a procedure, or complicated concept or skill. The most important use of KU's is in representing expert knowledge. Expert knowledge is emulated in order to show the student the knowledge, to allow a simulation to exhibit that knowledge, or to check a student's behavior against what an expert would do.

KU's can have several attributes associated with them besides those which allow expert emulation. Examples of pedagogical information that can be associated with each KU are: how to give examples of itself (see Rissland, Valcarce, & Ashley 84, Lenat 79, and Murray et. al. 86); describing its purpose, i.e. why know it or why use it; explaining when it should be used, i.e. under what problem solving context; the ability to quiz the student to determine if the student knows it; listing its prerequisite KU's; listing the assumptions implicit in the KU; and revealing the source of the KU knowledge, or an explanation/justification of how it was derived or inferred.

4.1.4 Object-oriented representations

We are using an object-oriented representation of the domain knowledge, which affords several desirable characteristics (see Bonar 86, and Stefik & Bobrow 85). In object oriented programming, data and procedures are combined into entities called "objects." An object is a data type combined with procedures (or operators) called "methods" specific to that data type. In this programming paradigm, one "sends a message" to the object, rather than calling a procedure. In our architecture, Knowledge Units, Examples, Lesson Specifications, etc., are all types of objects. Each type of object has methods associated with it for the operations specific to it. For instance, different types of knowledge (.i.e. sub-types of the Knowledge Unit type) may have different procedures for evaluating misconceptions related to them. One advantage of this formalism is "data-abstraction," meaning that the use of a procedure associated with an object does not require that the user know anything about the internal representation of the object (see Bonar, Cunningham, & Schultz 86). Without object oriented programming we may have to specify different diagnostic procedures (Diagnose-1, Diagnose-2, etc.) for each type of knowledge. Using methods, we send the message "(SEND knowledge-unit DIAGNOSE)", and the object receiving the message (the knowledge unit) will have its own diagnostic procedure executed. KU's can have methods such as "teach-me", "describe-me," "test-for-me," and "tell-prerequisites." A tutoring rule can specify that an example of the current

concept be given to the student, and the fact that different types of concepts must use different procedures to give examples of themselves is taken care of automatically.

Object oriented programming environments also have a mechanism for hierarchical relationships, facilitates classification grouping and property inheritance. For example, the general class "KU" may have a method for teaching its prerequisites. KU's may be in a hierarchy of categories of knowldge, such as Facts, Skils, and Concepts. These categories will, by default, inherit the "teach-prerequisites" method from the KU type. If, for example, the way Factual knowledge teaches its prerequistes is different than the norm (the default), we can define a new "teach-prerequisites" method associted with Facts.

4.1.5 Mental models and qualitative reasoning

There has been much focus of late on qualitative reasoning processes and understanding the internal mental representations people have of systems (Gentner & Stevens Eds. 83). It appears that expert problem solvers in many domains are able to picture, "run", or "envision" the workings of a system using imagination alone. Given the initial conditions of the system, the problem solver can "run" the model and make predictions about its behavior. Furthermore, poor problem solving abilities can sometimes be attributed to incorrect mental models (White & Frederiksen 86). Mental models deal almost exclusively with qualitative knowledge. In fact, it was research on the importance of qualitative reasoning which lead to the current interest in mental models. Several studies have tried to explain why students who could solve routine quantitative problems and do well on academic exams in math and science course have much difficulty with non-routine problems and problems which required assumptions and estimations (Lockhead & Clement, Eds. 86). It was found that deficiencies in qualitative understanding of these domains was a factor.

In tutoring systems, especially in technical domains, the importance of qualitative reasoning and qualitative questions should be stressed. Unfortunately, given the present state of the art, it is difficult to build computer expert qualitative problem solvers (but see deKleer and Brown 82, Forbus 84). This is an instance where the "Deep concepts" mentioned above may be needed. We want to teach qualitative knowledge, therefore we would like to be able to refer to instances of this knowledge (KU's) in our student model and our tutoring rules. The resulting representations for these Deep

concepts merely "stand for" the concepts, and can not be used in an expert system to solve qualitative problems. Several existing tutoring systems emphasize qualitative knowledge of physical systems and attempt to help students build correct and robust mental models (Burton & Brown's (82) SOPHIE, and Holland's STEAMER).

4.1.6 Novice domain models and moving targets

Two related issues will be discussed here. One is that an expert's representation of the knowledge in a domain may not be cognitively accessible by the novice in the domain. The expert has arrived at an efficient and powerful structuring for the many pieces of knowledge that comprise his expertise. It may not be possible to teach this expertise directly in this "compiled" form; attempts at teaching it may all be "over the student's head". In such a case it is necessary to incorporate knowledge of how the expert built up this knowledge from a novice state. Intermediate "non-expert" representations of the domain knowledge may be needed.

The second, related, issue is that the student model is an evolving set of knowledge (and mis-knowledge) pieces. In order to diagnose student knowledge we may need to represent successive levels of evolution from novice to expert understanding. Goldstein's (82) genetic graph is an attempt at representing the evolutionary connections between KU's. His Wumpus tutor can focus its remedial interventions around an appropriate level of expertise. Burton & Brown's WEST research (82) mentions adjusting the level of play of a computer coach if the student loses consistently.

White and Frederiksen (86) discuss "model progression" in student's understanding, and teaches successively more refined models of electric circuits. Anderson (83), in his ACT theory of cognition, proposes mechanisms for "knowledge compilation", and suggests designing tutors which can adjust their model of the student when separate pieces of knowledge are compiled into efficient units.

4.2 Task and diagnosis specifications

In our proposed architecture we separate knowledge about specific tasks from general (expert and pedagogical) domain knowledge because it is used and represented differently. Tasks involve applications of domain knowledge to specific situations.

4.2.1 Tasks

Tasks are problems for the student to solve, presentations with examples and questions to be given to the student, or other activities for the student to observe or engage in. The tutor may give a student a task as a means of teaching something, or as a means of diagnosing the students knowledge, or both. Task specifications describe how the task is to be set up, and how to interpret the student's behavior while she is engaged in the task. The simplest example of a task specification is a question (a block of text) and a description of correct answers to the question. An example of a more complicated task specification is initial conditions for setting up an electronic circuit simulation, complete with methods for analyzing the student's behavior relative to an expert's behavior. Task specifications can be arbitrarily underconstrained, expecting many details to come from the current tutoring context. For example, a task specification may set up a medical diagnostic case such as in GUIDON, and look at the current tutoring context for what disease the simulated diagnosis will diagnose.

4.2.2 Diagnostic specifications

The task specification can also contain information about how student behavior gives evidence for (or against) the existence of concepts or misconceptions in the student model. We have noticed, in designing tasks and rules for tutoring systems, that knowledge of how to diagnose a student's behavior is most often specific to a particular task (as opposed to a general diagnostic strategy). As a simple example: if the student answers "a" then increase the confidence that she has concept X, and if she says "b" then increase the confidence that she has misconception Y, or if she measures item C more than 3 times, she is missing concept Z.

For diagnostic information that are general to many tasks would be stored in the diagnostic tutroing rules (as opposed to in the diagnostic specification of a task).

Recall that diagnostic information can also be stored as pedagogical information in the domain model, provided it refers to a specific piece of knowledge. For example, associated with a concept we might find a pointer to a task that tests for that concept, or a more general description of behaviors that give evidence for (or against) that concept. So a task may point to a concept, indicating that the task requires knowledge of that concept, and a concept may point to a task, indicating how certain behavior in the task is relevant to that concept.

4.2.3 Simulation modules

We will assume that simulation modules and interface modules that can interpret the student's raw behavior (such as mouse clicks, text typed in, and operations performed to manipulate a simulation environment) exist outside the tutoring system architecture described here. The task specifications send information to these modules in a language understood by the specific module, and the module communicates with the tutor in a language designed for exchanging information with the student model, the domain model, and the other parts of the tutor. Note that this architecture assumes that a simulation is a component of the tutor that was designed after or along with the pedagogical aspects of the tutor. This is to contrast with designing a simulation first and trying to tuck the pieces of a tutor in around it.

4.2.4 Tasks as specific actions and instantiations of knowledge units

One can think of the description of the task as an instantiation of knowledge in the domain expert knowledge base. A task description may say that Frank throws a 5 lb. ball into the air with an initial velocity of 10 m/sec, and then ask several questions about this situation. A physics expert knowledge base knows about projectiles, gravity, and how to answer questions such as the one given, but it does not know about any particular situations, such as the one with Frank and his ball. Particular problem situations are specified in the task specification component. Our architecture also has components called Tutoring Actions (discussed in a later chapter). Tasks can also be thought of as instantiations of tutoring actions, or as specifications of general tutoring actions. For instance, a tutoring action may be "give an example," or "start up the simulation," and a corresponding task would specify the specific example given or parameters for the simulation.

4.3 Tutoring rules

Tutoring rules encode general knowledge about tutoring, communication (discourse), and student diagnosis. They incorporate information about what to do (for example: give an extreme case example, diagnose a misconception, etc.), when to do it (for example: whether to ask the student a question now, or not interrupt the student yet), and how to do it (for example: what form an utterance will take). For the time being, we can

picture the tutoring rules as production rules, i.e. as "IF...THEN" statements, where the IF part specifies some state of the world, and the THEN part specifies some action to take if the current context matches with the IF part. (Note: In chapter 6 we introduce some modifications to this basic rule formalism.)

Tutoring rules may contain information about how to teach or diagnose knowledge in a domain, but, in contrast to the pedagogical knowledge in the domain KB (knowledge base) and the diagnostic knowledge in the task KB, tutoring rules are fairly general. For instance, a tutoring rule may describe a state of the discourse where "giving an example of the correct concept" is appropriate. The pedagogical information in the domain model would know what the examples are for each concept.

In most ITS's the tutoring rules are not explicit, or are reported in vague or general terms. As such, they are more like general tactics than computationally precise rules. In this paper we advocate incorporating tutoring rules explicitly as production rules (or related structures as discussed in Chapter 6), which can be examined and modified by the designer. (See Woolf 84 for a theory of organizing the set of tutoring rules in an ITS in a network structure.) Tutoring rules will be discussed in more detail in Chapter 6.

4.4 The student model

Here we will discuss the types of knowledge needed to model the student's beliefs. For discussions on the complex tasks of diagnosing and modeling the student, see Clancey 86B, Anderson, Farrell, & Sauers 84, VanLehn 86, and Burton 82.

The student model usually contains an "overlay" (Goldstein 82) of the expert components of the domain model, i.e. a copy of (or pointer to) the KU's of the domain. These pointers can be annotated with information about the estimated strength of that knowledge in the student, and the uncertainty or reliability of the system's belief that the student has that knowledge. An overlay student model might contain information such as how often the KU was used, how often the student asked for help concerning that KU, whether the major prerequisites of that KU are met, etc. Note that some of this information is technically not a model of the student's knowledge, but evidential data supporting components of the student model.

The relationship between expert KU's and mis-knowledge units in the student model depends on the type of knowledge (whether fact, concept,

skill, etc.). For example, facts which are properties of things can be missing from the student model, or can exist with the wrong value. Representations of erroneous Deep Concepts, called misconceptions, may be pre-defined in the tutor, based on research on common misconceptions in a domain. Skill or procedural knowledge has several mis-skill variations. A skill can be applied correctly but in the wrong context, or it can be missing when needed, or it can be applied when needed but not executed correctly.

Much was said in previous sections concerning the student model and student diagnosis. To summarize: the student model may be a moving target, evolving as the session progresses; sub-expert levels of knowledge and "genetic links" may be needed to model student's knowledge; diagnostic specifications may be associated with certain tasks, and diagnostic actions may be pointed to from individual KU's; hierarchical organization of expert knowledge can indicate possible areas of weak subconcepts or subskills.

As mentioned earlier, expert knowledge may in some cases be too advanced for students to assimilate, and a representaiton of "novice" knowldge may be needed. Novice knowledge, though not as general or precise as expert knowledge, is more accessible. It can be used as an instructional stepping stone, or for more precision in diagnosis (i.e. an overlay of the novice knowldge units along with the expert KUs).

One advantage of representing the student model as a subset of the expert knowledge (along with deviations from expert knowledge) is that the student model can be "run" just like the expert model can. In so doing we can make predictions, using our current model of the student, about how she will behave given some task. These predictions can be used to generate appropriate task parameters, or to check the validity of the student model by comparing the prediction with the actual student response.

The student model can also contain more global descriptors of the student's cognitive or affective state (i.e. not tied to specific KU's in the expert model). For example: learning preferences, methods of tutoring that worked well for this student, general confidence level, etc.

4.5 The discourse model

The discourse model is a dynamic representation of the current state of the tutorial discourse, including information about what actions have transpired during this (and perhaps previous) tutoring sessions. Compared to the student model, which is a representation of what the student *knows*, the dis-

course model represents what the student and the tutor have *done*. Perhaps more important than a historical listing of significant events, are summary or abstracted properties such as the total number of wrong answers in the session, the number of changes in focus, how long ago the student asked for help, etc. This summary information can be updated continuously, or the history can be scanned backwards from the present to infer this information when it is needed.

The discourse model remembers the context in which the actions it records, or the behaviors observed, occur. For example, in Shute & Bonar (86) behaviors are recorded for eventual matching with test conditions "are all the simulation variables selected for recording in the student's notebook only those actively changing?"

The vast majority of ITS's incorporate what we are calling the discourse model into the student model, or have have no discourse model information at all.

4.6 The control data base

The control data base holds information, mainly for "bookkeeping" purposes, which is used by the decision mechanism, and which is not approprite for the Student or Discourse Models. Among other things it allows for goal driven dialogues, nested sub-goals, and schedueling of tutorial activities. It will be discussed further in the chapter 6.

In summary, we have indicated many ways to slice up the knowledge needed by an intelligent tutor. Since this framework has not yet been fully implemented, it is only a first pass suggestion. Ultimately, there are two criteria for deciding whether to make any given slice through the knowledge, i.e. whether to define a given distinction between types of knowledge. The first criterion is that the distinction must be referred to in the tutoring rules (or in forseen tutoring rules). That is, each new category is introduced because the designer encounters a tutoring tactic which distinguishes some characteristics of the knowledge (i.e. a distinction should not be incorporated just because it is philosophically appealing to think of the knowledge as falling into certain categories). For example, assume we have a category of knowledge called "Rules." It is observed that an expert teacher interrupts a student when he makes a mistake using "memorized" rules but not when the student makes mistakes on "heuristic" rules. In order to incorporate

this new rule into a system, we should subdivide the "Rules" category into "Memorized Rules" and "Heuristic Rules." The second criterion is that the distinctions made are defined precisely enough to be useful. for example, we will need a clear definition of the difference between the Memorized and Heuristic rules. As onther example, suppose that we find that in some domains it is practically impossible to judge whether a piece of knowledge is a fact or concept. In such a case it does no good to have a place to record this distinction in the knowledge base, or to refer to this distinction in tutoring rules.

4.7 The lesson specification

The lesson specification is a top level lesson plan, and/or a specification of the high level learning goals for the tutoring session. In contrast with the tutoring rules, which control the moment to moment local decisions of what to do next during a tutoring session, the lesson spec outlines the sequence of main sub- topics, activities, and tutoring styles to be used for a given session or topic unit. The Lesson Spec will be discussed in more detail in the next chapter.

Chapter 5

LESSON–LEVEL INSTRUCTIONAL SPECIFICATION

5.1 Global and local planning in ITS's

The sequence of information presented by a computer tutor can be thought of as being controlled at two levels: a global level and a local level. The global level is the Lesson Specification, where the instructional designer specifies curriculum level lesson structures and topics to be learned. At the local level, decisions are made on the fly by the tutoring rules, regarding such things as what examples to give, whether to present prerequisite materials, etc.

The Lesson Spec includes a top level specification of the instructional goals. Tutoring systems which are controlled only by local decisions made by small grain size (i.e. very specific) rules may wander in their presentation of material because the tutor is "near-sighted," i.e. has no overall goal or plan. A high level (i.e. lesson level) goal specification enables a more focused and consistent sequencing of main topics. The Lesson Spec not only specifies the general content of units of instruction, but also an instructional environment and pedagogical style, as we will outline below. Having such information in the Lesson Specs enables multiple instructional environments and teaching styles in a single tutoring session.

5.2 Learning and teaching at different levels

One factor that adds to the difficulty of computationalizing the human tutoring process is that it is often not simply a one directional path through a knowledge space; rather, the tutor switches focus between local and global issues, giving perspective and then coming back to the details. Also, effective learning in many domains requires some kind of spiraling through the knowledge space, going over the same material several times with different perspectives. It is difficult to evaluate the success of many existing computer tutors because they do not allow for various modes of instruction and multiple passes though material. In most domains one can not expect significant learning to take place with one pass. For example, doing effective problem solving requires knowledge of basic factual knowledge, yet understanding of the factual knowledge can only be obtained in a problem solving environment (or some other rich or realistic context)–a chicken-egg problem. The solution often applied in the classroom or text book is a layered approach in which topics are gone over in several passes, with increasing detail or increasing focus on application.

In a traditional science course a student may first skim the chapter or read the introduction to get an overview. Then she reads the chapter and/or goes to a lecture session to get introduced to the details of the new concepts. Then she engages in homework, labs, and discussion sessions to construct a more intimate grasp of the material, including interrelations between concepts and meta-knowledge about the material. Finally the student tries to put the instructional unit all together and synthesize it with previous material by writing up lab experiments and studying for exams. Computer tutors should likewise allow for several passes while learning a given topic, and have different leaning environments available for the different passes. Determining focus shifts at the global level can be done in the Lesson Spec. Each pass through the material may require a different learning environment and different tasks, and a different set of tutoring rules will be appropriate. Note that the more intelligent our tutors become, the more they will be able to infer things at the curriculum level. The extreme case is a system which knows only the domain knowledge and the pedagogical characteristics of that knowledge, and can decide on its own what the sequence of topics and the appropriate tutoring modes are. However, for the near future we should design systems that allow instructional specialists in the domain to specify some of the global goal structure and tutoring modes.

5.3 Tutoring Modes

One of the popular issues in ICAI research centers around how much control the student should have over his learning environment, and, conversely, how much control the tutor should have (see Papert 80 and Pea 83). The optimal degree of control is a function of many things, including the learning task and the type of "pass" through the learning material, i.e. whether the knowledge is being introduced or used to solve problems. The real question is not which strategies or instructional paradigm to use, but when to use each. This is especially true in trying to design a computer tutoring system which will have the flexibility to accommodate many domains and the instructional styles of many instructional experts. A spectrum of student control from exclusive student control to no student control is outlined below:

- Passive environments which embody the knowledge or properties of the domain; the student has full control in experimentation and play within some environment.
- "Coaching" environments which observe the student's actions, and interrupt only when a mistake is made or a significant learning opportunity is missed.
- "Mixed initiative" environments where the tutor has a definite agenda concerning what is to be learned, but the student has the ability to interact to gather information or change the direction of presentation. (This may be similar to "Socratic" teaching, but interpretations of the Socratic method vary in the literature, so we don't use it here.)
- "Pedantic" environments in which the student has little control, and the tutor presents material in a fairly lock-step fashion, similar to a lecture or a programmed learning environment.

Styles of learning environments such as those outlined above vary along several dimensions, including: 1. the amount of information needed about the student (i.e. the complexity of the student model and diagnosis procedure), 2. the degree of control which the tutor assumes, and 3. the types of facilities which are available to the student for gathering information, changing the direction of the instruction, and changing the parameters or components of a simulation environment.

We propose a data structure called a "Tutoring Mode" (or just Mode) which specifies the pedagogical characteristics of the learning environment

by specifying a set of tutoring rules to be invoked and a set of facilities or "Tools" available to the student. The set of tutoring rules which comprises any mode will usually overlap with the set of rules in other modes, i.e. some rules will be applicable to several modes. Examples of what is meant by "tools" above are commands such as glossary, help, review, new topic, and facilities such as meters, constructors, notebooks, etc. (and see Shute & Bonar 86, and Cook 83). Following is a possible set of tutoring modes, with their intended purposes and functions, which an ITS system may have available to it. They are listed in an order suggesting the "several passes" approach to learning mentioned above:

Summary/preview/or review

Highlight the learning objectives, usually in a context relating it to other topic areas. The student can ask for list of prerequisites and examples of the learning goals. The student can request questions which test competence of the learning objectives. This mode could be used as an introduction or conclusion/summary to an instructional unit, or it could be used as a review.

Pedantic

Here the tutor "lectures" on a topic, or explains, or creates a situation and solves it, giving explanations as it goes (as in the expert trouble shooter of SOPHIE II (Brown, Burton, & deKleer 82)). The student watches, and can interrupt to ask for a more detailed explanation of something, or to ask a specific question about a value, a glossary item, or a "what if" question, altering one of the situation parameters. This mode, similar to what is usually termed a "tutorial", is a bottom up teaching style; new information is introduced and built upon in a systematic way (compare with "mixed initiative" below, which is top down). Some uses of the Socratic teaching method is pedantic, since the student's learning is directed by questions asked by the tutor.

Passive learning environment

The student is allowed to experiment and play without interruption in a rich environment which embodies or simulates the objects, relationships, rules, and/or structure of the domain. No intentional feedback or diagnosis is given. Students can learn from their mistakes provided they can recognize mistakes and infer the cause of the mistake. Since there is no guidance, the environment must be motivationally stimulating. The student can ask questions about values or glossary

information. If the domain is highly information oriented, the student can browse through a data base. Examples of passive learning environments are the LOGO programming environment (Papert 80), and Schwarzt's Geometric Suppopser. Simulations without student feedback are also passive learning environments.

Mixed initiative teaching

The student is allowed to construct his own situation, or he is provided with one. His task is to solve a problem. The tutor has a quasi-rigid specification of what it wants the student to learn, though it may be deciding dynamically how best to tutor this information. The tutor monitors each step closely and provides feedback, diagnostic questions, or hints where appropriate (care must be taken not to interrupt too often). The tutor tries to correct misconceptions. The student is allowed to access a glossary, a calculator, an information "browser", and other tools. She is allowed to ask "what if" questions in a limited way (if they are not irrelevant to the current topic). This mode represents "top down" tutoring. Familiarity (though not mastery) with the basic facts, skills, and concepts is assumed. The tutor starts by presenting a non- trivial problem solving situation, and prerequisite knowledge is taught only as it is needed.

Coaching mode

This mode is appropriate for "informal learning environments" (Brown, Burton, & deKleer 82), such as games and on- the-job training. Burton and Brown also call it a "reactive" learning environment, because the student learns from her mistakes. The tutor analyses student work in the background, interrupting only the student makes a sufficiently serious mistake or misses a significant learning opportunity. Student diagnosis, if it is done, is difficult in coaching modes, since the tutor must be an expert problem solver in the domain, and recognize the student's knowledge, misconceptions, and plans without interrogating her. Coaching has been implemented with (Burton & Brown 82) and without (Brown, Burton, & deKleer 82) student modeling.

Test-out

The student is given a problem and is given no assistance. He may have limited access to tools such as a glossary or equation solver. Moving on to the next topic requires success with the test-out.

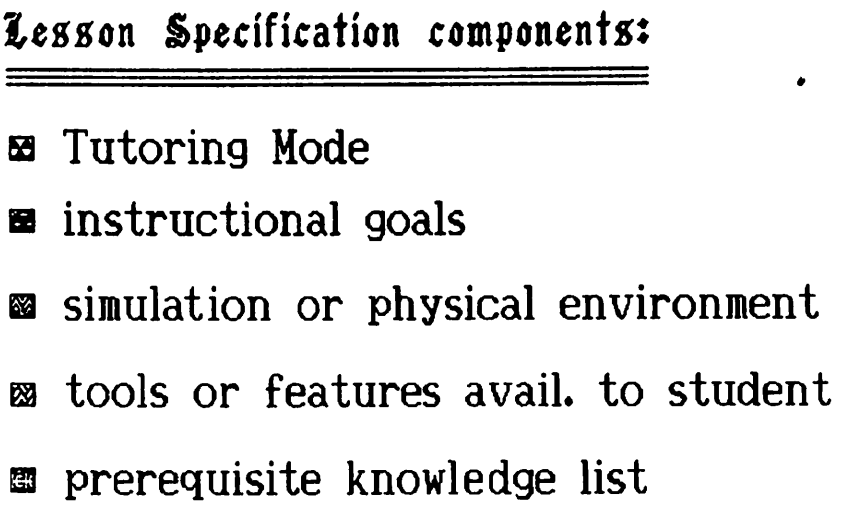


Figure 5.1: Lesson specification components

5.4 The Lesson Specification

The Lesson Spec specifies a sequence of main Topic Units (or phases of teaching a topic). Each topic unit contains the following information (as shown in figure 5):

- Specification of the "physical environment": This may set up a simulation, or a particular arrangement of screen windows, etc.
- Specification of the tools or features available to the student: measuring tools, help functions, information access functions, graphics tools. These tools partially determine the degree of control the student has over his environment.
- Prerequisites for this topic: Assumed knowledge and perhaps a pretest.

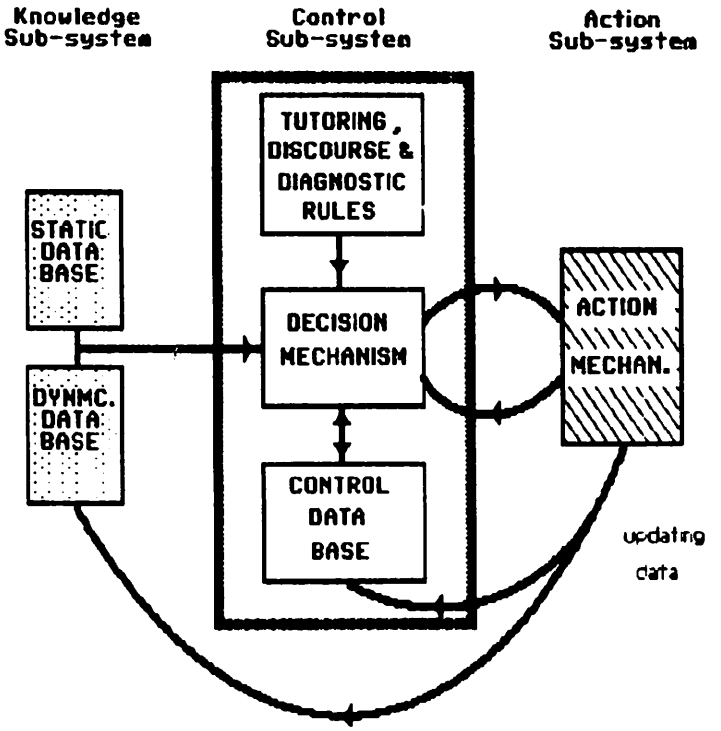


Figure 6.1: The flow of information and control

bases. Finally, control is returned to the Control Sub-system so that it can select the next action. Thus control alternates back and forth between the Control and Action Sub- systems.

Below we will discuss how the Control Sub-system is an embellishment on standard rule based control architectures (which we will also call "simple" or "vanilla" rule based architectures). Then we will expand upon what happens in the Action Sub-system. (Note: The next two sections can be skipped by those familiar with AI production system architectures.)

6.2 Vanilla rule based control structures

Control in a rule based system (also called a "production system") is accomplished by matching "IF..THEN" rules to a data base of information. The rules have the form "IF Situation THEN Action", where the Situation

Chapter 6

A GENERAL CONTROL ARCHITECTURE

6.1 Program control issues

The main issues in program control are 1. how information is stored (represented), retrieved, and passed between component parts of the program, and 2. how the sequence of actions the program will take is determined. Previous chapters have outlined structure for representing the knowledge needed in an ITS system. In this chapter we will introduce a control architecture which directs the actions that the tutor takes based on the contents of the these knowledge bases. This architecture is an attempt to systematize the actions and functions of existing tutoring systems.

The proposed architecture is a rule based system with several embellishments. Figure 6.1 shows an abstract view of the flow of information and control to and from a "Decision Mechanism." Note that figure 6.1 includes all the parts in figure 4.1 (the data bases) in a different conceptual structure, emphasizing control rather than knowledge representation issues. (In figure 6.1 the tutoring rules and control data base are depicted separately from the rest of the static and dynamic data bases in figure 4.1.) The tutoring rules and the control data base are considered a part of the "Control Subsystem." The Control Sub-system utilizes information from the "Knowledge Sub- system" in determining which tutoring action to execute next. The Control Sub-system then tells the "Action Sub-system" which action should be executed. The Action Sub-system then executes the specified tutoring action, along with other actions which maintain the environment and data

base becomes large. Vanilla rule based systems with large numbers of rules (or even a moderate number, say 50) are, for reasons given below, hard to manage. These disadvantages are discussed to motivate the embellishments to the simple rule architecture, which we will present later.

6.3.2 Confounding of different types of control information

Because of the simplicity of the control mechanism in vanilla rule systems, different types of control information are treated identically. In complex computer programs this can lead to an undifferentiated and baroque conglomeration of rules.

As an example, suppose that in some tutoring system an analysis of a student's answer can lead to suspecting a student misconception. The tutor checks the student model for previous evidence of that misconception in an effort to confirm its suspicion. If this hypothetical tutor does not find a "high" level of support it increases its level of suspicion, but otherwise performs no visible action. Let's say that we, as system designers, want to add a new rule. If there is "medium" level of evidence, the tutor should ask the student directly if he thinks he believes the misconception. We want the new rule to say "IF there is medium evidence for the suspected misconception, THEN ask the student about it directly". However, to ensure that the rule fires only when we want it to, the rule may have to say "IF the context is one of trying to verify a suspected misconception, AND we just checked the student model, AND there was medium evidence for the suspected misconception, THEN ask the student directly if he thinks he believes the misconception." There will be many rules which are intended to fire only in specific contexts, yet they will be checked for a match after every tutoring action, along with the entire rules base. In such cases antecedents like the first two "ANDs" in the above example rule have to be put in the rules to ensure that the system exhibits the type of structured program flow we get with programming languages, where fixed sequences of actions are easily specified. It is desirable to separate such "control knowledge," from the knowledge about how to tutor (see Lesser 84, Clancey 86A).

6.3.3 "Side effects" are used to control program flow

The action parts of rules are either propositions, as in "IF X is true THEN Y is true", or actions, as in "IF X is true THEN do D". Concerns for clarity would dictate that all the rule actions refer to domain related propositions or actions. But often there are rules, or consequences of asserting the rules,

usually contains several conditions (also called "antecedents") which must all be true for the rule to apply. The Decision Mechanism in a simple rule based system has a "matcher" that checks all of the rules and determines which ones apply (i.e. it looks for a match between the rule's antecedents and the information in the data base). It then chooses one of the rules that was successfully matched, and executes its Action (or "fires"). An algorithm for choosing which of several equally matching candidates to fire, called a "conflict resolution strategy," is needed. The optimal algorithm varies among applications. Examples are: choosing the first rule that was found to match, and choosing the rules whose antecedents are most specific (for example: "if its a dog" is more specific than "if its an animal"). After the rule fires, the state of the world has changed, a fact usually reflected by a modification to the data base. The matching procedure starts again, and this time a different rule should fire since the change in the data base causes different rules to match it. Simply stated, the match-fire process is repeated until the program terminates.

6.3 Advantages and limitations of vanilla rule based control

6.3.1 Advantages

Because production rules have a simple, standard (or "canonical") form, rule based systems are modular, flexible, easily modified, and their actions are easily traced. These are desirable characteristics for our general tutoring architecture, since a standard rule format will allow rules used in a tutoring system to be easily modified and easily transported to other systems (see Davis, Buchanan, & Shortlife 77). As mentioned previously, it is essential that we can easily experiment with and modify tutoring rules.

Rule based systems also have an "opportunistic" flavor to them, in that the flow of control is not rigid, as it is in programs where control is determined by procedures calling each other and passing parameters. Production systems match the rule base with knowledge about the current circumstance after every action. The system continually has its entire repertoire of rules available to it, so it is always ready to respond to changing situations. In tutoring systems this opportunism enables the tutor to react to unpredictable student behaviors.

Vanilla rule based control architectures have several disadvantages though, and we note these below. These limitations become prohibitive as the rule

- The grouping of rules into functional units (Tutoring Modes)
- An object-oriented data representation for rules
- A restriction on side effects
- A network formalism for representing patterns of rules and actions

6.4.1 Tutoring modes

We have already mentioned one technique which will help somewhat with the above problems. A Tutoring Mode, which is defined as a subset of the tutoring rules, can be specified in the Lesson Specification. This can greatly reduce the space of rules which must be searched during each control cycle.

6.4.2 Frame and object oriented representations of tutoring rules

Representing rules in frame-like structures provides two advantages: rules can be grouped into hierarchical classifications, and rules can inherit properties from rules above them in the hierarchy. Defining groupings of rules allows us to talk about entire sets of rules without specifying the members of the set. This makes writing Tutoring Mode specifications much easier. When we add more rules to a rule class, they automatically are included in the modes which refer to that class. We can also have "meta-rules" which limit the current applicable tutoring rules to one or a small number of groupings, without having to list all of the rules involved (Davis & Buchanan 77). An example of a meta-rule (i.e. a rule about rules) is: "IF the discourse has just started, do not use any rule in the rule-class Topic-Summary-rules."

The inheritance feature makes it easier to write and modify rules, because groups of rules can inherit antecedents or actions from a parent rule class. For instance, if we had a class of rules which encode knowledge about diagnosing misconceptions, the parent rule could have the antecedent "if the goal is to diagnose a misconception", and we would not have to specify that antecedent again for any rules below it in the hierarchy.

A further extension along these lines is representing rules in an object oriented paradigm. Such an implementation would incorporate the grouping and inheritance features listed for frames, and also have procedures called "methods" associated with rule classes (as described in section 4.1.5, and see Stefik & Bobrow 85). Different types of rules may require different types of processing. For example, there may be strategy rules, discourse rules, and

which exist only to manage program control. For example, if we used production rules to manage the control in figure 6.1, one might be "IF the Decision Mechanism has decided what action to take, THEN run the Action Mechanism." This type of information should not be put in the same rule set with rules which quide tutoring strategies. Also, rules which on the surface are about domain information may have additional "side effects" which set internal system parameters, such as what the last action was, how many times a rule was fired, etc. Tracing the flow of information in the system is more difficult when the control and domain information is treated identically.

6.3.4 Rule structure is opaque

Another difficulty with using rule based systems is that a uniform syntax for all rules hides any structure which they implicitly have. It may be that for some situations the desired flow of control is a rigid "flow chart" like network of actions. Conditionals, looping, and "case" statements in traditional programming languages make this structure explicit. In rule based systems the antecedents have to be rigged, as in the example above, to produce the desired effect, and the underlying structure is hidden to those who need to understand or modify the set of rules.

6.3.5 Lack of focus

Since all of the rules of a vanilla rule system are at the same level of importance, the rules specify primarily low level actions, and decisions are made on a local scale. The system is only concerned with the very next thing it should do. As a result, it is hard to make programs proceed with a high-level focus, and they tend to wander and meander toward their solutions, as if they were navigating with very limited visibility. (Some production systems, such as OPS-5, have been designed to take goal driven control into account.)

6.4 Modifications to vanilla rule based architectures

To eliminate the problems and limitations mentioned above, four modifications to the vanilla rule based architecture will be presented:

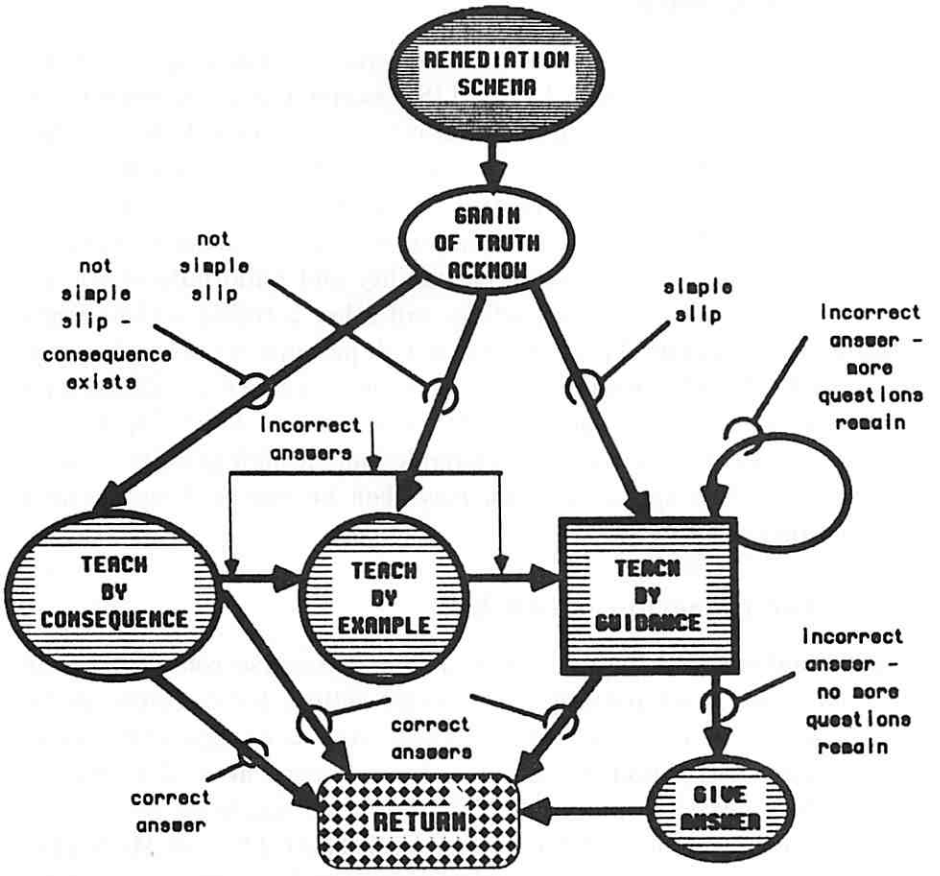


Figure 6.2: A sample Tutoring ACTion Network

diagnostic rule. Using methods we can specify procedures such as "check for match", "execute action", etc. such that the details of their operations will depend on the class of rules being used.

6.4.3 No "side effects"

In this architecture we restrict the Action specified by tutoring rules to be the name of a Tutoring Action (not a LISP expression to be evaluated). The rule specifies which action to take, but it is not taken right away, rather the action name is determined and the action itself is executed when the Control Sub-system relinquishes control to the Action Sub-system. This allows for a more uniform rule representation and easier tracing of what the system is doing. To further improve tractability and uniformity of control decisions, we will not let tutoring actions call other tutoring actions (they can however, call lower level procedures), and all parameter passing between actions must be done through one of the data bases (which are global data objects). For an action to specify that another action follow it, it must update the Control Data Base (for example: put a high priority goal on the goal stack). This specified action may then be executed on the next decision-action cycle.

6.4.4 Tutoring action networks

Woolf (84) and others have noted that human discourse contains certain common patterns. Such patterns of discourse actions (or discourse goals) are hidden when the discourse rules are represented as a large set of canonical production rules. Woolf (84) developed a transition network formalism for representing discourse patterns. This formalism has been recently redesigned in the form of discourse action networks (DACTNs, see McDonald, Brooks, Woolf, Werner 86). Our tutoring system architecture incorporates TACTNs (Tutoring ACTion Networks), which are patterned after DACTNs, but simpler. TACTNs are procedural networks which specify common or default action patterns. See figure 6.2 for an example. They are similar to ATN representation structures (Woods 70), which have been widely used in parsing, but are modified to support planning discourse actions. In ATNs the arcs represent states and the nodes represent tests. In TACTNs the arcs are tests and the nodes are actions. TACTNs combine the antecedent-action formalism of the production rule paradigm used in many expert systems with a recursive procedural network formalism which has been used in planning (Sacerdoti 74).

match. Thus the rest of the rule base is a third priority level.

TACTNs can have other TACTNs as nodes in their networks. This allows for recursive and subgoal types of control behavior. It is the job of the Control Data Base (below) to keep track of TACTN nesting. The need for subgoal or nested discourse patterns is supported by Grosz (80), whose theory of discourse structure incorporates a hierarchy of "discourse segment purposes."

6.5 The Control Data Base and goal driven tutoring

The Control Data Base stores information relative to the internal control mechanisms of the Control Sub-system. The types of information stored will depend on specific design decisions. Stacks of goals and/or pending tutoring actions will be included. Rather than specify that an action be run, a tutoring rule might specify that a goal or action be put on the stack. Then the Decision Mechanism may re-arrange this stack (sometimes called an agenda) to account for iteractions between goals, or to select the highest priority goal or most urgent action.

We have not said much in this paper about goal driven tutoring, but some representation of tutoring goals may be needed. Gross (86) has shown that some aspects of human discourse can be modeled using goal stacks. This is especially applicable to tutoring discourse. Here is an example of several levels of nested goals of sub-discourses: a goal to "teach Newton's third law" may activate a subgoal of "teaching Newton's law in static situations" which may active a goal to give an example. While giving this example the tutor may diagnose a misconception, and decide to correct it with a counter-example. While giving the counter-example the student may ask for some information. While the tutor answers the student's question, it has all of the above goals still active, waiting for their sub-goals to be completed. I.E. it is simultaneously trying to teach Newton's third law, teach it for static situations, present an example of it, correct a misconception about the example, and answer a student's inquiry.

If the decision making is to be goal centered, then some of the tutoring rules must match against goals. (ex: "IF the goal is to verify a misconception, THEN..."). The Control Data Base can also keep track of the current context, such as the current and recent topics, goals, actions, etc. Part of the complexity of human tutoring lies in the fact that the tutor is simulta-

The nodes are Actions and the arcs between nodes are complex predicates, called "Situations", which access and test information in the data base. After finishing an action, control passes to one of the children nodes depending on which Situation is true. The links imply rules of the form "IF you just finished doing Action x, and (..the rest of the antecedents...) THEN do Action y". That is, they are like rules which fire in a specific action pattern context. Since the Decision Mechanism need only check for a match between the data bases and few Situations leading from the current Action node, much efficiency is gained (compared with matching with all the rules in a vanilla rule system). Also, the structure of the action pattern is explicit and clear. TACTNs can be graphically displayed for inspection and modification.

In our architecture the Knowledge Sub-system distinguishes tutoring rules from actions, and the Control Sub-system selects one rule and sends it to the Action Sub-system to execute. Therefore, it is not obvious where TACTNs, which combine features of both rules and actions, would fit into the architecture. TACTNs are a subtype of Actions. That is, a TACTN can be specified anywhere an Action can be specified, i.e. by a tutoring rule or as a node of a TACTN. If a TACTN is called, it is "expanded" into its component action pattern.

When a TACTN is invoked (by a tutoring rule or another TACTN), the system continues to cycle through the Decision Sub-system and the Action Sub-system with each Action in the network. When a TACTN is in effect the Decision Mechanism will ignore the search through the entire rule base and choose the Action specified in the network. The reason we continue to cycle through the Decision and Action Sub- systems is that there may be certain control or bookkeeping procedures which need to be done before and after each Action is executed. This also lets us account for the fact that there may be extenuating circumstances where the normal action pattern specified by the TACTN should not be followed.

TACTNs facilitate a three level prioritization of tutoring rules. The arcs coming from the current TACTN node define the default actions in the given context, and are the middle priority level. We can also specify some rules in the rule base as "high priority rules" (or interrupt rules, or daemons, or meta-rules), which get checked before the Situations in the TACTN are checked. For example, we may always want to check that the student hasn't pushed the "quit" button before selecting an appropriate action. If none of the high priority rules fire, and if none of the TACTN arcs fire, then the Decision Mechanism can revert back to looking at the entire rule base for a

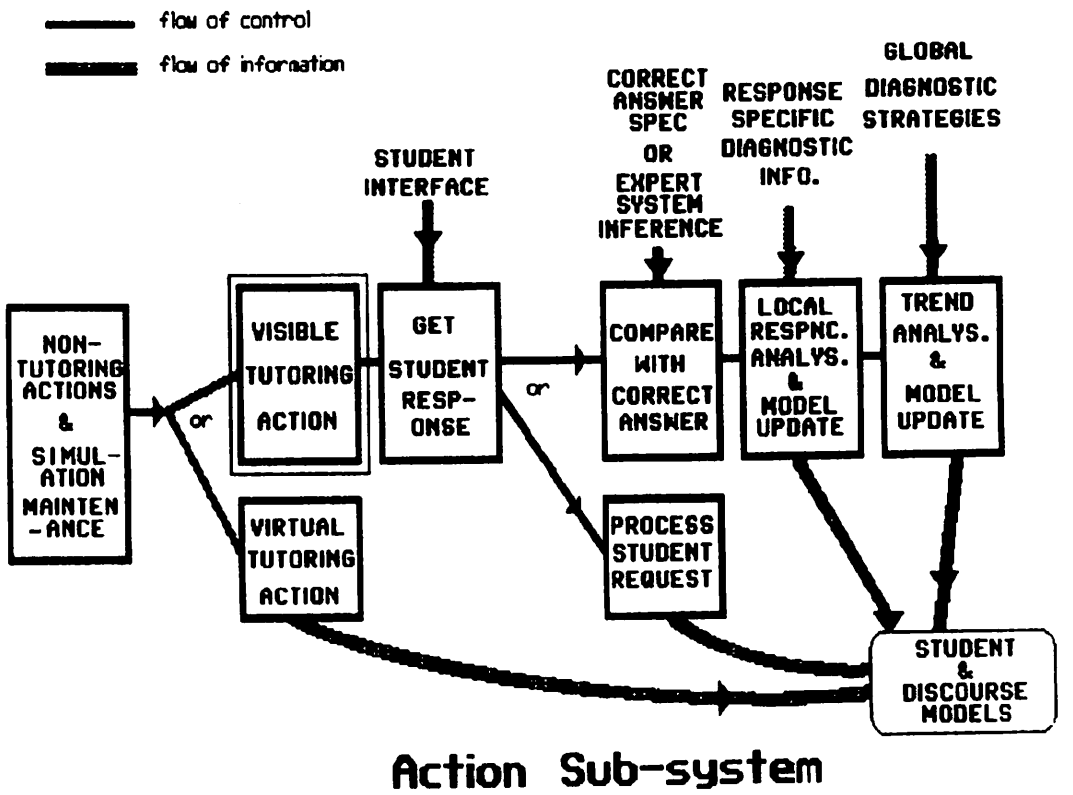


Figure 6.3: The Action Sub-system

neously trying to satisfy several goals, such as conveying a concept, keeping the student interested, not offending the student, using time efficiently, etc. It may be possible to have the Decision Mechanism choose an action which satisfies the maximum number of several simultaneously active goals.

6.6 The Decision Mechanism

The Decision Mechanism has the functions of both matcher and planner. The algorithms it uses may be quite complex, and are only suggested in this paper. Matching rule conditions with the data bases must be combined with some conflict resolution algorithm, as discussed above. Since some actions may only add goals or sub-goals to the control data base, pending actions and goals may accumulate. Some of these goals or actions may have higher priority. Also there may be significant interaction amongst them. In such cases it is the Decision Mechanism which must plan actions which most efficiently satisfy the set of pending goals and/or actions.

Part of the job of the Decision Mechanism is to manage the updating and use of the information in the Control Data Base. It uses current goals, TACTN-in-progress flags, etc. in the Control Data Base to coordinate control actions.

6.7 The Action Sub-system

The main functions of the Action Sub-system are to execute the tutoring action chosen by the Decision Sub-system, accept student input, and update the Student and Discourse Models. Figure 6.3 shows the components of the Action Sub-system.

6.7.1 Non-tutoring actions

This component is a catch-all for things such as upkeep of the simulation and maintenance of other non-tutor aspects of the learning environment. We assume that this box is essentially independent of the tutoring control and data, or at least does not change any of the tutor's data bases. In our general tutoring system the simulation and user interface are a components of (or are called by) the tutor.

MUST-NOT have ..." The output of the compare answer component could be in term of "xx was INCLUDED; yy was OMITTED, and zz EXTRA input was given" (Servi & Woolf 86).

6.7.5 Student and Discourse Model updating

Figure 6.3 shows two components which update the Student and Discourse Models based on the student's behavior. With these two components we distinguish two levels of diagnostic analysis. The "local response analysis" updates the models based upon information in the Task Specs. This information specifies how specific student responses to specific questions contribute to the student model. The "trend analysis" uses diagnostic rules (or, equivallently here, strategies) to analyse patterns of student behavior or belief accross the history of the student's behavior, and accross the entire student model. For example: "if the student's beliefs in concepts X, Y, and Z are under 30 %, then assert that she has misconception G at 80 %"; or "if the student asks for a hint 5 times in a row, then assert that she is Confused."

6.8 Blackboard archetectures

The architecture presented could be viewed from the perspective of the blackboard architecture paradigm (Nii 86). Blackboard architectures are characterized by having a global data base, opportunistic control, and modularized knowledge sources. They are most useful in situations requiring complex control, opportunistic action execution, and uncertain or conflicting data. In our architecture, the data bases are global, and the various parts of the system share information almost exclusively through the data bases (compared with passing information directly to each other). Control information is contained within a separate complex data structure, as in the Hearsay III blackboard architecture (Lesser 84). The Decision Mechanism prioritizes and selects actions and goals in both bottom up (data driven, as with actions that are triggered by certain patters of student behavior) and top down (goal driven, as with actions triggered as a result of a tutoring goal in the Lesson Specification) inference methods, as in some blackboard systems.

These similarities with blackboard architectures suggest investigating whether our architecture could be configured even closer to the blackboard model. Our architecture is presented in terms of rules and actions, as opposed to knowledge sources. This may be because we have viewed the tu-

6.7.2 Visible and virtual Tutoring Actions

Some tutoring actions are "virtual actions" (or "update actions"). Virtual actions do not make any noticeable change in the student environment. An example would be setting a goal to probe for a misconception. It would be later, when some other action actually did something to satisfy this goal, that the student would see any effect. Visible tutoring actions are actions that do something the student can notice. One exception is the Null action, a tutoring action which does nothing, leaving the student alone and passing control back to the Control Sub-system (hopefully a popular action in most tutoring sessions).

6.7.3 Get student response

After all visible tutoring actions, including the Null action, the computer will check for a student response. The student response component converts the raw information from the simulation or user interface, such as mouse positions, menu choices, typed input, data tables, etc., into some canonical (standardized) form acceptable by the succeeding components. Depending on the type of tutoring action, the Get-student-response component may wait for the student to do anything, or wait for the student to do as much as he wants, until he pushes the "done" button, or not wait at all, and continue on if the student has not done anything since it last checked for a student response. After completion of some tutoring actions control passes to the "compare with correct answer" component (see figure 6.3). This could be as trivial as matching the answer with a string, or a complex inferene concerning the acceptability of the student's response. Get-student-response also looks for student interrupts such as help requests, and processes these accordingly.

6.7.4 Compare with correct answer

This component compares the student behavior with the correct or anticipated behavior and outputs a measure of this comparison. The correct behavior can come from a "correct answer specification" for the problem given, or from asking the expert system what the correct answer is. We recommended designing a standardized language for the inputs and outputs of most of the components in the Action Sub- system. For example, correct answer specs could be phrased in terms of keywords such as "MUST have these three things", or "needs 2 OUT-OF the following 4 things:..., but

toring task in terms of discourse planning, as opposed to problem solving or classification (as would be the case if the overall focus were one of student diagnosis). We will be investigating whether or not the architecture would run more smoothly if represented in terms of modular knowledge sources, each one knowing the conditions under which is was applicable, and the types of conclusions which it could offer.

Chapter 7

TUTORING SYSTEM DESIGN STAGES

Up to this point we have been discussing a general control and knowledge representation architecture for representing and using the various types of information needed in intelligent tutoring systems. Now we will look at the steps that might be taken in implementing this architecture to design a tutor for some specific domain. We will assume for now that an ITS "shell" exists, with the various components as described in previous chapters, and our task is to determine the specific information to be entered into these components. If such a shell does not exist, the steps outlined here can still be used, and the architecture described in previous chapters can be used as a guideline for organizing the information. This chapter and the next deal with knowledge acquisition. This chapter suggests a methodology for discovering and implementing the rules and actions in the system, and chapter 8 suggests a methodology for analysing the domain and the tutoring goals so as to make the ITS more flexible and perspicuous.

7.1 Tutoring rule design stages

The tutoring rules are one of the last components implemented in a tutoring system, because their precise specification requires several other components to be designed first. Figure 7.1 shows a sequence of design activities leading up to the implementation of tutoring rules. In this chapter, for simplicity, we will use the term "tutoring rules" to refer to all kinds and levels of tutoring rules, diagnostic rules, Tutoring ACTion Networks, and tutoring Modes.

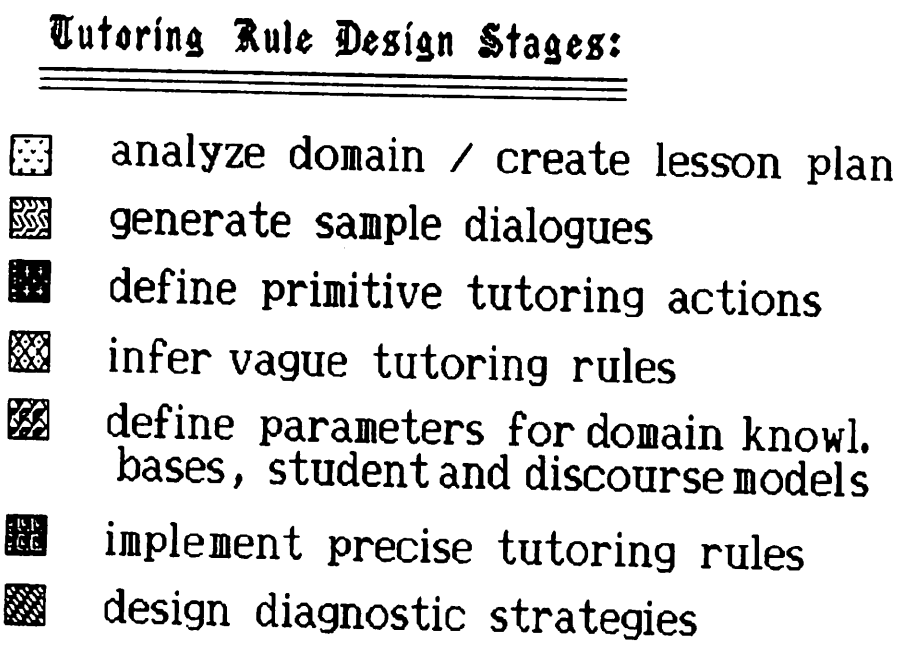


Figure 7.1: Tutoring rule design stages

After an analysis of the domain to organize the knowledge to be taught and the general tutoring methods employed, sample dialogues are generated. Sample dialogues could include transcriptions of taped interview studies, dialogues existing in the literature from research on teaching the subject, and/or synthetic dialogues generated by the domain expert/teacher.

Generation of the sample dialgogues is important. The less synthetic they are, the better. Though the effort required to tape and analyse actual or mock tutoring sessions is non-trivial, and certain aspects of the dialogue between the proposed ITS and the student can not be faithfully measured in a non-computer tutoring session.

They provide much design-relvant information which would not be available from a set of un-tried specifications of how the tutor should behave. Also, the process of generating and analysing them helps to reify and shed new light on the initial design conceptions. Sample dialogues should provide information relevant to:

- The variety of types of questions, explanations, or prompts the tutor will need
- The form and content of the inturruptions and questions the student should be able to ask
- The degree of naturalness or mechanicalness of the dialogue
- The degree of interaction (i.e. complexity) between the discourse context and tutoring strategy used

The dialogues (along with annotations by the interviewer) are analyzed in several steps (see figure 7.1). First the primitive tutoring actions are defined. Examples are: congratulate-correct-response, give-extreme-case-example, introduce- new-topic, give-away-correct-answer, etc. Next "vague" tutoring rules are inferred from the dialogue. They are of the form "IF <some condition> THEN <Action>," and are called vague because at this stage they must refer to well-defined tutoring actions (defined in the previous step), but can have anything in the "some condition" part. The vague rules try to explain the conditions which motivated the instances of the tutoring actions in the dialogue. Next we analyze the vague tutoring rules to determine parameters for the Student Model, Discourse Model, and the pedagogical aspects of the Domain Model. For example, if a vague rule says "IF the student is confused THEN give-leading-question," then we need a parameter

in the Student Model called "confusion-state." A vague rule saying "IF the student answered incorrectly too many times in a row, THEN change-focus" requires a "number-of-wrong-answers-in-a- row" parameter in the Discourse Model. A vague rule saying "IF the information being taught is of a factual nature, THEN don't ask probing questions" requires a parameter in the Domain Model for "knowledge-type," which distinguishes factual types of knowledge from other types. A vague rule which says "IF introducing a new topic THEN review-previous-topic" requires a record of the previous topic in the Discourse Model. Next, precise tutoring rules are implemented. Their condition refers to well defined parameters in the knowledge bases, such as "IF student-confused THEN give-leading-question." The actual implementation of the rules may be quite complex, since many of them may be interdependent. Rules may be defined hierarchically, grouped into tutoring Modes, and arranged in networks, as suggested in Chapter 6. The data base parameters may have to be redefined iteratively in order to make distinctions of fine enough grain to enable clarity of the tutoring rules.

The last item in figure 7.1 is designing diagnostic strategies (or rules). Above we have identified parameters in the Student Model such as "is-confused," an "has-adequate-knowledge-of-current- topic." It remains to determine how these parameters will be measured, i.e. what student behaviors will provide evidence for the Student Model parameters (including the expert knowledge "overlay" portion of the Student Model). We do not address diagnostic strategies much in this paper, but their design will typically be one of the more difficult aspects of tutoring system implementation (see Sleeman & Henley 82 and Clancey 86B).

7.2 Tutoring system design phases

In figure 7.2 we elaborate on the activities shown figure 7.1, and include steps in building the knowledge bases, as well as in designing the tutoring rules. This design activities chart is for instruction on the order of one lesson, one topic unit, or one chapter. Figure 7.2 outlines how information from one activity feeds into another. The connections between activities indicate that some of the activities can be going on in parallel. The steps are grouped into 5 phases, roughly outlining these activities: 1. Specification of tutoring goals and strategies, 2. sample dialogue generation and analysis, 3. putting the rules inferred from the dialogue into a computationally precise form, 4. implementing the components of the tutor, and 5. testing and refinement.

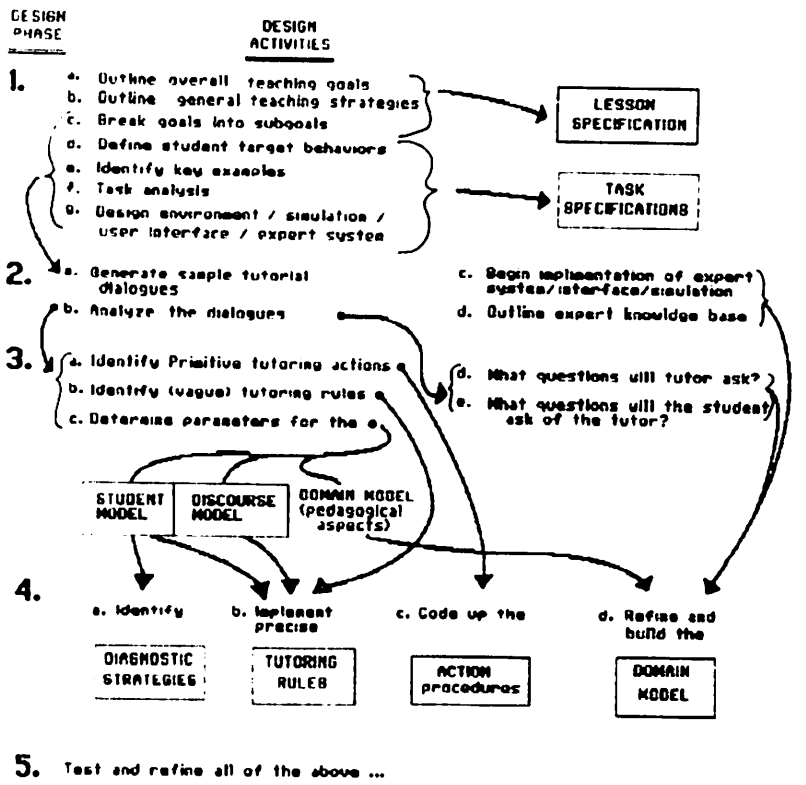


Figure 7.2: Steps in designing an ITS

This activity organization is not intended as a rigid temporal scheduling of personnel tasks. The designers of an ITSs may find that they are skipping all over this chart. The chart gives an indication of what design tasks need to be in progress before others can be seriously attempted.

7.3 Phase 1: Global goal and strategy specification

In phase 1 we first outline the overall teaching goals for the topic unit (item 1a). This includes the important facts, skills, and concepts to be learned. For example, to become familiar with contact forces in static situations, and to be able to solve simple word problems involving static forces. At this stage we also need some general tutoring strategies for our domain (item 1b). For example: using an environment in which the student can create

objects of different weights and sizes and measure the forces they have on each other; the student will be allowed to experiment in this environment, and then the tutor will ask her questions about situations that the tutor presents.

The above are not really planning tasks, they just represent the ideas which motivated the designers to build a computer tutor in the first place. These ideas are next refined. The learning goals are broken into subgoals (item 1c). The goal skills and knowledge are analyzed to produce a network of prerequisite skills and knowledge, as is popular in classroom curriculum design (see Joyce & Weil 72, and Barr et. al.'s BIP tutor discussed in Wenger 86). The more detailed this analysis the better. The student target behaviors are identified (item 1d, discussed in more detail below). Key examples and expected misconceptions are identified (item 1e). The tasks or activities that will allow the learner to build her knowledge and achieve the learning goals are specified and ordered (item 1f).

Also in this first phase, initial specifications for the domain expert system, learning environment (including the user interface), and simulations are drawn up (item 1g).

7.4 Phase 2: Sample dialogue generation and analysis

The simulation and/or learning environment is designed so that coding of these can begin (item 2c). "Knowledge acquisition" of the domain knowledge starts, i.e. putting the knowledge in terms appropriate for a computer knowledge base. The knowledge hierarchy developed above is refined and organized into knowledge units. Types of links between the knowledge units are specified (item 2d).

Occurring simultaneously with the above, the knowledge organization and learning tasks defined in phase 1 are used to draw up a primitive lesson plan which can be tried on students. Sample tutorial dialogues are obtained as described above (item 2a). These dialogues (and/or the interviewer's comments of the tutorial sessions) are analyzed for the purpose of obtaining the information in phase 3 (item 2b).

Once the simulation/learning environment is implemented, it should be used to generate more sample tutorial dialogues. At this phase the computer is used as a tool for simulating the domain, presenting examples, experimenting, etc., and the interviewer does all tutoring and analysis of student actions

(i.e. he takes the place of the Decision Sub-system described in Chapter 6).

7.5 Phase 3: Defining the tutoring rules

From the dialogues we would like to infer the tutoring rules which are being used. Referring to figure 4.3 (the Control Sub-system), we can describe the tutoring rules as a mapping from data about the domain (the static information) and about the current context (the dynamic information) to a set of potential tutoring actions. (The Decision Sub-system figure 4 carries out the mapping which the tutoring rules define.) That is, there is a correspondence between all possible tutoring situations, and all possible tutoring actions, and this correspondence is specified by the tutoring rules, which (we assume for simplicity) are of the form "IF <condition> THEN <action>." There are three things to define: the set of actions, the set of conditions, and the set of rules (which are a mapping from the conditions to the actions).

In phase 3 we analyze the dialogues to define primitive tutoring actions and the parameters for the Student Model, the Domain Model, and the pedagogical aspects of the Domain Model, as described above and shown in figure 9 (items 3a, 3b, and 3c in figure 9).

Also in phase 3, somewhat independent of the above, is an analysis of the discourse to determine what kinds of knowledge and questions the tutor should be able to ask, and what kinds of questions about the domain knowledge the tutor should be able to answer. This information is needed to better define what knowledge is needed in the domain knowledge base, and how it should be represented.

7.6 Phase 4: Implementing the system components

Phase 4 represents the point at which the code for the various components is implemented into computer knowledge bases or LISP code. (The Student and Discourse Models are boxed in phase 3 rather than phase 4 because they do not contain knowledge about the student or discourse until tutoring actually begins. Only their structure is defined.)

After a precise vocabulary for describing tutoring actions and parameters is complete from phase 3, the Tutoring Rules can be implemented. As mentioned above, this will include defining Tutoring Modes and Tutoring ACTion Networks. Making tutoring rules precise may take many iterations.

For example, the vague rule "IF the student is confused THEN give a hint," may eventually become "IF the student is confused on the current KU while the tutor is presenting new knowledge, and she hasn't been given any hints before, THEN give a level-one hint." We must evaluate the many possible interactions and the possibilities of rules appearing to be applicable in situations where it should not be.

The Domain Model is implemented according to the analysis of the expert knowledge and the sample dialogues, as indicated in figure 9. Diagnostic strategies can then be incorporated into the Task Specs and the Domain Model (see section 4.2).

Phase 5 makes explicit the iterative nature of the design process, and includes the obvious activity of analyzing what we have so far, and if we are satisfied, trying it out on some students and repeating the whole process to refine the system.

Again, figure 10 does not suggest a step by step sequence of activities, but shows conceptually how the design of different components are related. At every step along the way the designers will probably be noting ideas about activities lower in the diagram, and be circling back to refine earlier activities. The diagram also highlights the importance of the sample tutorial dialogues, form which much of the information is gleaned.

Chapter 8

KNOWLEDGE ENGINEERING FACTORS

In the previous chapter we discussed how tutoring rules could be designed based on an analysis of sample tutorial dialogues, which were in turn planned according to the teaching goals of the topic to be tutored. We implicitly assumed that the tutoring rules implemented in a computer tutor attempt to duplicate the behavior of human teachers. However, many of the tutoring rules and other design decisions incorporated into tutoring systems will be based on principles and assumptions about teaching in the domain, not on an analysis of (real or synthesized) human tutorial dialogue. Even in the case of tutoring rules which are patterned after discourse protocols, there are assumptions about teaching and learning in the domain behind the lesson plan and the dynamic decisions made by the human tutor. In this chapter we discuss ways of evaluaating the many factors which come into play in ITS design, some of which are usually implicit. We suggest several perspectives for the analysis of design factors, and give examples of how this information might effect the design of the system. We also advocate an explicit accounting of the design decisions made by research teams.

In looking at the many tutoring systems in the literature, it is apparent that no two of these incorporate the same rules about effective tutoring. Few tutoring systems have both an explicit philosophy of tutoring and learning and an explicit realization of this philosophy into tutoring rules (but see Anderson's research). For the sake of system clarity, modification, and evaluation (as discussed in chapter 2), it is desirable to be explicit about both the overall philosophy behind a tutor, the rules used to guide the tutorial

discourse, and the assumptions behind each of the rules. Also, it is only by being aware of the underlying assumptions behind the rules of a given tutoring system that we can decide whether we can incorporate similar rules into a new system.

8.1 Rules, principles and assumptions

Appendix 1 shows tutoring rules and principles from some well known tutoring systems. One observation is that they don't all agree on how to tutor. This is not surprising, considering the variety of domains being tutored, and the variety of underlying psychological, pedagogical, and philosophical assumptions (both explicit and implicit) supporting the design of each. Another observation about these rules and principles is that they describe factors on many different levels. Some rules involve psychological or philosophical assumptions; some rules are given without explaining why they should work; some refer to vague strategies, and others to specific actions. Some are more relevant to a particular domain than others. As an illustration of a continuum of specificity of tutoring rules or principles, consider the following ("English-ized") rules from hypothetical tutoring programs:

1. *If question 12 is answered wrong, give explanation 5.*
   This is the type of very specific coupling of diagnosis and action found in (non-intelligent) computer assisted instruction programs.

2. *If the student gets more than three questions wrong in a row, then do a remedial exercise for the current topic.*
   This type of rule is more typical of intelligent tutors. The system must monitor the student's answers and infer which remedial exercise to give. Note that it says nothing about the reasons why the rule is applicable.

3. *If the student is very CONFUSED, then GIVE-HINTS for the current topic.*
   (Assume here that the property CONFUSED is determined by some low level data, such as the number of wrong answers, and GIVE-HINTS is interpreted differently for different topics.)
   This rule is at a more abstract level. Abstraction makes the rules more transparent, and allows a separation of the rules themselves from the details of how the conditions and actions are implemented. A diagnostic strategy must infer the level of confusion from student behavior

(such as number of times asking for help). The GIVE-HINTS action is a general action which gets interpreted differently for different types of topics.

4. *If a student answers a problem incorrectly, give a series of hints, from vague, to more specific, to giving away the answer. Give the student several opportunities to think again about the situation, and learn from mistakes, before giving authoritarian feedback, which the student can accept and memorize without critical thought.*
This represents the principle behind the previous rule. It states a pedagogical bias or strategy, and is not precise enough to be part of a computer program (with today's technology).

5. *People learn through an active process of concept formation while trying to account for new observations in the context of their previous knowledge.*
This is a theory– a psychological, or philosophical assumption. It represents the reason for the previous principle and the purpose for the rule above it.

Note that one can (barely) conceive of a computer tutor which could take principles and infer tutoring rules, or even take psychological assumptions and infer principles and rules. This of course is stretching it, but points out that as we progress down the list, more intelligence is needed to translate the rule into a specific tutoring action. We suggest that ITS rules be implemented on the level of item 3, and that all such rules (and other design decisions involved in system design) are explicitly annotated with the principles and/or assumptions which are behind them (as in items 3 or 4). This allows systems to be evaluated and modified on the bases of their underlying assumptions, and allows tutoring systems to explain why they are performing certain tutoring or discourse actions as they do so. The ability to explain tutorial discourse is beneficial for testing the system and for exhibiting computer "candidness" to the student.

8.2 Introduction to an analysis of the factors affecting tutoring system design

In designing new tutoring systems, we would like to glean the most effective tutoring rules from the research of our colleagues who have designed and

tested these systems. Our question is not "what are the correct rules and principles for tutoring?," since these will differ according to domain and tutoring goals. Our question is not even "what are the correct rules for my domain?," since, as discussed in Chapter 5, a variety of tutoring styles (or Modes) should be incorporated. What we want to know is "which rules are appropriate in each particular tutoring situation?" In order to answer this question by looking at existing systems, and in order to organize design decisions concerning appropriate tutoring rules for new systems, we will present an analysis of the factors involved in describing tutoring situations and assumptions.

The left side of figure 8.1 shows four types of factors involved in the design of tutoring systems. These factors are the (implicit or explicit) assumptions about the domain, the learning goals, and pedagogy, which underlie the specific contents of the various components of the system. The behavior of the tutoring system is determined by the contents of these system components (as shown in the right side of figure 8.1): the tutoring Rules (including meta- rules, TACTNs, diagnostic rules, etc.), the Lesson Specification (which specifies the overall teaching goals, lesson plan, and tutoring modes), the examples and student tasks specified in the Task Specifications, and the simulation / leaning environment / user interface. In all tutoring systems, regardless of their architecture and components (i.e. whether of not they have the components shown to the right in the figure), assumptions in the four areas show to the left of figure 8.1 underlie the design and behavior of the system. We will discuss each of the four areas below. The categories given are somewhat fuzzy and overlapping, but we present them as distinct categories to provide a framework within which to do the analysis.

The **"pedagogical characteristics of the target knowledge"** are determined through an analysis of the differing needs and constraints for learning different types of knowledge. Many of these characteristics and the reasons why they are useful, have been discussed in the Chapter 4 (on knowledge representation). For example, fact/skill/concept distinctions, and distinguishing between qualitative and quantitative knowledge. In this chapter we will outline some finer distinctions based on pedagogical needs.

All domains seem to require a wide variety of types of target knowledge, such as facts and skills, so rules referring to types of target knowledge (mentioned above) are fairly general and applicable to many domains. These rules determine the dynamic decisions made by the tutor. Domains also have predominating characteristics which determine the overall design decisions for tutors, such as what general tutoring strategies will be used, how the simu-

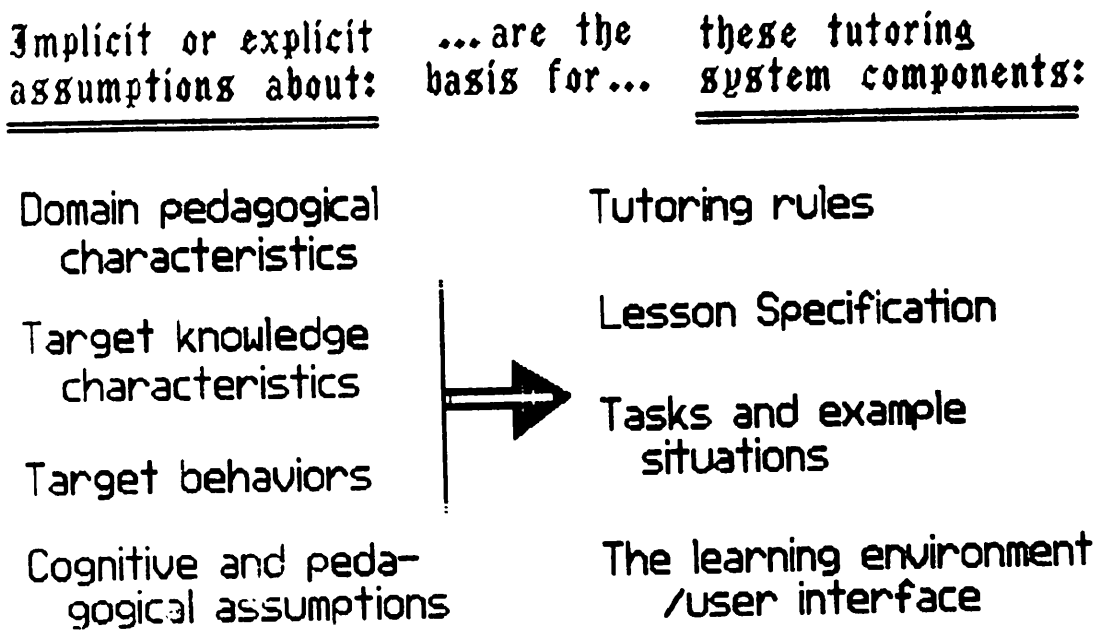


Figure 8.1: Factors involved in the design of ITSs

lation/learning environment/user interface is designed, and the contents of the Lesson Specification.

The **"pedagogical characteristics of the domain"** refer to characteristics which determine the overall approach for a tutor. For example, even though tutors for electronic circuit design and LISP programming may have similar rules about how to teach certain categories of knowledge, there are probably significant differences in their most auspicious overall tutoring approaches. To make clearer the distinciton between domain characteristics and target knowledge characteristics: both domains use many types of knowledge. Some types of knowldge, such as "memorized algorithms," may be common to both domains. Even though considerations about domain idiosyncratics determins the design of some tutoring strategies (such as how much time should be spent in coaching environments), there may be other strategies (such as how to teach memorized alogirthms) which depend only on the type of knowldge, and are independent of the domain.

The **"target behavior"** item in figure 8.1 refers to a specification of what student behaviors are expected as the result of successful learning. It is not enough to specify only the target knowledge–"the laws of electricity," for example. This alone leaves an uncertainty concerning how we will know when the student knows it, and to what depth we expect to teach it. Using specific target behaviors (such as successfully solving problems, or answering questions) aids in evaluating the effectiveness of the system, designing diagnostic rules, and focussing the design of the system so as to encourage these behaviors in the student. Note that tutoring system design can be guided by either target behaviors or target knowledge (or both). Tasks (which specify behavioral tests in their questions for the student) can be incorporated into the system with the goal of teaching and testing certain target knowledge units, or, alternatively, the knowledge units incorporated can be determined based on what knowledge the student will need to exhibit a certain target behavior.

The last item in figure 8.1, **"cognitive and pedagogical assumptions,"** refers to the biases and assumptions about learning and teaching which the designers hold. For the most part these are taken as true, and are difficult to verify experimentally to the satisfaction of all. These assumptions are the ideological foundations upon which the design of a system is based. We do not advocate any particular philosophy, but suggest that the assumptions be made explicit, and present some examples.

The initial design phase of a tutoring system should include consideration of factors from all four of these area. Below we will enumerate a few

factors from each area, and show how they affect the design of some common tutoring systems.

8.3 Pedagogical characteristics of the domain

Teaching about physical systems

Domains which involve physical systems, man made ones (such as circuits and steam boilers–SOPHIE and STEAMER) and naturally occurring ones (meteorology and medicine, WHY and

GUIDON), may require that the tutoring system be able to generate sophisticated explanation about causal and functional relationships (as in WHY). Man made systems will be more focussed on the function or purpose of the components. Natural systems will focus more on the causal relationships between components. To contrast, teaching abstract (man-made) systems, such as programming (PROUST and The LISP Tutor), geographical facts (SCHOLAR), and language (CALLE (Xerox 85) and CALEB (Cunningham et. al. 85)) may not need complex explanation facilities.

Domains prone to misconceptions

Misconceptions can occur as a result of non-academic experiences, or during instruction. Misconceptions which arise during instruction can result if a concept or skill is over-generalized, over-specified, or just mis-understood. White & Frederiksen (86) argue that these types of misconceptions can be greatly reduced with careful sequencing in instruction. However, learning in some domains, such as physics (Clement 85, McDermott 84), probability/statistics (Tversky & Kahneman 74), and programming languages (Soloway & Johnson 84, Bonar & Soloway 85), can be severely impeded by the influence of beliefs or patterns of inference developed quite independent of any academic context (see Claxton 86 and diSessa 85). In such domains tutoring rules must incorporate detection and remediation of common anti-productive beliefs.

Problem solving domains

In some domains (or parts of domains) the focus is predominantly on problem solving, rather than learning new facts, skills or concepts. Examples of tutoring systems in these domains are SOPHIE, GUIDON,

The Geometry Tutor, and Keynesville (Shute & Bonar 86). These tutors pay explicit attention to the problem solving process (diagnosis, proof, or rule induction), as well as the specific rules and heuristics used to solve the problems. Student modelling (if done) is relatively more illusive in these domains.

Information density

Domains such as programming (in The LISP Tutor) and medical diagnosis (in GUIDON) can be characterized as having many rules, each with relatively straightforward application (here "rules" refers to the human problem solving rules, such as those found in a textbook, not computer program rules). Domains such as physics and electricity have relatively fewer rules or concepts. In these domains the rules can be easily learned, but applying them properly can be less straightforward. In domains of low information density, there is more need for examples showing multiple viewpoints (as in WHY), non-examples, and boarder cases (Rissland, Valcarce, & Ashley 84, and Tennyson & Park 80).

8.4 Pedagogical characteristics of the target knowledge

Newness of the knowledge

If the subject being taught is new to the student, tutoring rules may be needed which introduce it, piece the new knowledge together in a systematic way (SCHOLAR, LISP, CALLE, CALEB), or allow the knowledge to be discovered (Keynesville, WEST, WUMPUS, MENDEL). If topic is not new to the learner, the tutoring may focus more on diagnosing misconceptions and

giving the student a chance to practice using the knowledge in a variety of contexts (as in SOPHIE, GUIDON).

Memory intensive vs. inference intensive knowledge

When asking questions about information which requires primarily recall or recognition of facts or patterns, the strategy of immediate feedback may be best (Anderson 84, and *The Little Lisper*, a programmed learning text). When asking questions which require more deductive or creative thought, it may be advisable to let the learner

learn from mistakes, giving only enough information to let the learner debug his own knowledge.

Other characteristics: Meta- and qualitative knowledge

In chapter 4 we gave reasons for distinguishing between qualitative and quantitative knowledge, and between knowledge and meta-knowledge. Appendix 2 has a detailed breakdown of types of knowledge for pedagogical purposes (compared with Chapter 4, which categorized knowledge according to knowledge representation needs). It is included to suggest a taxonomy which may help distinguish different types of knowledge. We will not discuss it in detail however, since at this point there is no clear science for tutoring methods based on different knowledge types of this fine a breakdown.

8.5 The target behavior

Solving unfamiliar problems

Teaching such that the student can solve routine problems in a domain differs greatly from expecting her to solve unique problems, i.e. problems that don't fit nicely into any previously learned pattern (Chi et. al. 81, Larkin 83). The level of difficulty and/or depth of learning which is expected for any given knowledge unit is best specified by listing examples of the types of problems which the student is expected to solve. Some tutoring systems, such as WUMPUS, Whites electricity tutor, and WEST, pay specific attention to the various levels of expertise in understanding a knowledge unit.

Target behavior for memory intensive learning

In specifying the target behavior for knowledge which is predominantly memorized, such as facts, simple skills, formulas, relationships, etc., a distinction can be made between expecting (in order of difficulty:) recognition ("Is this a marsupial?"), associative recall (give a hint or strongly associated context: "are all things with hair marsupials?"), free recall ("tell me all the kinds of marsupials"), and generation (creating a new exemplar of the knowledge type). Which of these is expected will effect how the knowledge is taught (Cofer 79). For example, knowledge always presented in a limited context may not be retrievable in a different context.

Target behavior for domain-independent skills

Tutors which attempt to teach domain independent complex skills, such as investigative/inquiry/experimental skills (Keynesville, SOPHIE), or logical inference skills (such as WUMPUS) may have to give students tasks from several domains before they are sure that the student has such a skill. This is not currently done.

8.6 Cognitive and pedagogical assumptions

Using examples

Tutoring systems differ appreciably in the importance they place on examples and how they are presented. Rissland (83) stresses the importance of presenting examples as well as definitions of concepts, and Tennyson & Park (80) give psychological data implicating their importance in the learning process. In WEST, providing examples for the "issues" taught is key. Collins (77) and Tennyson & Park have techniques for selecting and ordering examples, and using counter-examples and examples which focus on specific features.

Teaching mental models

Several research groups stress the importance of assisting the student in the construction of runable mental models of the system under study (deKleer & Brown 82, White & Frederiksen 86, Papert 80). Tutoring under this assumption involves asking the student to make predictions about the behavior of the system under different conditions, and allowing the student to play with a simulation of the system in order to intuit the patterns of its behavior and a causal model of its operation.

The role of feedback

Opinions differ concerning the type and frequency of feedback needed in tutoring. Anderson, as well as those in the behaviorist tradition, emphasis the importance of immediate feedback. Others, of a more constructivist bent (as mentioned above) would rather err on the side of giving too little feedback, since too much can effect the student's motivation and cognitive engagement. Most researchers have some allocation for giving hints before giving away an answer, but the degree differs. Burton & Brown (82) and others have mentioned the importance of positive as well as negative feedback.

Constructivist paradigms

There are many consequences resulting from a constructivist (explained

in a previous chapter) theory of learning. Constructivism is a "paradigm" (as is behaviorism), not a tight theory, and tutoring strategies based on it can be quite diverse and even incongruent. Some (Clement, McDermott, and others), focus on the importance of recognizing and remediating "deep" misconceptions, persistent erroneous intuitions inferred from common experiences. Some focus on the need for accessible learning "tools" and rich environments, giving the learner maximal control over the learning situation. Some (White and Frederiksen and others) stress the need for thorough analysis of prerequisite knowledge, and careful sequencing of the topics taught. Some stress the need to encourage in the learner a state of cognitive dissonance (or disequilibrium) (Lawson and others), which will motivate the learning process. Some (Polya and others) stress the need to apply new knowledge in realistic problem solving context. Some (such as Papert and Clement) mention the importance of anthropomorphizing the concepts; i.e. encouraging the learner to put himself in the place of the gear or the ice puck in the problem.

Theories of the mind

Anderson seems to be the only ITS designer to date who is basing his tutors on a specific cognitive theory. The theory includes assumptions about the limitations of working memory (which result in design decision to give the student a larger effective working memory), and assumptions about the structure of knowledge in the mind. Part of Anderson's theory (the ACT* theory of cognition (Anderson 83)) is that human skills can be modeled using a set of production rules (if-then rules), and this forms the basis for many design decisions (Anderson, Boyle, Farrell, & Reiser 84). (Others have represented skill knowledge as production rules in tutoring systems, but have not designed entire tutors based on a cognitive science theory, as does Anderson.)

Chapter 9

CONCLUSIONS

9.1 Review

We have given many suggestions aimed at systematizing the design of Intelligent Tutoring Systems, and providing some common ground on which to compare systems and share research ideas. After outlining some problem areas in the field, and the research goals which these suggest (Chapter 2), we presented architectures for knowledge representation (Chapter 4 and 5) and program control (Chapter 6). We then outlined how these architectures could be used in implementing a tutor in some domain (Chapters 7 and 8). Along the way we have introduced terminologies and classifications to help make the process more precise. At UMASS we have begun to implement these ideas.

9.2 Contributions

This paper has both summarized past work in the field and offered new suggestions. As such it serves as an introduction to the literature, an introduction to the main concepts in ITS design, an analysis of previous work in them, and a recommendation for improvements. The ideas herein come from the literature and our experience at UMASS implementing tutors and general components of tutors. Therefore some of the suggestions are culled from the literature and others are original. Below we will highlight the portions of the paper which are innovative or potential new contributions (these claims are accurate only to the extent of my familiarity with the ITS literature). The claims that certain ideas are original pertains to within the ITS

research literature only. I assume that concerns similar to ours are spawning similar ideas elsewhere. Also, we pick and choose from several emerging AI technologies. This paper does not contain contributions to the fields of artificial intelligence or educational psychology, but we apply research in these fields to the field of intelligent tutoring systems in perhaps novel ways.

- The control architecture proposed in Chapter 6 (figures 6.1 and 6.3) is original (although the individual AI methodologies, such as object-oriented programming, meta-rules, etc. are all well established technologies.)
- The Tutoring Action Discourse Networks, as a means for describing common or default discourse patterns is original (Woolf & Murray 86).
- The "High Level Lesson Planning Specification" Chapter (5) is predominantly original. There has been little research effort going into designing ITS systems flexible enough to teach in a variety of diverse "modes" or tutoring styles, teaching different aspects of a unit of knowledge in different ways, or at different levels. (Although the fact that there is a spectrum of tutoring styles in ITS's has been often discussed, for example, in Wenger 86).
- The introduction of "Tasks" (containing example situations and task associated with them) as entities in a separate knowledge base (separate from actions and expert knowledge) is new. The introduction of an explicit "Lesson Specification" data structure may be new, but I think that very similar constructs exist implicitly in some ITS systems.
- The Action Mechanism (figure 6.3), embedding the execution of each tutoring action which the tutoring rules select within a structure which contains ubiquitous action procedures (such as get-student-response), may be new.
- The division of diagnostic components into local response analysis and trend analysis (figure 6.3) is new.
- The categorization of knowledge distinguishing deep concepts and nexus concepts is new.
- A general methodology for design and knowledge engineering for arbitrary tutoring systems, such as the one given in Chapters 7 and 8, has not been seen in the literature, though many ITS system designers may well have similar ideas.

- The focus on a general system architecture or authoring shell is not unique (Bonar is working on it, and Clancey has suggested it), but it is quite uncommon.

- The concern for a more precise terminology with which to describe types of knowledge and actions is also uncommon (but see Clancey 86B). Our way of taxonomizing knowledge and the factors effecting knowledge engineering may be useful.

9.3 Remarks on ITS evaluation and the effects of ITS novelty

The art of designing Intelligent Tutoring systems is in its infancy. Though the field has existed since approximately 1970 (Carbonell 70), reports of systems being used successfully to teach non-trivial samples of students are only recently being rumored (but see Suppes' EXCHECK tutor described in Wenger 86). This is understandable. It took many years for those using new technologies such as the book or the television to agree on felicity guidelines. The computer as a learning medium will have a similar, though perhaps accelerated, learning curve. The design of successful ITS systems taxes the state of the art in AI technologies such as knowledge representation, interface design, machine learning, and natural language processing. Also, codifying the principles of good human tutoring will be difficult, considering that even for classroom or textbook teaching, it is hard to find research evidence for teaching principles which have proven to be successful. However, the existence of computer tutors will contribute to the analysis of teaching and learning in general, since using computers eliminates many uncontrollable factors involved in human-human interactions, and because attempts to codify tutoring rules forces instructional experts to examine what they are doing at a new level of detail and clarity.

Because of the newness of the field, we should be cautious about evaluating systems which present the student with many options. Learners (school-aged and adult) are not used to being immersed in such flexible instructional environments. Students will now be able to manipulate these novel learning environments in ways impossible with textbooks or lectures. Here are some examples of the power learners can have over their computer tutoring systems (given in informal natural language, but they could also be menu items):

- Effecting the rate of delivery:
  "Please slow down, I'm confused" (or "speed up, I'm bored").
  "Please give a more detailed explanation of that example."
  "Stop here–I'll be back tomorrow."

- Exploring the space of knowledge:
  "How does entropy relate to the stuff in the last chapter?"
  "What concepts do I need to have before I can understand entropy?"
  "Can I preview what's coming in the next chapter?"
  "What does 'vectored interrupt' mean?"
  "List all the formulas involving friction."

- Asking "meta-dialogue" or pedagogical questions:
  "Why did you give me that example?"
  "Is this topic difficult to learn?"
  "What facts do you think I don't know at this point?"

- Choosing the material presented:
  "I want to try the next experiment now."
  "Please give me another example of convex polyhedra."
  "Give me a trace of how you concluded that disease from the data."

- Choosing the teaching mode used:
  "I learn better if you give quick feedback for wrong answers."
  "Could you hold my hand through this one–explaining each step?"
  "Don't give me hints so fast, I want to think it through myself."

- Experimenting and "playing" in rich, complex, or realistic environments or simulations:
  "What if we replaced the law of gravity with a different one?"
  "Now I'm approaching the third moon of Jupiter–fire left thruster number 2 for 3 seconds."

But having all this power available does not mean it will be utilized. Learners are, for the most part, not used to this kind of control. They are not used to monitoring their learning or problem solving progress (Confrey 85) and/or altering their learning environments according to reflective meta-cognitive analysis. When we evaluate the success of these powerful environments using random students, we are sure to find that they under-utilize the system, and do not learn as much or as quickly as expected. We may need to first instruct students in how to make full use of the potential of such

systems; how to explore, play, question, and summarize; how to monitor their own progress; how to communicate with a computer tutor on a "meta-dialogue" level. Fair evaluations of ITS's may only come with a generation of students who have used them several times.

On a similar note, we who are in the business of designing intelligent tutoring systems are under somewhat of a handicap. We (99.9 percent of us) were not taught using ITS's. Our education experiences have been traditional, despite our belief in the power of alternate leaning methods. As we design our systems we can only hypothesize about the experiences of a novice in some domain sitting at a computer terminal and interacting with it. Compare this with textbooks and lectures. We all have our own guide-lines concerning how to write papers or give talks so that the audience will be attentive and motivated, and learn most effectively. We have these guide-lines because we have been reading books and listening to lectures for years, making mental notes on what is effective and what doesn't work. The vast majority of us have not even once learned a new topic in some field via a computer (not to mention an ITS). Perhaps the next generation of ITS designers, based on their experience using the systems we are now building, will produce intelligent tutoring systems beyond our imagination. Of course, if they build systems that match what we've imagined, that would be quite a feat itself.

Appendix A

SAMPLE TUTORING RULES AND PRINCIPLES

Below we show tutroing rules and principles from six well known ITS research projexts. (Note: Reasons for tutoring rules marked with a "*" are reasons which I inferred based on a description of the system.)

1. Stevens & Collins (77,82, Collins 77); SCHOLAR and WHY. No reasons are given for these. These rules were gleaned from analysis of Socratic tutoring dialogues. The main pedagogical assumption is that the Socratic technique is effective for teaching specific exemplars, causal dependencies, and reasoning skills.

   (a) If the start of a dialogue then ask about a known case.

   (b) If the student gives an explanation for a factor on a causal chain where there are also prior factors, then ask for the prior factors.

   (c) If the student gives as an explanation one or more factors that are not necessary, then present a counter-example.

   (d) If the student is missing a particular factor, then show an extreme case of this factor.

2. Burton & Brown (82); the WEST tutor:

   (a) Do not tutor until the student has a chance to discover for himself as much of the structure of a situation as possible. Reason*: Constructivist paradigm knowledge is constructed from experiences according to existing knowledge; also, be conservative so as

not to interrupt the student too often, as in when he is making simple slip rather than exhibiting a fundamental misconception or lack of knowledge.

(b) Provide concrete examples of, as well as descriptions of, concepts (or "issues").

(c) Before giving advice, be sure the issue is one in which student is weak. Reason*: Inappropriate advise or criticism inhibits motivation.

(d) After giving advice, allow the student to try to use that advice immediately. Reason: To encode the new information most effectively in "episodic memory."

(e) If a student asks for help, provide several level of hints. Reason: Forces the student to continue to be mentally engaged and gives repeated opportunities for him to discover.

3. Goldstein (82); the WUMPUS tutor:

(a) Assumption: Knowledge evolves along genetic links–from specification to elaboration, deviation to correction, abstraction to refinement, and specialization to generalization.

(b) Give highest priority to teaching skills at the "frontier" of knowledge. Reason*: These are not to difficult or too trivial. "...learning is facilitated by being able to explain a new skill in terms of those already acquired (pg. 64)."

(c) Give multiple explanations for a topic, corresponding to the different links attached to it.

4. Clancey (82) GUIDON:

Clancey describes three types of tutoring rules: 1. for selecting discourse patterns, 2. for choosing domain knowledge, and 3. for maintaining communication module. The rules shown in the paper are quite specific to the domain, and we refer the reader to the article.

5. Brown, Burton & deKleer (82); SOPHIE:

(a) Teach in the context of problem solving using a simulation. Reason: Experiential learning "capitalizes on episodic memory" so

that the experience is "anchored to a personally meaningful context." "The problem solving process gives rise to experience that structure the factual knowledge" (pg. 228).

(b) First watch the expert solve a problem. Reason: Gives the student a "graceful introduction" to the system being studied, and the types of reasoning involved.

(c) Let the student learn form mistakes; Encourage him to "formulate, test, and witness the consequences of his ideas." Reason: "Every time the student makes a wrong prediction, he has an opportunity to go through the "What? That can't be! Aha¡' cycle which improves the accuracy of his world view" (pg. 233).

(d) If the student produces an effect that illustrates a principle that he has just learned, and does not notice the connection, the system should point it out and give opportunity to change directions.

(e) Expose student to alternative ways to solve a problem. Reason: This more correctly demonstrates how an expert solves problems, considering several methods before choosing one.

6. Anderson (Anderson, Boyle, Farrell & Reiser 84); the LISP Tutor and the Geometry Tutor :

(a) Tutor according to the goal structure of problem space. Reason: "Problem solving behavior is organized around a hierarchical representation of goals" in the mind.

(b) Provide instruction in the problem solving context. Reason: "Memories are associated with the features of the context in which they are learned."

(c) Provide immediate feedback for errors. Reason: To make the learning process more efficient; when feedback is delayed, it is harder for the student to properly assign blame to the faulty behavior.

(d) Provide tools which increase the student's effective working memory capacity. Reason: some errors in problem solving are due to working memory overload–minimizing these errors allows the student to focus more efficiently on the domain knowledge.

Appendix B

A KNOWLEDGE TAXONOMY

(A somewhat ad-hoc taxonomy of types of knowledge, showing how such a taxonomy might be structured. Categories actually implemented in a tutoring system should distinguish features of the knowledge that are relevant to tutoring rule decisions.)

1. Primitive knowledge types:

   - needed for complex skills
   - little or no uncertainty
   - student demonstration of knowledge or ignorance is straightforward
   - can be demonstrated out of context–without being 'used'

   (a) facts
      - values (memorized numbers, dates, names, etc.)
      - formulas, laws, rules
      - definitions

   (b) relationships (2-place predicates)

   (c) algorithms, procedures, simple skills

2. Complex knowledge:

   Schemas of primitive knowledge

(a) deep concepts (gestalts)

(b) nexus concepts (structural combination of other knowledge)

(c) heuristic knowledge (fuzzy skills)

(d) runable mental models (with parameters)

(e) ubiquitous non-domain-dependent knowledge (concepts and skills): logic, probability, causality/correlation, estimation, inference skills, conservation.

3. Meta-knowledge

Ancillary knowledge about the primitive or complex knowledge units

(a) source of a knowledge unit (defined, empirical, practical)

(b) limitations and assumptions beneath a knowledge unit

(c) intended uses, purposes of a knowledge unit

(d) role within a larger context

(e) knowledge about learning, refining, verifying the knowledge unit (how to, when to)

(f) personal limitations related to the knowledge unit - working memory limitations, uncertainty of correctness, weak pre-requisites

(g) coordination of multiple interpretations/dimensions (definition, diagram, examples, graphical, textual)

4. Complex skills:

For non-trivial problem solving tasks

(a) analysis

- measurement skills - what (variable isolation), how (precision, accuracy, error), expected worth/relevance (possible outcomes with implications), observation
- data collection and analysis
- classification problem solving
- fault diagnosis
- hypothesis formation, predicting
- theory validation

(b) synthesis

- system design
- programming/algorithm design
- repair, prescription
- theory formation
- organization/taxonomy creation – generalization, refinement
- creativity??

(c) transformation: (–of one problem or representation into another, such as transforming a word problem into an algebraic representation.)

References

[1] Abelson, H. (1982). *Apple LOGO*, Byte/McGraw-Hill, Peterborough, NJ.

[2] Abelson, H., diSessa, A., & Rudolph, L. (1974). "Velocity Space and the Geometry of Planetary Orbits," MIT AI Memo 320.

[3] Ableson,H. & diSessa, A. (1979). *Turtle Geometry*, MIT Press, Cambridge, MA.

[4] Adams, J. (1974). *Conceptual Blockbusting, A Pleasurable Guide to Better Problem Solving*, W. W. Norton & Co., New York.

[5] Alexander, et. al. (1986). "Knowledge Level Engineering: Ontological Analysis," *AAAI-86 proceedings.*

[6] Anderson, J, Farrell, R. & Sauers, R. (1984). "Learning to Program in Lisp," *Cognitive Science*, Vol. 8.

[7] Anderson, J. (1976). *Language, Memory, and Thought*, Erlbaum Assos., Hillsdale, NJ.

[8] Anderson, J. (1980). "A General Learning Theory and its Applications to the Acquisition of Proof Skills in Geometry," CMU tech. report 80-2.

[9] Anderson, J. (1983). *The Architecture of Cognition*, Harvard Univ. Press, Cambridge, MA.

[10] Anderson, J. (1984). "Cognitive Psychology and Intelligent Tutoring," draft paper.

[11] Anderson, J. (1985). "Skill Acquisition: Compilation of Week-Method Problem Solutions," ONR report ONR-85-1.

[12] Anderson, J. & Reiser, B. (1985). "The Lisp Tutor," *BYTE*, April 1985.

[13] Anderson, J. & Skwarecki, E. (1986). "The Automated Tutoring of Introductory Computer Programming," Communications of the ACM, Vol. 29 No. 9.

[14] Anderson, J., Boyle, F. & Reiser, B. (1985). "Intelligent Tutoring Systems," *Science*, Vol. 228.

[15] Anderson, J., Boyle, F. & Yost, G. (1985). "The Geometry Tutor," *IJCAI-85 proceedings.*

[16] Anderson, J., Boyle, F., Farrell, R. & Reiser, B. (1984). "Cognitive Principles in the Design of Computer Tutors," *Proceedings of Cognitive Science Society Conference.*

[17] Arons, A. (1984). "Computer-Based Instructional Dialogs in Science Courses," *Science*, Vol. 224(4653).

[18] Ashley, K. (1984). "Reasoning by Analogy: A Survey of Selected AI Research with Implications for Legal Expert Systems," *Proceedings of the First Ann. Conf. on Law and Technology.*

[19] Ashley, K. & Rissland, E. (1985). "Toward Modeling Legal Argument," aaa UMass Dept. of Computer and Information Science working paper.

[20] Barr, A. & Feigenbaum, E. (Eds.) (1982). "Applications- oriented AI Research: Education," *The Handbook of Artificial Intelligence, Volume 2*, William Kaufman, Inc., Los Altos.

[21] Barr, Cohen, & Feigenbaum (Eds.) (1982). *The Handbook of Artificial Intelligence, Volumes I, II, and III*, William Kaufman, Inc., Los Altos, CA.

[22] Bayman, P. & Mayer, R. (1983). "A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements," Communications of the ACM, Vol. 26 No. 9.

[23] Bobrow, D. et. al. (1986). "CommonLoops, Merging Lisp and Object-Oriented Programming," ACM, September 1996.

[24] Bonar, J. (1982). "Natural Problem Solving Strategies and Programming Language Constructs," UMass Dept. of Computer and Information Science tech. report 82-11.

[25] Bonar, J. (1985). "Understanding the Bugs of Novice Programmers," UMass Dept. of Computer and Information Science tech. report 85-12.

[26] Bonar, J. & Cunningham, R. (1986). "Bridge: An Intelligent Tutor for Thinking About Programming-DRAFT paper," LRDC working paper.

[27] Bonar, J. & Soloway, E. (1983). "The Bridge From Non- Programmer to Programmer," Umass Dept. of Computer and Information Science tech. report 83-18.

[28] Bonar, J. & Soloway, E. (1985). "Pre-Programming Knowledge: A Major Source of Misconceptions in Novice Programmers," *Human-Computer Interaction*, Fall 1985.

[29] Bonar, J., Cunningham, R., & Schultz, J. (1986). "An Object-Oriented Architecture for Intelligent Tutoring Systems," *Proceedings of the First Annual Conf. on Object Oriented Programming, Systems, Languages, and Applications.*

[30] Bork, A. (1985). *Personal Computers for Education*, Harper & Row, Cambridge, MA.

[31] Brachman, R. & Levesque, H. (Eds.) (1985). *Readings in Knowledge Representation*, Morgan Kaufman Publ. Inc., Los Altos, CA.

[32] Brachman, R. & Schmoltze, J. (1985). "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, No. 9.

[33] Brown, D. (1986a). "The Role of Analogies, Metaphors, and Models in Assisting Comprehension," UMass Cognitive Processes Research Group working paper.

[34] Brown, D. (1986b). "Understanding Understanding Mechanical Systems in a Newtonian Framework," UMass Cognitive Processes Research Group working paper.

[35] Brown, D. & Clement, J. (1986a). "An Investigation of the Effectiveness of Logical and Causal Explanations for Overcoming Misconceptions in Mechanics," UMass Cognitive Processes Research Group working paper.

[36] Brown, D. & Clement, J. (1986b). "Overcoming Misconceptions in Mechanics: Theory and Practice–A Literature Review," UMass Cognitive Processes Research Group working paper.

[37] Brown, D., Clement, J., & Murray, T. (1985). "Tutoring Specifications for a Computer Program Which Uses Analogies to Teach Mechanics," UMass Cognitive Processes Research Group working paper.

[38] Brown, J. S. & deKleer, J. (1984). "A Framework for a Qualitative Physics," *Proceedings of Cognitive Science Society.*

[39] Brown, J. S., Burton, R. R., & deKleer, J. (1982). "Pedagogical, Natural Language, and Knowledge Engineering Techniques in SOPHIE I, II, and III," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[40] Brown, J. S., Burton, R., & Clancey, W. (1984). "Applications of Artificial Intelligence to Training and Education," *AAAI 1984 Conference Tutorial Program.*

[41] Bruner, J. (1966). *Toward a Theory of Instruction*, Harvard Univ. Press, Cambridge, MA.

[42] Burton, R. R. (1982). "Diagnosing Bugs in a Simple Procedural Skill," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[43] Burton, R. R., & Brown, J. S. (1982). "An Investigation of Computer Coaching for Informal Learning Activities," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[44] Carbonell, J. R. (1970). "AI and CAI; An Artificial Intelligence Approach to Computer Aided Instruction," *IEEE Transactions on MAN-Machine Systems*, MMS-11(4).

[45] Carr, B. & Goldstein, I. (1977). "Overlays: a Theory of Modelling for Computer Aided Instruction," MIT AI Memo 406.

[46] Chandrasekaran, B. (1985). "Generic Tasks in Expert System Design and Their Role in Explanation of Problem Solving," The Ohio State Univ., Dept. of Computer and Information Science, tech. report.

[47] Chandrasekaran, B. & Josephson, J. (1986). "Explanation, Problem Solving, and New Generation Tools: A Progress Report," The Ohio State Univ. Dept. of Computer and Information Science, tech. report.

[48] Charniac, E., & McDermott, D. (1985). *Introduction to Artificial Intelligence*, Addison-Wesley Publ. Co., Reading, MA.

[49] Chi, M., Feltovich, P. & Glaser, R. (1981). "Categorization and Representation of Physics Problems by Experts and Novices," *Cognitive Science*, (5).

[50] Clancey, W. (1982). "Tutoring Rules for Guiding a Case Method Dialogue," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[51] Clancey, W. (1984). "Teaching Classification Problem Solving," *Cognitive Science Socitey Proceedings.*

[52] Clancey, W. (1985). "Heuristic Classification," *Artificial Intelligence*, January, 1985.

[53] Clancey, W. (1986a). "Qualitative Student Models," *Annual Review of Computer Science*, February 1986.

[54] Clancey, W. (1986b). "From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ONR Final Report 1979-1985," *The AI Magazine*, August, 1986.

[55] Claxton, G. (1985). "Teaching and Acquiring Scientific Knowledge," *Kelly in the Classroom: Educational Applications of Personal Construct Psychology*, T. Keen & M. Pope (Eds.), Cybersystems Inc., Montreal.

[56] Claxton, G. (1986). "Towards a Learning Theory for Science Teachers," *Children's Intuitive Scientific Ideas*, P. Block & A. Lowens (Eds.), Groom Helm Publ. Inc., .

[57] Clement, J. (1982). "Student's Preconceptions in Introductory Mechanics," *American Journal of Physics*, Vol. 50 No. 1.

[58] Clement, J. (1983). "A Conceptual Model Discussed by Galileo and Used Intuitively by Physics Students," *Mental Models*, Gentner & Stevens (Eds.), Lawrence Erlbaum, Hillsdale, NJ.

[59] Clement, J. (1986). "Methods for Evaluating the Validity of Hypothesized Analogies," *Cognitive Science Society proccedings.*

[60] Clement, J. & Brown, D. (1984a). "Finding 'Anchor Beliefs' for Grounding Instruction of Student's Intuitions," UMass Cognitive Processes Research Group working paper.

[61] Clement, J., & Brown, D. (1984b). "Using Analogical Reasoning to Deal With 'Deep' Misconceptions in Physics," UMass Cognitive Processes Research Group working paper.

[62] Clement, J., Lockhead, J. & Soloway, E. (1980). "Positive Effects of Computer Programming on the Student's Understanding of Variables and Equations," *Proceedings of the Assos. for Computing Machinery.*

[63] Cohen, P. (1984a). "Progress Report on a Theory of Endorsements: A Heuristic Approach to Reasoning About Uncertainty," UMass Dept. of Computer and Information Science Tech. Report 84-15.

[64] Cohen, P. (1984b). "Progress Report on the Theory of Endorsements: A Heuristic Approach to Reasoning About Uncertainty," UMass Dept. of Computer and Information Science tech. report 84-15.

[65] Cohen, P. (1985). "Numeric and Symbolic Reasoning About Uncertaity in Expert Systems," UMass Dept. of Computer and Information Science Tech. Report 84-25.

[66] Cohen, P., & Gruber, T. (1985). "Reasoning About Uncertainty: A Knowledge Representation Perspective," UMass Dept. of Computer and Information Science tech. report 85-24.

[67] Cohen, P., Day, D., Delisio, J., Greenberg, M., Kjeldsen, R., Suthers, D. & Berman, P. (1986). "Management of Uncertainty in Medicine," UMass Dept. of Computer and Information Science working paper.

[68] Cohen, P., et. al. (1985). "Representativeness and Uncertainty in Classification Systems," UMass Dept. of Computer and Information Science tech. report 85-26.

[69] Collins, A. (1977). "Processes in Acquiring Knowledge," *Schooling and the Acquisition of Knowledge*, Anderson, Spiro, & Montague (Eds.), Lawrence Erlbaum Associates, Hillsdale, N. J..

[70] Collins, A. (1985). "Component Models of Physical Systems," *Proceedings of Cognitive Science Society.*

[71] Collins, G. (1985). "Teleology + Bugs = Explanations," *Cognitive Science Society conference proceedings.*

[72] Confrey, J. (1985). "A Constructivist View of Mathematics Instruction: A Theoretical Perspective," *Proceedings of the American Educational Research Association.*

[73] Cook, M. (1983). "An Intelligent Tutor for Plane Geometry," UMass Dept. of Computer and Information Science masters project computer program.

[74] Cunningham, P., Iberall, T., & Woolf, B. (1985). "CALEB: An Intelligent Second Language Tutor," UMass working paper.

[75] Davis, R. & Buchanan, B. (1977). "Meta-Level Knowledge: Overview and Applications," *IJCAI-77 Proceedings.*

[76] Davis, R., Buchanan, B., & Shortlife, E. (1977). "Production Rules as a Representation for a Knowledge-Based Consultant Program," *Artificial Intelligence*, 8(1).

[77] DeBono, E. (1970). *Lateral Thinking; A Textbook to Creativity*, Ward Lock Educational Ltd., London.

[78] Dede, C. (1985). "A Review and Synthesis of Recent Research in Intelligent Computer Assisted Instruction," To be published in the International Journal for Man-Machine Studies.

[79] Dede, C. (1986). "Artificial Intelligence Applications to High Technology Training," to be published in the Journal of Educational Communications and Technology.

[80] deKleer, J. & Williams, B. (1986). "Reasoning About Multiple Faults," *AAAI-86 proceedings.*

[81] deKleer, J., & Brown, J. S. (1980). "Mental Models of Physical Mechanisms," Xerox Palo Alto Report CSI-3.

[82] deKleer, J., & Brown, J. S. (1982). "Assumptions and Ambiguities in Mechanistic Mental Models," Xerox Palo Alto Tech. Report.

[83] diSessa, A. (1977). "On 'Learnable' Representations of Knowledge: A Meaning for the Computational Metaphor," MIT A.I. Lab. LOGO Memo No. 47.

[84] diSessa, A. (1979). "On Learning Representations of Knowledge: A Meaning for the Computational Metaphor," *Cognitive Process Instruction*, Lockhead & Clement (Eds.), The Franklin Institute Press, .

[85] diSessa, A. (1985). "Knowledge in Pieces," Address to the Fifteenth Symposium of the Jean Piaget Society.

[86] diSessa, A. & Ableson, H. (1986). "BOXER: A Reconstructible Computational Medium," Communications of the ACM, Vol. 29 No. 9.

[87] Driver, R. (1973). "The Representation of Conceptual Frameworks of Young Adolescent Science Students," Ph.D. dissertation, University of Illinois.

[88] Eisenberg, M. & Peelle, H. (1983). "APL Learning Bugs," Communications of the ACM 8/83.

[89] Elliot, P. (1978). "Computer 'Glass Boxes' as Advanced Organizers in Mathematics Instruction," Int. Jrnl. Math, Science, and Technology Education, Vol. 9 No. 1.

[90] Erman, L., Hayse-Roth, F., Lesser, V. & Reddy, D. (1980). "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computer Survey Report 84-21*, Vol. 12 N0. 2.

[91] Falkenhainer, B., Forbus, K., & Gentner, D. (1986). "The Structure-Mapping Engine," *AAAI-86 proceedings.*

[92] Farrell, R., Anderson, J., & Reiser, B. (1984). "An Interactive Computer-based Tutor for LISP," *AAAI-84 proceedings.*

[93] Flavell, J. (1980). "Speculations About the Nature and Development of Metacognition," *Metacognition, Motivation, and Learning*, Kluwe & Weinert (Eds.).

[94] Forbus, K. (1981). "A CONLAN Primer," BBN report No. 4491.

[95] Forbus, K. (1982a). "Spatial and Qualitative Aspects of Reasoning About Motion," MIT working paper.

[96] Forbus, K. (1982b). "Qualitative Reasoning About Physical Processes," MIT working paper.

[97] Forbus, K. (1983a). "Qualitative Reasoning About Space and Motion," *Mental Models*, Gentner & Stevens (Eds.), Earlbaum Asso., Hillsdale, NJ.

[98] Forbus, K. (1983b). "Measurement Interpretation in Qualitative Process Theory," *IJCAI-83 proceedings.*

[99] Forbus, K. (1984a). "Qualitative Process Theory," MIT Dissertation.

[100] Forbus, K. (1984b). "An Interactive Laboratory for Teaching Control System Concepts," BBN report No. 5511.

[101] Forbus, K. (1985). "The Problem of Existence," *Proceedings of the Cognitive Science Society Conference.*

[102] Forbus, K. & Gentner, D. (1984). "Learning Physical Domains: Towards a Theoretical Framework," *Machine Learning: Recent Progress*, Carbonell & Mitchell (Eds.), Tioga Press, .

[103] Forbus, K. & Gentner, D. (1986). "Causal Reasoning about Quantities," *Proceedings of the Cognitive Science Society.*

[104] Forbus, K. & Stevens, A. (1981). "Using Qualitative Simulation to Generate Explanations," BBN report No. 4490.

[105] Genesereth, M. R. (1982). "The Role of Plans in Intelligent Teaching Systems," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[106] Gentner, D. (1983). "Structure-Mapping: A Theoretical Framework for Analogy," *Cognitive Science*, Vol. 7 No. 2.

[107] Gentner, D. & Stevens, A. (Eds.) (1983). *Mental Models*, Lawrence Erlbaum Asso., Hillsdale, NJ.

[108] Gentner, D. & Toupin, C. (1986). "Systematicity and Surface Similarity in the Development of Analogy," *Cognitive Science*, Vol. 10.

[109] Ginsburg, H. & Opper, S. (1969). *Piaget's Theory of Intellectual Development, An Introduction*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

[110] Glaser, R. (1984). "Education and Thinking: The Role of Knowledge," *American Psychologist*, Vol. 39 No. 2.

[111] Goldberg, F. & McDermott, L. (1984). "Common Sense Knowledge Versus Formal Physics Knowledge in Geometrical Optics," presented at Amer. Assos. of Physics Teachers conference.

[112] Goldstein, I. P. (1982). "The Genetic Graph: A Representation of the Evolution of Procedural Knowledge," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[113] Gordon, W. (1961). *Synectics*, Harper & Row, New York.

[114] Grosz, B. (1986). "A Theory of Discourse Structure," *Proceedings of the Cognitive Science Society.*

[115] Grosz, B. & Sidner, C. (1985). "The Structure of Discourse Structure," *AAAI-85 Proceedings.*

[116] Hayse, P. (1985a). "The Second Naive Physics Manifesto," *Readings in Knowledge Representation*, Brachman & Levesque (Eds.), Morgan Kaufman, Los Altos.

[117] Hayse, P. (1985b). "The Logic of Frames," *Readings in Knowledge Representation*, Brachman & Levesque (Eds.), Morgan Kaufman Publishers, Inc., Los Altos, CA.

[118] Heller, H. & Reif, F. (1984). "Prescribing Effective Human Problem-Solving Processes: Problem Description in Physics," *Cognition and Instruction*, 1 (2).

[119] Horowitz, P. & White, B. (1986). "Thinkertools Annual Progress Report," Bolt Beranek and Newman Technical Report.

[120] Horvitz, E., Heckerman, D., & Langlotz, C. (1986). "A Framework for Comparing Alternative Formalisms for Plausable Reasoning," *AAAI-86 proceedings.*

[121] Joyce, B. & Well, M. (1972). *Models of Teaching*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

[122] Kimbal, R. (1982). "A Self-improving Tutor for Symbolic Integration," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[123] Kulik, C. & Kulik, J. (1986). "Effectiveness of Computer- Based Education in Colleges," to appear in AEDS Journal.

[124] Kulik, C., Kulik, J., & Shwalb, B. (1986). "The Effectiveness of Computer-based Adult Education: A Meta-analysis," Jrnl. Educational Computing Research, Vol. 2(2).

[125] Kulik, J. (1985). "Consistencies in Findings on Computer- Based Education," presented to the annual meeting of the American Educational Research Association.

[126] Lakatos, I. (1976). *Proofs and Refutations, The Logic of Mathematical Discovery*, Cambridge Univ. Press, Cambridge, MA.

[127] Larkin, J. (1980). "Spatial Reasoning in Solving Physics Problems," CMU working paper.

[128] Larkin, J. (1983a). "A General Knowledge Structure for Learning or Teaching Science," *Classroom Computers and Cognitive Science*, Wilkinson (Eds.), Academic Press, .

[129] Larkin, J. (1983b). "The Role of Problem Representation in Physics," *Mental Models*, Gentner & Stevens (Eds.), Erlbaum Asso., Hillsdale, NJ.

[130] Lawson, A. & Wollman, W. (1975). "Physics Problems and the Process of Self-Regulation," *The Physics Teacher*, Nov. 1975.

[131] Leant, D. (1983). "On Automated Scientific Theory Formation: A Case Study Using the AM Program," *Machine Intelligence*, VOl. 9.

[132] Leiberman, H. (1980). "There's More to Menu Systems Than Meets the Screen," MIT tech. report.

[133] Leiberman, H. (1985). "An Example Based Environment for Beginning Programmers (Draft)," MIT report.

[134] Lesser, V. (1984). "Control in Complex Knowledge-based Systems," Tutorial at the IEEE Computer Society AI conference.

[135] Littman, D., Pinto, J., & Soloway, E. (1986). "An Analysis of Tutorial Reasoning About Programming," *AAAI-86 proceedings.*

[136] Lockhead, J. & Clement, J. (Eds.) (1979). *Cognitive Process Instruction: Research on Teaching Thinking Skills*, The Franklin Institute Press, .

[137] London, B. (1986). "Integrated, Multiple Viewpoints in Diagnostic Student Modelling," *submitted to AAAI-86.*

[138] Malone, T. & Levin, J. (Eds.) (1981). "Microcomputers in Education: Cognitive and Social Design Principles," working paper.

[139] Mats, M. (1982). "Towards a Process Model for High School Algebra Errors," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[140] McCloskey, M. (1983). "Intuitive Physics," *Scientific American*, Vol. 48.

[141] McDermott, L. (1984). "Research on Conceptual Understanding in Mechanics," *Physics Today*, July 1984.

[142] McDonald, D., Brooks, J., Woolf, B., & Werner, P. (1986). "Transition Networks for Discourse Management," UMass Dept. of Computer and Information Science working paper.

[143] Michener, E. R. (1978). "Understanding Understanding Mathematics," *Cognitive Science*, Vol. 2, No. 4.

[144] Miller, M. L. (1982). "A Structured Planning and Debugging Environment for Elementary Programming," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[145] Minsky, M. (1981). "A Framework for Representing Knowledge," *Mind Design*, J. Haugeland (Eds.), MIT Press, Cambridge, MA.

[146] Murray, T. & Woolf, B. (1986). "Knowledge Representation in a Physics Tutor," UMass Dept. of Computer and Information Science technical report 86-37.

[147] Murray, T., Clement, J., & Brown, D. (1985). "A Computer Program which Utilizes a 'Bridging Analogies' Tutoring Strategy for Dealing with Misconceptions in Mechanics," Cognitive Processes Research Group Tech. Report.

[148] Murray, T., Woolf, B., Brown, D., Clement, J., & Schultz. K. (1986). "Tutorial Strategies for Computers which Deal with Misconceptions in Physics," submitted to Third Int. Conf. on AI and Education.

[149] Nickerson, R., Perkins, D. & Smith, E. (1985). *The Teaching of Thinking*, Lawrence Erlbaum Asso., Hillsdale, NJ.

[150] Nii, H. Penny (1986). "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *The AI Magazine*, Summer-86.

[151] Novak, G. (1977). "Representations of Knowledge in a Program for Solving Physics Problems," *IJCAI-77 Proceedings.*

[152] Novak, G. & Araya, A. (1980). "Research on Expert Problem Solving in Physics," *Proceedings of the First National Conf. on AI.*

[153] O'Shea, T. (1982). "A Self-improving Quadratic Tutor," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[154] O'Shea, T. & Self, J. (1983). *Learning and Teaching With Computers: Artificial Intelligence in Education*, Prentice-Hall Inc., Englewood Cliffs, N.J..

[155] Osborne, R. (1984). "Children's Dynamics," *The Physics Teacher*, Nov. 1984.

[156] Palies, O., Caillot, M., Cauzinille-Marmeche, E., Lauriere, J., & Mathieu, J. (1980). "Student Modelling by a Knowledge-based System," to be in Computational Intelligence, AI and Education issue.

[157] Papert, S. (1980). *MindStorms-Children, Computers, and Powerful Ideas*, Basic Books, Inc., New York.

[158] Park, O., Perez, R., and Seidel, R. (1986). "Intelligent CAI: Old Wine in New Bottles - Or, Is It Just Vinegar?," *Artificial Intelligence and Instruction: Application and Methods*, Kearsley (Eds.), Addison-Wesley, Reading, MA.

[159] Pea, R. (1983). "LOGO Programming and Problem Solving," Bank Street College tech. report 12, April 1983.

[160] Peelle, H. A. (1982). "Computer Vitamins," *Educational Computing Magazine*, September 1982.

[161] Peelle, H. A. (1986). "What is 'APL Thinking'?," *Proceedings of the APL86 conference.*

[162] Peelle, H. A. & Eisenberg, M. (1983). "APL Teaching Bugs," UMass School of Education working paper.

[163] Perkins, D. (1986). *Knowledge as Design*, Lawrence Erlbaum Asso., Hillsdale, NJ.

[164] Piaget, J. (1972). *The Principles of Genetic Epistemology*, Basic Books, NY.

[165] Polya, G. (1973). *How To Solve It*, Princeton University Press, New Jersey.

[166] Reif, F. & St. John, M. (1979). "Teaching Physicists' Thinking Skills in the Laboratory," *Amer. J. Phys.*, 47(11), Nov..

[167] Reisberg, D. & Chambers, D. (1986). "Neither Pictures Nor Propositions: The Intentionality of Mental Imagery," *Cognitive Science Society Proceedings.*

[168] Reiser, B., Anderson, J., & Farrell, R. (1985). "Dynamic Student Modeling in an Intelligent Tutor for Lisp Programming," *IJCAI-85 Proceedings.*

[169] Richer, M & Clancey, W. (1985). "Guidon-Watch: A Graphic Interface for Viewing a Knowledge-Based System," IEEE CG&A, November, 1985.

[170] Rissland, E. (1978). "Understanding Understanding Mathematics," *Cognitive Science*, Vol. 2 No. 4.

[171] Rissland, E. (1980). "The Structure of Knowledge in Complex Domains," UMass Dept. of Computer and Information Science tech. report 80-07.

[172] Rissland, E. (1981). "Constrained Example Generation," UMass Dept. of Computer and Information Science tech. report 81-24.

[173] Rissland, E. (1982). "Examples in the Legal Domain: Hypotheticals in Contract Law," *Proceedings of Cognitive Science Society.*

[174] Rissland, E. (1983). "A.I. and the Learning of Mathematics," UMass Dept. of Computer and Information Science tech. report 83-40.

[175] Rissland, E. (1984a). "Argument Moves in Hypotheticals," *Proceedings of the First An. Conf. on Law and Technology.*

[176] Rissland, E. (1984b). "Learning How to Argue: Using Hypotheticals," UMass Dept. of Computer and Information Science working paper.

[177] Rissland, E. & Ashley, K. (1980). "Hypotheticals as Heuristic Device," working paper.

[178] Rissland, E. R., Valcarce, E., & Ashley, K. (1984). "Explaining and Arguing With Examples," *AAAI-84 Proceedings.*

[179] Sacerdoti, E. (1974). "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, 5(2).

[180] Samples, B. (1978). *The Metaphoric Mind-A Celebration of Creative Consciousness*, Addison-Wesley, Reading, MA.

[181] Schank, R., & Childers, P. (1984). *The Cognitive Computer, On Language, Learning, and Artificial Intelligence*, Addison- Wesley Publ. Co., Reading, MA.

[182] Schank, R., & Riesbeck, C. (1981). *Inside Computer Understanding*, Lawrence Erlbaum Asso., Publ., Hillsdale, NJ.

[183] Schoenfeld, A. (1983). "Metacognition and Epistemological Issues in Mathematical Understanding," Univ. of Rochester working paper.

[184] Schwartz, J. (1980?). "The Geometric Supposer," (Software for APPLE computers).

[185] Schwebel, M. & Raph, J. (1973). *Piaget in the Classroom*, Basic Books, Inc., New York.

[186] ServiB, D. & Woolf, B. (1986). "E.T.: An Architecture for Error-Triggered Remediation In Intelligent Tutoring Systems," Presented at the 1987 AI in Education conference.

[187] Shapiro, S. & Rapaport, W. (1986). "SNePS Considered as a Fully Intentional Propositional Semantic Network," *AAAI-86 proceedings.*

[188] Shavlik, J. & Dejong, G. (1985). "Building a Computer Model of Learning Classical Mechanics," *Proceedings of the Cognitive Science Society.*

[189] Sherwood, B. & Kane, D. (1980). "A Computer-based Course in Classical Mechanics," *Computers and Education*, Vol. 4, Pergamon Press Ltd, .

[190] Shute, V. & Bonar, J. (1986). "Intelligent Tutoring Systems for Scientific Inquiry Skills," *Proceedings of the Cognitive Science Society.*

[191] Simpson, R. (1985). "A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation," Ph.D. Thesis, School of Info. & Computer Sci., Georgia Inst. of Technol..

[192] Slater, J., Petrossian, R., & Shyam-Sunder, S. (1985). "An Expert Tutor of Rigid Body Mechanics: Athena Cats - MACAVITY," IEEE.

[193] Sleeman, D. (1982). "Assessing Aspects of Competence in Basic Algebra," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[194] Sleeman, D., & Hendley, R. J. (1982). "ACE: A System Which Analyses Complex Explanations," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[195] Sleeman, D., Brown, J. S. (Eds.) (1985). *Intelligent Tutoring Systems*, Academic Press, New York, NY.

[196] Soloway, E. (1986). "Learning to Program = Learning to Construct Mechanisms and Explanations," Communications of the ACM, Vol. 29 No. 9.

[197] Soloway, E. & Johnson, L. (1984). "Remembrance of Blunders Past: A Retrospective on the Development of Proust," *Proceedings of the Cognitive Science Society.*

[198] Soloway, E., Woolf, B., Rubin, E., & Boarth, B. (1981). "MENO-II: An Intelligent Tutoring System for Novice Programers" IJCAI-81 Proceedings.

[199] Solso, R. (1979). *Cognitive Psychology*, Harcourt Brace Jovanovich, Inc., New York.

[200] Steele, G. (1984). *Common LISP, The Language*, Digital Press, .

[201] Stefik, M. & Bobrow, D. (1986). "Object Oriented Programming: Themes and Variations," *The AI Magazine*, Winter-86.

[202] Stevens, A. & Collins, A. (1977). "The Goal Structure of a Socratic Tutor," BBN Report No. 3518.

[203] Stevens, A., Collins, A., & Goldin, S. E. (1982). "Misconceptions in Students' Understanding," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[204] Sullivan, M. & Cohen, P. (1984). "An Endorsement-Based Plan Recognition Program," UMass Dept. of Computer and Information Science tech. report 84-33.

[205] Tennyson, R. & Park, O. (1980). "The Teaching of Concepts: A Review of Instructional Design Research Literature," *Review of Educational Research*, Vol. 50 No. 1.

[206] Tversky, A. & Kahneman, D. (1974). "Judgement Under Uncertainty: Heuristics and Biases," Science, Vol. 185.

[207] Van Lehn, K. (1983). "Felicity Conditions for Human Skill Acquisition: Validating an AI-Based Theory," Xerox Palo Alto Research Center Technical Report CIS-21.

[208] Varma, V. & Williams, P. (Eds.) (1976). *Piaget, Psychology and Education*, F. E. Peacock Publ., Inc., Itasca, IL.

[209] Wenger, E. (1986). *Knowledge Communication Systems*, to be publ. by Morgan & Kaufman, Inc., Los Altos, CA.

[210] Weyer, S. (1982). "Searching for Information in a Dynamic Book," Xerox Palo Alto tech. report no. SCG-82-1.

[211] Whimbley, A. & Lockhead, J. (1982). *Problem Solving and Comprehension*, The Franklin Institute Press, Philadelphia.

[212] White, B. (1984). "Designing Computer Games to Help Physics Students Understand Newton's Laws of Motion," *Cognition and Instruction.*

[213] White, B. & Frederiksen, J. (1986a). "Intelligent Tutoring Systems Based Upon Qualitative Model Evolution," *AAAI-86 Conference Proceedings.*

[214] White, B. & Frederiksen, J. (1986b). "Progressions of Qualitative Models as a Foundation for Intelligent Learning Environments," BBN report No. 6277.

[215] White, B. & Frederiksen, J. (1986c). "Qualitative Models and Intelligent Learning Environmetns," *to appear in AI and Education*, Lawler and Yazdani (Eds.), Ablex Publ. Corp., .

[216] Winston, P. H. (1984a). *LISP*, Addison-Wesley Publ. Co., Reading, MA.

[217] Winston, P. H. (1984b). *Artificial Intelligence*, Addison Wesley Publ. Co., Reading, MA.

[218] Woods, W. (1970). "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, Vol. 13, No. 10.

[219] Woolf, B. (1984). "Context Dependent Planning in a Machine Tutor," UMass Dept. of Computer and Information Science technical report 84-21.

[220] Woolf, B., Blegen, D., Verloop, A. & Jansen, J. (1986). "Tutoring a Complex Industrial Process," *AAAI-86 Proceedings.*

[221] Woolf, B. & McDonald, D. (1983). "Human-Computer Discourse in the Design of a PASCAL Tutor," Umass Dept. of Computer and Information Science tech. report 83-27.

[222] Woolf, B. & McDonald, D. (1984). "Building a Computer Tutor: Design Issues," *IEEE Computer*, September 1984.

[223] Woolf, B. & McDonald, D. (1985). "Understanding Discourse Conventions in Tutoring," UMass Dept. of Computer and Information Science tech. report 85-22.

[224] Woolf, B. & Murray, T. (1987). "A Framework for Representing Tutorial Discourse," IJCAI-87.

[225] Woolf, B., Pustejovsky, J., McDonald, D. & Lander, S. (1980). "Discourse Processing in a Knowledge Acquisition Dialog," UMass Dept. of Computer and Information Science working paper.

[226] Xerox Special Information Systems, Vista Laboratory (1985). "CALLE Project Report," .

[227] Yankelovich, N., Meyorowitz, N., & van Dam, A. (1985). "Reading and Writing the Electronic Book," *Computer*, October, 1985.

[228] Yazdani, M. (1986). "Intelligent Tutoring Systems: An Overview," *Expert Systems*, Vol. 3. No. 3.